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：

导　　师：

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 26 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:

School of Software

Shanghai Jiao Tong University

Shanghai, P.R. China

November 26th, 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

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——为多个行业的智能化升级奠定了坚实的基础。

在这一大背景下，随着大语言模型（Large Language Model, LLM）的快速发展，其在智能系统构建中的应用价值愈发显著。通过将 LLM 与不同领域的工具集成，能够有效提升用户体验和系统效率。而作为连接不同系统的通信桥梁，API（Application Programming Interface，应用程序编程接口）是智能系统集成的重要工具。通过 API，开发者可以调用服务功能或数据而无需了解其具体实现，为实现智能系统提供便利。

在消费出行等复杂场景中，大语言模型驱动的智能体可以集成多种有关 API，自动识别用户需求并灵活调用工具，为用户提供高效解决方案。这种多工具协同能力让智能系统更能适应复杂任务。而在这种协作机制中，编排技术起到了关键作用。

编排的核心将不同的工具、模块或服务按照任务目标进行组织和协调。编排的作用在于将复杂任务拆解为多个子任务，并通过工具之间的逻辑关系和数据流动完成整体任务。它不仅是实现多工具集成的关键技术，也是提高系统自动化和智能化水平的重要途径。通过编排，系统能够在单一工具功能的基础上实现更高层次的组合功能，同时确保各工具间的数据交互和逻辑调用的正确性。在面对动态变化的用户需求或复杂的任务逻辑时，编排技术尤其能够提升任务执行的效率和准确性。

传统的编排实现中，不同工具之间的接口适配依赖人工定义和开发。开发者需要明确设计工具之间的数据传递逻辑，并手动编写适配代码来实现数据格式的转换和兼容。这种方式不仅耗时费力，而且难以适应复杂多变的场景。

引入大语言模型后，编排技术在理解复杂任务需求和动态生成工具调用路径方面得到了显著增强。大语言模型凭借其强大的语义理解和生成能力，能够在缺乏明确工具调用路径定义的情况下，基于用户目标实时推断任务执行流程。它可以识别工具间的依赖关系，优化工具调用顺序，并根据任务上下文动态调整工具的输入输出格式。通过自动生成接口适配逻辑，大语言模型大幅降低了人工编写代码的复杂性，降

低了编排的技术门槛。同时，LLM 还能根据环境的动态变化实时调整任务规划，使系统更高效地应对需求的不确定性和变化。

然而，大语言模型在工具编排领域的应用仍然面临着以下两方面的问题。

- 针对大规模工具集时的选择: 当面对大规模的工具候选集时，大语言模型的推理和规划能力存在显著不足。首先，随着工具数量的增加，模型在从众多工具中筛选出合适工具组合时，选择的难度急剧上升，导致容易选择错误的工具或者是出现”幻觉“现象，如选择不存在的工具。此外，大语言模型缺乏对多步骤复杂任务的全局规划能力，其生成的调用路径通常停留在局部优化阶段，而无法兼顾任务的整体逻辑完整性和效率，难以支撑高效的工具协作。
- 对复杂工具依赖关系的处理能力不足: 多工具协作的任务中，工具之间复杂的依赖关系进一步增加了模型编排的难度。模型在识别工具间隐式依赖时能力有限，例如，当一个工具的输出需要经过特定处理才能作为另一个工具的输入时，大语言模型难以仅从工具文档中捕捉到这一逻辑，导致工具无法正确执行。此外，工具调用顺序的动态管理也存在不足，在依赖关系随上下文变化的情况下，模型难以灵活调整调用顺序。

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工具调用与工具图谱的研究进展。首先，本节会介绍大语言模型智能体工具调用方面的研究，包括不同的流程搭建和提示词工程的方法，详细介绍了各种提升模型在工具编排、调用方面的能力。其次，本节还会介绍有关图谱与大语言模型结合的应用场景，特别是在工具图上的应用，比如通过对工具之间的时序关系、资源依赖关系建模，从而得到工具调用路径的现有方法。

1.2.1　大语言模型智能体工具调用

为大型语言模型（LLMs）引入外部工具显著增强了智能体在应对复杂现实任务时的能力[1-3]。通过支持功能调用，LLMs 能够获取最新信息、提升专业技能、执行精确计算并调用第三方服务，同时功能调用也能提高回答生成过程中的透明性和鲁棒性，让回答更具有可解释性和可靠性。因此大语言模型智能体在多个领域实现了广泛的应用，例如多媒体内容搜索[4]、财务分析[5]以及旅行规划[6]。

然而，要充分发挥功能调用的潜力并高效完成复杂任务，必须应对功能调用本身所带来的技术挑战。这些挑战并不仅限于简单的接口调用，而是伴随着实际应用场景

的多样化需求和复杂性。例如，从如何管理多种 API 的协作，到如何优化工具调用的顺序和数据依赖关系，这些都对大语言模型的能力提出了更高的要求[1-2]。为深入理解这些问题，我们需要分析功能调用在实际应用中的复杂性及其解决思路。

功能调用的复杂性主要来自实际应用中 API 的多样性、复杂性及其相互依赖的特性[2]。例如，现实场景中的 API 参数往往不仅限于简单的字符串或数字，还可能包括列表、字典、嵌套结构，甚至这些类型的组合。参数的数量可以从零到几十不等，其应用领域覆盖了多个行业和业务场景[7]。此外，为完成一项任务，通常需要多个工具协同工作，单一 API 难以满足复杂任务的需求[1]。更复杂的是，一个 API 的输入可能依赖于另一个 API 的输出[2]，进一步增加了功能调用的挑战性和复杂性。

在大语言模型工具调用上，可以通过微调和非微调的方式来提升大语言模型的工具调用能力。[2,8-10]等研究者通过微调开源的大语言模型来增强模型的工具能力。然而这些方法通常需要额外的工具调用数据集来进行参数微调，难以扩展到闭源的黑盒 LLMs。并且该方法在以“即插即用”方式集成外部工具方面缺乏灵活性。因此，本方案不涉及到对大语言模型的微调，我们仅介绍不需要参数微调的实现方式。

大语言模型的工具调用流程通常可以划分为以下四个核心阶段[4,11-12]：任务规划、工具选择、工具调用和响应生成。尽管其他框架可能存在差异，但通常也是在此基础上进行修改、合并或删减。接下来，我们将按照这些模块化分层依次展开介绍。

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，如何识别用户意图并明确定义好任务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，并对复杂任务进行拆解。

现有研究[13]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[12]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[4]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。

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。基于关

键词的检索：通过精确匹配实现用户需求和文档之间的对齐和查询，如 TF-IDF[14]和 BM25[15]；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[2]，在其中作者们对 BERT 模型在工具数据集上进行微调，提升了其在工具检索场景上的能力，进一步增强了基于语义检索的准确性。

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信息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。

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）[16]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[17]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）[18]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）[19]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[20]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[21]中指导大模型按照指定格式来思考和行动。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[22]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[23]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[13]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[24]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，直到走到标记为结束节点的节点。ToolLLM[2]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[25]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。

在真实场景中，工具的数量通常非常庞大。如果将所有工具的描述都作为 LLM

的输入，会面临上下文窗口长度限制和模型生成时间延迟的限制。因此，近期的研究越来越多地关注先通过检索器筛选工具[2,24,26]，再让大语言模型进行选择。

1.2.1.3 工具调用

在工具调用部分，大语言模型主要负责根据工具描述文件来提取工具调用所需的参数，并对工具服务器请求服务。这一过程要求 LLMs 不仅能够正确提取参数的内容和格式，还必须严格遵循规定的参数输出来输出。

无需微调的方法主要通过提示词工程、多智能体协作来提升工具调用能力。基于提示词工程的方法利用少样本（few-shot）技术提供参数提取的示例，从而增强 LLM 提取参数的能力[4,24,27-28]。Reverse Chain[29]采用逆向思维，首先为任务选择最终工具，然后让 LLM 填写所需的参数；如果缺少参数，则基于描述选择额外的工具来补全参数，最终完成任务。EasyTool[30]通过提示 ChatGPT 重写工具描述，使其更简洁并直接包含工具功能的使用指南，增强 LLM 对工具功能和参数需求的理解。ConAgent[31]引入了一种多智能体协作框架，其中一个专门的执行智能体负责任务执行和工具调用。

1.2.1.4 响应生成

由于工具输出形式多样且复杂，这些结果可能以文本、数字、代码等不同格式呈现，因此无法直接提供给用户查看。需要通过大语言模型（LLM）对这些输出进行组织和分析，围绕用户的查询提取相关信息，并结合模型自身的知识生成完整且清晰的回答。根据将工具输出融入大语言模型提示词的方式，现有方法主要分为直接插入方法和信息整合方法两类。

直接插入方法是早期研究中常见的一种实现方式[8-9,32]。模型会首先输出带有占位符的回答，在得到工具调用结果后将占位符替换为工具结果。然而，由于工具输出结果的形式和内容往往难以预测，这种方法可能会影响用户体验。

为了克服这一问题，许多研究选择将工具输出作为上下文的一部分输入到 LLM，以便生成更高质量的响应[33]。不过，由于 LLM 的上下文长度有限，对于某些工具输出内容，直接输入会面临挑战。为此，不同方法提出了多种解决方案。例如，RestGPT[4]通过预定义模式（schema）来简化冗长的工具输出，这些模式包括示例、格式和可能错误的说明文档。ToolLLaMA[2] 采用截断策略，将输出裁剪到适当的长度范围内，但可能因此丢失解决用户问题的关键信息。ReCOMP[34] 提出了一种压缩机制，可以将冗长的工具输出浓缩为简洁的版本，仅保留核心内容。

1.2.2　大语言模型与工具图谱结合的应用

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：

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；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；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[35]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[24,36-37]。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[24]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[36-37]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。GNN4TaskPlan 中[38]通过构建工具图和使用图神经网络，实现了非训练和基于训练的两种方式，使用图神经网络与当前流行的提示词工程的方法形成了互补，能够有效提升大语言模型在任务规划上的准确性。ControlLLM[27]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。

尽管现有工作尝试将工具图谱与大语言模型结合，但仍存在显著局限。首先，工具的规模和场景较为局限，如 TaskBench[39]的工具节点数较少，而 ControlLLM[27]仅

针对多媒体 API，缺乏对大规模的通用工具场景的覆盖。其次，工具图谱中关系建模简单，工具图谱中一般只存在一种工具之间的关系，如时序关系[24]或者媒体资源依赖关系[27]，未将工具之间的不同种依赖关系建模在同一张图。最后，图谱构建策略过于依赖人工或调用数据，未充分利用工具描述文档中丰富的参数和语义信息，导致图谱表达力和适用性不足。这些问题限制了方法在复杂任务场景中的通用性和实用性，有较大的改进空间。

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：

- **如何通过工具调用路径数据构建一个高效且结构化的工具图谱？**本问题旨在研究如何筛选、清洗现有的工具调用路径数据，并构建一个大型工具图谱，从而将路径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路径数据构建大型工具图谱的方法，旨在辅助智能工具搜索与调用流程。首先，我们通过设计两种数据筛选策略，提取高质量的工具组与有效调用路径。其次，我们定义了工具图谱概念模型和层次关系，建模工具之间的时序与资源依赖逻辑。最终，构建并导入 Neo4j 数据库的高质量工具图谱，实现了可视化与高效查询，为复杂任务的工具调用与规划提供支持。
2. 为了有效利用构建的工具图谱的知识，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，我们提出了记忆框架，通过维护记忆知识，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

如图1.1所示，本论文的内容组织结构分为以下几章：

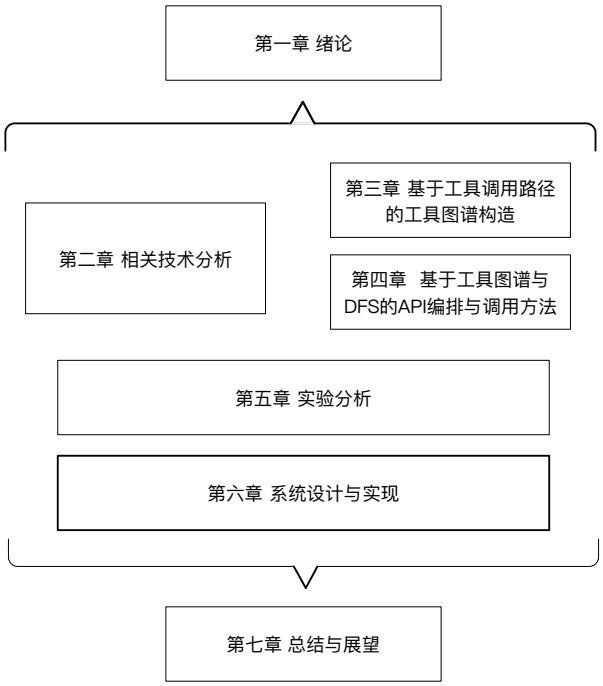

图 1.1　论文组织结构
Figure 1.1　Structure of the Paper

第一章为绪论，本章从研究背景出发，简要阐述了研究的问题与难点，分析了国内外的研究现状，介绍了本文的主要工作内容，并对全文的组织结构进行了概述。

第二章为相关理论与技术，本章主要介绍了研究相关的核心概念与技术背景。首先，介绍了知识图谱的定义、分类及基于 API 的知识图谱研究；其次，阐述了大语言

模型的定义、发展历程及其关键技术，包括大模型智能体、提示词工程和检索增强技术等；最后，探讨了大语言模型与图谱结合的应用方案及其典型案例。

第三章为基于工具调用路径的工具图谱构建方法及实现，本章提出了一种利用工具调用路径数据构建大型工具图谱的方法。内容包括数据筛选与清洗、图谱概念模型的设计和图谱数据的抽取，为后续基于大语言模型智能体的工具编排与执行流程提供支持。

第四章为基于工具图谱和智能体的工具编排和调用方法的设计与实现，本章聚焦于基于大语言模型智能体的工具任务解决方案，涵盖任务分解、工具选择、工具调用及工具总结等关键流程。

第五章为实验分析，本章通过对 API 检索器模型进行实验，验证了微调模型在工具检索中召回率和 NDCG 指标的提升。随后，对整体的 API 编排与调用流程进行了评估，结果与多个基线方法进行对比，证明了本文方法的可行性与优越性。此外，通过消融实验，进一步验证了系统中各模块的必要性及贡献。

第六章为系统设计和实现，本章在前几章的基础上完成了系统的架构设计和功能模块实现，最终构建了可视化界面及任务流程平台。具体内容包括系统框架设计、关键功能模块开发及系统展示。

第七章为总结与展望，本章回顾了全文的研究工作，概括了主要成果，并针对本研究的局限性提出改进方向，展望了未来在知识图谱与大语言模型结合领域的研究前景。

第 2 章　相关技术分析

2.1　传统服务编排

随着互联网技术的快速发展，用户对软件系统需求日益增加。为平衡服务稳定性和需求灵活性，微服务架构应运而生。相比于面向服务的体系架构（Service Oriented Architecture，SOA），微服务架构更注重服务的独立性。每个微服务作为独立单元开发、部署、扩展，通过定义明确的接口提供服务。这种架构大幅提升了软件系统的灵活性与可扩展性。

2.1.1　服务编排模式

在微服务架构中，多个服务协作完成完整业务流程的过程称为服务编排。常见的编排方式有两种：

- **服务编制**：中心化模式，控制中心定义服务的执行流程，各服务无需了解具体组合顺序。

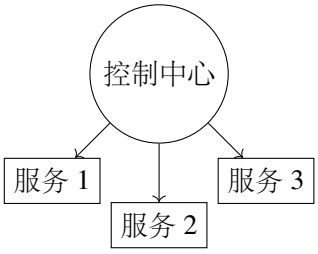

图 2.1　服务编制模型

- **服务编排**：去中心化模式，服务通过消息队列等机制相互通信，完成资源与信息的交换。

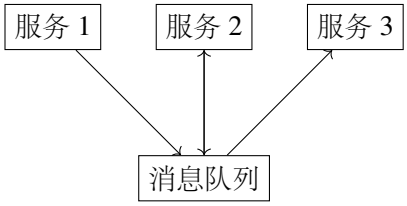

图 2.2　服务编排模型

两种方式都旨在组合与协调微服务，提供复杂业务支持。

2.1.2 服务编排实现方式

微服务架构提出后，涌现了许多编排工具与语言，例如 BPEL 和 BPMN。

- **BPEL**（Business Process Execution Language）：基于 XML 的标准服务编制语言，用于定义业务流程的抽象和可执行方案。
- **BPMN**（Business Process Model and Notation）：业务流程建模语言，简化了流程设计与实施的沟通，适合流程可视化建模。

此外，Netflix Conductor、Zeebe 等流程引擎，以及 Activiti、jBPM 等开源框架，提供了更多编排支持。实际应用中，许多项目采用 BPMN 或 BPEL 结合工作流框架构建微服务系统。

然而，传统方法面临以下挑战：

1. **复杂性与适应性**：如 BPEL 需清晰接口定义，但许多现实 API 接口不规范，若要转为规范定义需人工参与。
2. **快速迭代的需求**：由于持续集成等概念的流行，服务编排不仅需要能够静态地组合定义规范的服务，还需要支持快速组合新加入的服务。
3. **高门槛与资源消耗**：对于众多且不断变化的业务需求，每次都需要开发人员全程参与，对微服务进行明确定义、对业务流程重新进行编排，需要耗费大量人力。

本项目通过结合大语言模型（LLM）的语义解析能力与知识图谱技术，提出低门槛的服务编排方案。利用 LLM 对用户需求自动解析，结合知识图谱自动生成服务调用流程，减少开发者参与和人力成本。最终目标是实现用户需求驱动的智能化服务编排，提升效率与灵活性，同时降低学习和部署成本。

2.2 知识图谱

2.2.1 知识图谱的简介与定义

知识图谱是一种对现实世界中知识和概念进行建模的技术。虽然“知识图谱”这一术语早在 1972 年便已出现[40]，但现代意义上的知识图谱概念起源于谷歌在 2012 年发布的 Google Knowledge Graph[41-42]。知识图谱采用基于图的数据模型，适用于需要整合、管理和从多样化的数据中提取价值的应用场景[43]。最初，知识图谱旨在增强搜索引擎的理解能力，为用户提供更加智能化的搜索体验。然而，随着技术的不断发展，知识图谱如今已被广泛应用于人工智能领域，包括语义搜索、推荐系统、大数据分析、智能问答系统等。

在知识图谱中，知识以结构化图的形式表示，其中节点用于表示实体，如人物、地点、事件或抽象概念，而节点之间的边则表示实体间的语义关系或逻辑关联。知识图谱的核心单元是三元组 (subject, predicate, object)，即“头实体-关系-尾实体”。这种形式化的表达方式使得知识可以在不同系统中共享、分析和推理，为各类人工智能应用提供支持。

知识图谱可以形式化地定义为一个有向图 $G = (V, E, R)$，其中：

- V 是节点集合，表示知识图谱中的实体或概念；
- $E \subseteq V \times R \times V$ 是边集合，表示实体之间的关系；
- R 是关系集合，定义了节点之间可能的语义或逻辑关系。

每条边 $(h, r, t) \in E$ 表示一个三元组，其中：

- $h \in V$ 是头实体（head entity）；
- $r \in R$ 是实体之间的关系（relation）；
- $t \in V$ 是尾实体（tail entity）。

通过这种结构化的三元组表示，知识图谱能够捕获实体及其相互间的复杂关系，支持语义推理、知识查询和知识增强等应用。

2.2.2　知识图谱的种类

知识图谱的发展大致经历了四个阶段[44]：

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）。

2.2.3 知识图谱的应用

2.3 大语言模型

2.3.1 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，如：GPT 系列[45-46]，Claude 系列[47]，Gemini[48-49]等等，这些模型的权重不开放给开发者，因此只能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[50]，ChatGLM[51]，Qwen[52]，LLaMA[53]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，也可以部署在本地 GPU 服务器上。

2.3.2 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（Prompt）来从大语言模型中提取知识的过程，已成为提升预训练大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。我们可以根据是否提供自然语言文本的参考样本将提示词方法分为以下两种。

- **零样本提示（Zero-Shot Prompting）**：在零样本提示设置（Zero-Shot，Radford 等，2019）中，大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。

除了根据提示词中的样本数量分类，许多提示词会规定大语言模型用特定的模式来进行，以下是几种针对提升大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）**：[16]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。

2.3.3 大语言模型智能体

大语言模型也可以作为“Agent”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[54]。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”[55]。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[56]。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[57]，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[58-59]。这些能力在智能体中非常有用，因此衍生出了许多基于大模型的智能体研究[4,11,60-61]，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[62]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[63]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。[58]中将大模型常用的记忆方式分为两种：仅使用短期记忆，和记忆混合方式。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。

2.3.4　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[64-65]。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。
- **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根据实体之间的关系来进行检索，如 k 跳邻居[66-67]。

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。

2.3.5 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[68]，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[69]，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[35,42,70]，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在图上进行搜索[37]得到准确性知识。

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[71]，又可以增强大语言模型的可解释性[72]。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[73]和推理[74]。

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。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 HNSW 来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计算方式，包括欧式距离、余弦相似度和点积。

2.4.3 LangChain

LangChain 是一个专为开发大语言模型（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 LangGraph

LangGraph 是一个灵活的库，在 LangChain 的基础上，通过引入图结构，使开发者可以设计更加复杂的工作流和多 Agent 架构。它的关键特点包括：它支持循环、分支控制和状态持久化，能够实现更复杂的代理工作流。LangGraph 的核心功能包括循环与条件分支的实现、自动状态保存、人机协作、以及流式输出支持。LangGraph 可与 LangChain 无缝集成，也可独立使用。

2.4.5 本章小结

本章主要介绍了与本课题相关的技术背景。首先阐述了知识图谱的起源与定义，并概述了不同类别的知识图谱及其应用方式。接着，介绍了大语言模型的基本概念、核心技术，以及其在智能体、提示词工程和检索增强生成中的具体应用。最后，讨论了如何将知识图谱与大语言模型相结合，以提升其在各类任务中的表现。

第 3 章　基于工具调用路径的工具图谱构建

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，通过图的格式对工具信息进行建模后，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图谱构造的策略，并基于此方法实现了一个大规模的工具图谱。

本章将会围绕着工具图谱构建中的一些挑战出发，具体来说：1. 如何筛选调用路径数据，得到高质量的工具集以及工具调用路径？2. 如何设计工具图谱的结构，以表示工具之间的依赖关系？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，我们提出了基于工具描述信息和基于工具调用路径的两种数据筛选策略，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路径数据。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，通过对工具路径数据进行计算，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高质量的工具图谱。数据筛选和清洗流程中，本文制定了根据工具来筛选和根据调用路径筛选两种策略，得到高质量的工具调用路径作为图谱构建的基础数据。其次，我们设计了知识图谱的本体模型结构，对工具之间的依赖关系进行了建模，将工具数据映射到工具图谱上。在进行图谱搭建时，我们设计了两种图上的权值：边上的工具转换权值和节点上的工具可用性权值，分别表示工具之间的依赖关系和工具的可用性。并通过图谱构建策略计算出这两个权值和制定了更新策略。最后，我们根据清洗后的数据和计算得到的工具图边权/点权得到了一个工具图谱，用于指导大语言模型的工具调用，具体使用方法在第四章有详细阐述。

3.3 数据收集和清洗

3.3.1 数据集介绍

在大语言模型工具调用研究中，存在许多公开的数据集和 Benchmark，这些数据集通常包括工具的描述文件（例如 JSON 格式或 OpenAPI 格式）、工具调用路径，以及工具调用需求的测试数据等。这些数据集为大语言模型工具调用的性能评估和模型优化提供了重要的基准资源。

目前常见的工具调用数据集包括 ToolBench[2]、API-Bank[75]、TaskBench[39]和 ToolAce[76]等。这些数据集在工具数量、描述细节、调用复杂度以及场景覆盖方面具有各自的特点。其中，ToolBench 以其覆盖范围广、工具真实、调用路径数据丰富等优势，被广泛应用于大语言模型工具调用相关的研究任务中。

在本研究中，选择 ToolBench 作为主要数据集，原因在于其具备以下特点。首先，ToolBench 的数据来自真实世界的 API 工具，这些工具不仅可以直接调用，其中部分工具甚至是免费的。这种基于真实环境的数据，使研究结果更具实践意义。其次，ToolBench 中的 API 通过统一的密钥（key）即可调用，无需额外的注册或复杂的鉴权步骤，极大地降低了数据使用的门槛和开发成本。此外，ToolBench 包含了大量 API 工具，覆盖 49 个类别的 16464 个真实的 RestAPI 工具，工具描述较为详细且覆盖范围广泛，即使部分工具描述存在不足，也可以通过平台寻找功能相似的替代工具。最后，ToolBench 提供了丰富的工具调用路径数据集，共计 126486 条工具调用路径，这些路径数据不仅体现了工具协作完成任务的逻辑关系，还为研究工具调用规律提

供了宝贵资源。

ToolBench 的整体构造聚焦于多情景的工具调用，其数据来源于开放的 API 托管平台“RapidAPI Hub”。在 RapidAPI Hub 平台上，API 工具以层级化结构进行组织。首先是大类别（如天气、金融、翻译等），每个大类别包含多个工具集（tool collections），每个工具集中进一步包含若干具体的 API 工具。用户通过订阅工具集即可访问其对应的 API 工具。

ToolBench 包含多种类型的数据，包括每个工具集的详细描述信息（例如工具集名称、描述、类别等），工具的输入和输出参数列表，工具调用的示例代码，以及工具调用路径数据等。这些数据为工具调用任务的研究提供了全面支持。在本文的研究中，为了构建工具图谱，重点使用了 ToolBench 中的工具调用路径数据。这些路径数据描述了不同工具协作完成任务的调用顺序和逻辑关系，为工具图谱的构建奠定了基础。

值得注意的是，RapidAPI Hub 平台上的 API 工具具有一定的功能相似性，多个工具可能在功能上互为替代。因此，对于同一用户任务，通常存在多条调用路径可以达到目标结果。这种特性使得任务路径并非唯一，而是存在多样化的选择。研究中，可以通过分析调用路径的最终输出结果是否符合预期来评估路径的合理性，即判断经过一组 API 顺序调用后的系统行为是否满足任务需求。这种多路径、多结果的特性为研究工具调用的灵活性和鲁棒性提供了更多的分析维度。

3.3.2　数据清洗

3.3.2.1　工具节点筛选

在筛选高质量 API 工具之前，我们首先限定了工具的应用领域，聚焦于购物、旅游、天气和餐饮四个类别。这些类别的 API 工具功能丰富，为用户提供了广泛的应用场景支持。

在工具的初步筛选后，我们对 API 的描述信息丰富度进行了严格评估和筛选。为此，我们使用了大语言模型进行批量打分，并设计了一套评分标准来量化描述信息的质量，从三个维度进行评估：描述的完整性（如是否详细说明功能）、易读性（如是否语义清晰）。

每个维度的分数范围为 1 到 3 分，其中：

- 1 分：不满足要求；
- 2 分：基本满足要求，但有一定的不清晰或者不完整；
- 3 分：描述非常详细，语言清晰流畅，信息完整且丰富。

筛选规则：只有在所有维度上均得分为 3 分的工具才会被保留下来。

为验证大语言模型的打分可靠性，我们抽样了 100 个 API 的信息提交给 3 位人工评审员进行独立打分，并与大语言模型的打分结果进行了对比分析。

一致性（Consistency）在此处定义为人工打分与大语言模型打分结果的完全一致性。如果人工评审员与大语言模型对同一 API 的打分在某一维度上相同，则认为该 API 在该维度的评分是一致的。表中的一致性百分比是根据以下公式计算的：

$$\text{一致性 (\%)} = \frac{\text{人工与大模型打分相同的 API 数量}}{\text{总 API 数量}} \times 100$$

该指标用于量化大语言模型评分与人工评审之间的匹配程度，从而验证模型评分的可靠性。

表3.1展示了两者在不同维度下的评分分布及一致性。

表 3.1 Comparison of Human and Model Scoring Distribution

Dimension	Score	Human Count	Model Count
Completeness	1	20	22
	2	40	38
	3	40	40
Completeness Consistency	84%		
Readability	1	18	20
	2	42	40
	3	40	40
Readability Consistency	79%		

由上可见，人工打分与大语言模型打分具有较高的一致性，说明大语言模型评分与人工评审具有较高的可信度。

在完成描述信息筛选后，我们对筛选出的 API 进行了连通性测试，以确保其实际可用性。测试内容包括：

- 响应状态码；
- 请求响应时间；
- 错误信息；
- 返回内容的完整性。

最终，仅保留响应状态码为正常且在规定时间内返回有效内容的 API 工具。

通过筛选，我们从初始的工具池中提取了 **150 个高质量 API 工具**，这些工具在描述信息和功能性两方面均表现优异，涵盖购物、旅游、天气和餐饮四个领域。筛选

后的 API 工具分布情况如下：

表 3.2　Distribution of API tools by category after filtering

Category	Number of Tools	Number of APIs
Shopping	18	40
Travel	22	50
Weather	15	45
Dining	10	25

3.3.2.2　工具调用路径筛选

工具调用路径数据可能会有未完成任务导致失败/工具调用出错等问题，并且数据集中的工具调用是以决策树的方式组织的，需要从中进行剪枝获得正确的线性的工具路径。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：

删除失败任务：为确保工具调用路径能准确解答用户需求，我们仅保留成功调用并产生结果的工具调用路径，删除那些无法得出有效结果的工具调用决策树。具体依据调用路径中的 `"finish_type"` 字段进行筛选，其中 `"give_answer"` 的路径保留，`"give_up"` 的路径丢弃。

对工具调用树进行剪枝: 在 ToolBench 中的工具调用树中，工具的探索和调用路径是以决策树的形式组织的，因此我们需要对工具调用树进行剪枝，以获得一条线性的工具调用路径。因此，我们通过数据中的 `"pruned"` 字段进行剪枝，仅保留 `"pruned"` 为 `"false"` 的字段。

删除调用错误节点：在筛选优质工具路径后，我们需要进一步清洗工具路径数据，以确保调用路径上的工具都可执行。我们在调用路径中根据`"error"` 字段的内容来决定是否删除节点。对于调用过程中发生错误的工具节点（如未授权错误、API 不可用等错误），将其从路径中剔除，以确保路径的完整性和可执行性。最终，保留下的应该是一条线性且无错误的调用路径。

3.4　图谱构建

3.4.1　工具图谱概念模型

工具图谱中的实体类别主要包括三种：工具节点、工具组节点和工具类别节点。每种实体类别分别表示不同的抽象层次和功能，具体如下：

3.4.1.1 工具（Tool）

工具节点是图谱的基础实体，表示具体的 API 接口工具或单个功能模块。其结构如图3.1所示：

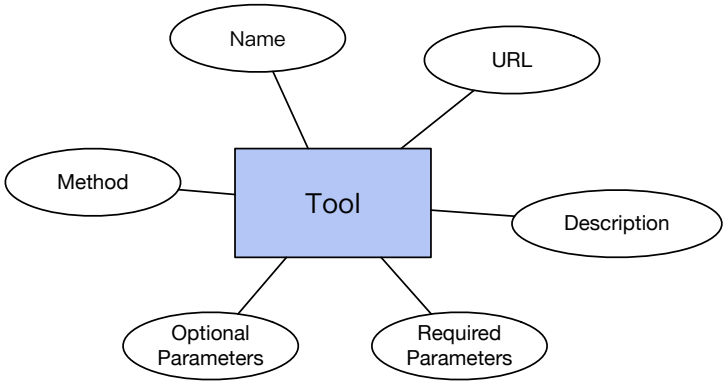

图 3.1 知识图谱工具实体
Figure 3.1 Tool Entity Structure

- **Name（名称）**：工具的唯一标识名称。
- **URL（接口地址）**：API 工具的访问地址。
- **Description（描述）**：工具的简要说明，通常用于描述其用途或功能。
- **Method（请求方法）**：支持的 HTTP 方法，如 GET、POST 等。
- **Required Parameters（必需参数）**：调用 API 所需的必填参数列表。
- **Optional Parameters（可选参数）**：调用 API 的可选参数列表。

在工具图谱的组织中，每个工具节点只属于一个工具组。多个工具可以归属于同一个工具组，从而形成对工具功能的逻辑分类。例如，“酒店搜索”工具和“酒店评论查询”工具都可能属于“旅游服务工具组”。这种一对多的归属关系确保工具图谱结构的清晰性，同时避免工具节点的多重归属问题。

3.4.1.2 工具组（Tool Group）

工具组节点表示一组功能相关的工具的集合，用于组织和分类多个工具节点。其结构如图3.2所示：

- **Group ID（工具组 ID）**：工具组的唯一标识符。
- **Name（名称）**：工具组的名称。
- **TDescription（工具组描述）**：对该工具组的整体描述。
- **Home URL（链接）**：工具组的官网链接或信息页面。

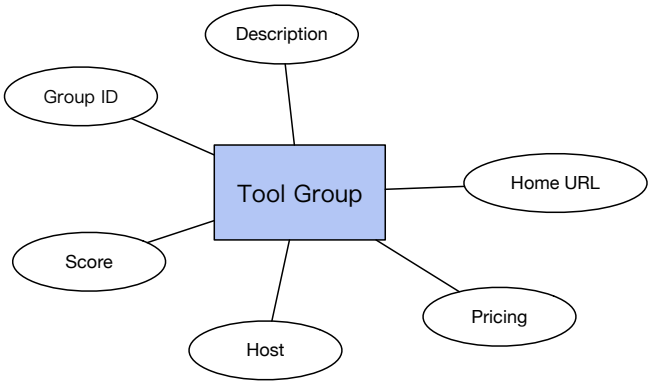

图 3.2　知识图谱工具组实体
Figure 3.2　Tool Group Entity Structure

- **Pricing（定价）**：工具组的定价信息，例如“免费（FREE）”。
- **Score（评分）**：工具组的相关指标，包括：
  - **AvgServiceLevel（服务水平平均值）**：表示服务稳定性的指标，百分比表示。
  - **AvgLatency（平均延迟）**：工具组调用的平均延迟时间，单位为毫秒。
  - **AvgSuccessRate（成功率平均值）**：表示 API 调用成功率的平均值，百分比表示。
  - **PopularityScore（流行度评分）**：工具组的受欢迎程度评分。
- **Host（请求地址）**：工具组所关联的主要域名地址。

工具组进一步与工具类别关联，每个工具组只属于一个工具类别，从而实现高层次的分类。例如，“旅游服务工具组”可以归属于“旅游与出行”类别。工具组与工具类别之间的一对多关系，能够有效区分不同工具组的功能领域。

3.4.1.3　工具类别（Tool Category）

工具类别节点表示工具所属的领域、用途或分类。例如，工具类别可以表示工具的主要负责内容为“旅游出行”、“美食”或“电影媒体”。其结构如图3.3所示：

- **Name（名称）**：类别的名称，用于标识该类别。
- **Description（描述）**：对工具类别的简要说明，描述其应用场景或技术范围。

工具类别节点与工具组节点的关系是唯一的映射关系，每个工具组只能归属于一个工具类别。此外，工具节点的类别由其所属工具组的类别决定，工具节点和工具组的类别在逻辑上保持一致。这种设计保证了工具图谱的分类层次统一性。例如，“酒

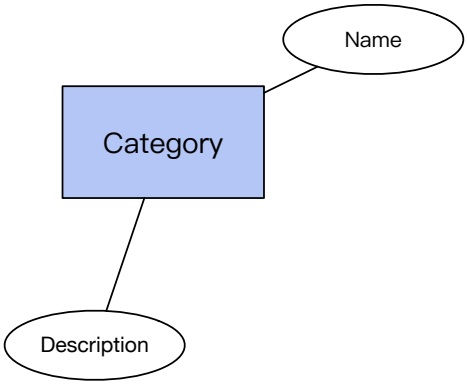

图 3.3 知识图谱工具类别实体
Figure 3.3 Category Entity Structure

店搜索”工具属于“旅游服务工具组”，“旅游服务工具组”属于“旅游与出行”类别，因此“酒店搜索”工具的类别也是“旅游与出行”。

工具之间的依赖关系通常分为三种[39]: 时序依赖、资源依赖、环境依赖。环境依赖指工具运行所需的外部环境条件（如特定硬件、操作系统或软件版本等）。在本研究中，由于工具多为基于 REST API 的服务接口，不涉及环境依赖。因此，本文在工具图谱构建中仅考虑时序依赖和资源依赖两种依赖关系。

在工具图谱的概念模型中，依赖关系主要包括以下三种类型:

3.4.1.4 时序依赖（Sequential Dependency）

时序依赖指的是在历史工具调用路径中被连续调用的两个工具之间的关系。描述如下:

- **起点（Source）**: 工具 A。
- **终点（Target）**: 工具 B。
- **关系描述（Description）**: 工具 B 在调用时需要工具 A 的先行调用。

3.4.1.5 强资源依赖（Strong Resource Dependency）

强资源依赖指的是某种资源或信息仅能通过前一个工具获取，然后提供给后一个工具。例如，一个旅行商的“酒店 ID 搜索”工具获取特定的酒店 ID，而“酒店评论搜索”工具必须依赖该 ID。描述如下:

- **起点（Source）**: 工具 A。
- **终点（Target）**: 工具 B。
- **资源类型（Resource Type）**: 强依赖的资源类型，例如 “酒店 ID”。

- **关系描述（Description）**：工具 A 输出的资源是工具 B 必须依赖的输入。

3.4.1.6　弱资源依赖（Weak Resource Dependency）

弱资源依赖指的是某种资源或信息可以通过另一个工具获取，但不是必须的。例如，“根据经纬度搜索降雨情况”工具可以直接调用，也可以通过“城市经纬度搜索”工具提供经纬度作为输入。描述如下：

- **起点（Source）**：工具 A。
- **终点（Target）**：工具 B。
- **资源类型（Resource Type）**：弱依赖的资源类型，例如 “经纬度”。
- **关系描述（Description）**：工具 B 的输入可以由工具 A 提供，但也可以通过其他方式获取。

3.4.2　实例模型构建

本节详细阐述工具知识图谱的构建过程，包括节点和依赖关系的抽取、存储，以及如何将数据导入 Neo4j 进行存储和查询。同时探讨如何扩展图谱构建的内容和方法，为更丰富的应用场景提供支持。

3.4.2.1　基于规则的工具有关实体抽取

工具组、工具和工具类别节点的抽取主要依赖规则匹配。从现有的 JSON 格式工具描述文档中，按照预定义规则通过代码匹配关键信息。工具组的抽取字段包括产品 ID、工具组名称、工具组描述和主页链接等信息，这些字段为工具组提供全局标识及其关联性。工具节点的提取则更加具体，记录每个工具的名称、API 地址、功能描述、HTTP 方法（如 GET 或 POST）、必需参数和可选参数等关键信息。工具类别节点用于标识技术领域或功能分类，从技术标签中提取类别名称、描述及相关技术栈等字段。

通过规则提取的节点数据存储为结构化格式。这些文件为图谱数据导入提供了清晰的数据基础，确保后续步骤的顺利进行。

表 3.3　工具实体数据格式

Category	Product ID	Tool Name	API Description	Hash ID
Food	`api_dcbd91ed...`	Worldwide Recipes	Get detail of recipe	`167fc042...`
Food	`api_dcbd91ed...`	Worldwide Recipes	Get recipes by author	`d8c4f1e7...`
Food	`api_dcbd91ed...`	Worldwide Recipes	Get reviews	`47f3a387...`

3.4.2.2 基于规则的时序依赖关系抽取

时序依赖是通过分析工具调用日志数据来构建的。首先，对调用日志按照时间戳进行排序，从而生成工具调用的顺序链条。接着，识别这些调用链中的连续工具对，将其标记为时序依赖。这种依赖通常用于捕获工具在真实调用路径中的关系，反映调用流程的顺序逻辑。例如，旅行工具链中“酒店搜索”紧接着“酒店评论查询”，这反映了用户实际使用时的时序逻辑。

抽取后的时序依赖关系记录起点工具、终点工具、依赖类型（标记为“时序依赖”）以及描述字段。时序依赖的记录不仅有助于绘制工具调用路径，还为分析流程优化提供依据。

3.4.2.3 基于大语言模型的资源依赖抽取

我们首先尝试了一种基于规则匹配的方案，通过对工具 API 的输入参数和输出字段进行名称和类别的比对来识别潜在的依赖关系。这种方法的实现逻辑较为简单，对结构化和模式化的依赖关系具有较好的识别能力。然而，在实际应用中，该方案暴露了两个主要问题：

- 参数名称不一致导致的遗漏。同一参数可能在不同工具中使用不同的名称，例如“酒店 ID”在某些工具中可能表示为不同的字段。这种不一致容易导致依赖关系的遗漏。
- 参数名称相同但语义不同引入的噪声。同名参数在不同工具中可能具有不同的语义。例如，`ID` 在“酒店搜索”工具中表示“酒店的唯一标识符”，而在“航班搜索”工具中可能指“航班班次编号”。这种情况下，会误判依赖关系，从而引入噪声。

为了解决上述问题，我们引入了大语言模型对 API 之间的依赖关系进行分析和生成。利用 LLM 强大的语义解析能力，我们能够直接分析工具的 API 文档（包括功能描述、输入和输出字段定义等），并从语义层面识别字段间的关系。引入 LLM 后的方案具有以下显著优势：

- 高效处理参数名称不一致的问题。LLM 能够基于上下文语义理解字段的实际含义，而不是简单地依赖名称匹配，从而避免依赖关系的遗漏。
- 准确区分参数语义差异。对于同名字段，LLM 能够结合工具功能和字段定义的语义信息，区分不同工具中参数的具体含义，从而减少噪声的引入。
- 简化流程并提高准确性。通过让 LLM 直接生成依赖关系，可以跳过基于规则匹配的复杂流程，在大规模工具集成中展现出更高的适用性。

总体而言，引入 LLM 的方案，不仅弥补了基于规则匹配的局限性，还通过语义

理解能力显著提升了依赖关系识别的全面性和准确性，为资源图谱的构建提供了更加可靠的基础。

3.4.2.4　图谱数据导入

构建完成的工具知识图谱数据需要导入 Neo4j 图数据库，以支持高效存储和复杂查询。数据导入分为节点和关系两部分。首先，工具、工具组和工具类别被转化为图中的节点。它们分别通过名称、ID 等标识字段建立类型，并与属性字段一一关联。

关系数据的导入使用工具节点的标识字段建立工具之间的依赖关系，包括时序依赖、强资源依赖和弱资源依赖。依赖关系被标注有明确的类型和描述字段，便于在查询中准确反映其性质。导入完成后，可以通过可视化工具验证图谱中的节点和关系，确保结构的准确性和逻辑性。

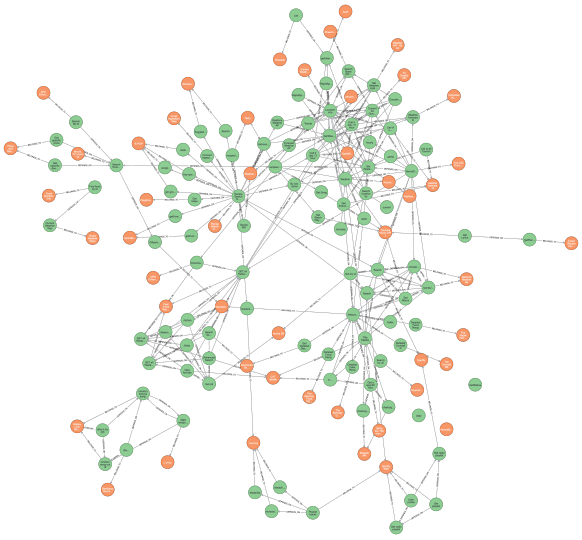

图 3.4　工具图谱
Figure 3.4　Tool Knowledge Graph

3.5　本章小结

本章提出了一种基于工具调用路径的工具图谱构建方法，系统化地完成了从数据筛选到图谱验证的核心流程。首先，针对工具描述信息和调用路径数据，设计了两种数据筛选策略，提取出高质量的工具组和有效的工具调用路径，为图谱构建奠定了坚实的数据基础。在图谱结构设计方面，定义了工具图谱的核心概念模型，包括工具节点、工具组节点和工具类别节点，明确了它们之间的层次关系与依赖结构。通过构

建时序依赖和资源依赖两种关系，建模了工具之间的调用逻辑，保障了图谱的结构化和实用性。最终，结合数据筛选、清洗与数据格式转换等操作，成功构建了一个高质量的工具图谱，并将其导入 Neo4j 数据库以实现可视化和高效查询，为复杂任务场景下的工具调用与规划提供了有力支持。

第 4 章　基于工具图谱与深度优先遍历的工具编排与调用方法

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工具知识图谱，包括有对调用路径数据的清洗、工具图谱的构建方法、工具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，图上固有的路径不能满足变化的用户需求。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工具选择方法。使用大语言模型进行选择和编排能够提升系统的动态性和灵活性，但仅依赖大语言模型的语义分析能力难以处理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工具路径选择的部分，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工具调用路径。最后，为了进一步精简返回结果内容、减少推理延迟和提升系统效率，我们提出了一种响应压缩方法，通过让模型选择与用户需求有关的字段来生成压缩后的响应结果，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. **任务分解模块**：任务分解模块通过对用户模糊或复杂需求进行解析，将其拆解为明确且独立的子任务，使复杂问题变得更易执行和解决。模块利用语义分析，

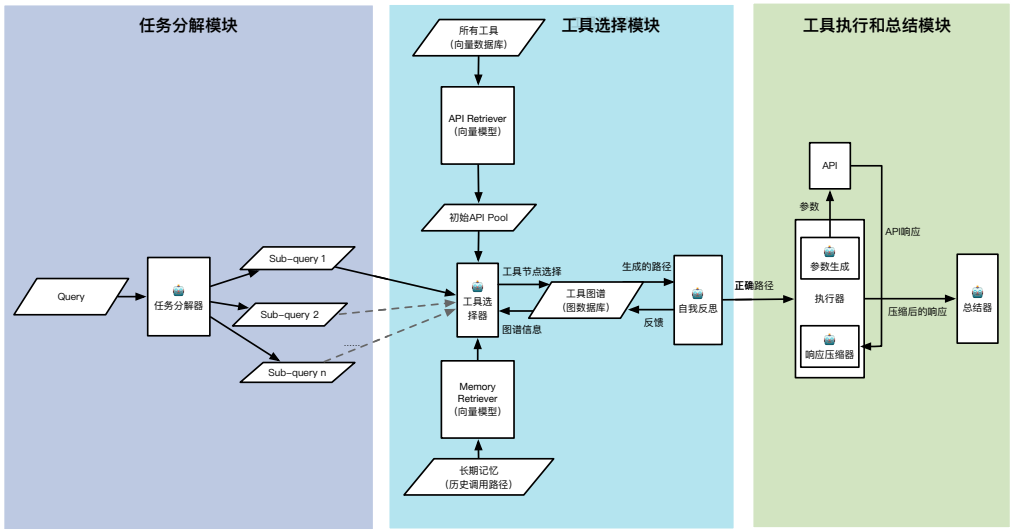

图 4.1　整体框架

Figure 4.1　Overview of the Dynamic Tool Selection Framework

根据任务的逻辑结构和类别对需求进行分解，既提升了任务处理的效率，也增强了系统的响应准确性。同时，通过识别任务间的依赖关系，确保了整体任务流程的有序性和可执行性。

2. **工具选择与编排模块**：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. **工具调用与总结模块**：在得到了整体的路径后，任务执行模块负责调用工具，并将结果进行汇总，最终输出结果。其中涉及到了工具参数配置、工具响应压缩模块。工具参数配置模块直接将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。工具响应压缩模块则负责对工具的响应结果

进行压缩，以减少响应时间，提升交互效率。我们将任务总结智能体作为一种特殊形式的工具，在所有前序工具执行结束后调用，针对用户需求输出最终结果。

在流程上，对于每个用户需求，只会执行一次子任务分解。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工具，因此会多次调用工具执行模块。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　任务分解模块

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执行单元，包括“子任务编号“、“子任务描述”、“子任务类别”等重要信息。子任务之间可能会存在时间或者参数上的依赖，该依赖会在任务分解的时候通过特殊的占位符来表示，用于串联整体的任务编排流程。

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。逻辑上分解有利于明确每个任务的目标，没有依赖关系的任务可以并行执行，能够有效提高任务的效率和正确率。按照类别来分解能够缩小在工具池中的搜索范围，提高搜索效率和准确性。其提示词如下所示：

Task Decomposition and Response Planning with GPT-4

Instruction Prompt: Decompose the task and plan its steps in a structured way. Identify tool dependencies and output the result as a JSON list.

Task Description: 1. Break down the task into multiple subtasks in a step-by-step manner. Each subtask should be labeled as **StepX** (e.g., Step1, Step2). 2. Specify tool usage and dependencies: - If a step depends on another step's result, represent the input using its output reference, e.g., `Input: [A1]` where `A1` refers to the result from Step1. - Use multiple inputs if needed, e.g., `Input: [A1, A3]`. 3. Output the plan in a `JSON` format list. 4. Ensure logical dependencies and a clear structure for each subtask.

Example Task: What is the most popular travel destination in Europe? And what's the weather like there?

Expected Output: {

```
"Steps": [
{ "Step1": "Identify the most popular travel
destination in Europe.", "context": [],
"category": "travel" },
{ "Step2": "Fetch the current weather for the
identified destination.", "context": ["A1"],
"category": "weather" },
{ "Step3": "Generate a summarized response
combining the destination and weather
information.", "context": ["A1", "A2"],
"category": "llm" }
] }
```

通过以上的大语言模型提示词工程，我们可以让大语言模型输出 JSON 格式的任务执行流程，并且根据任务之间的依赖关系对任务进行排序。

如图4.2最终形成的子任务图是一个有向无环图，其中每个节点代表一个子任务，每条边代表任务之间的依赖关系。

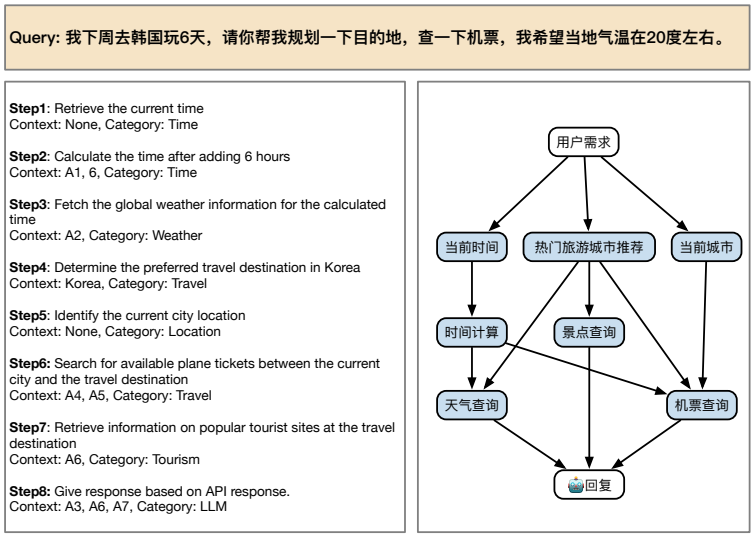

图 4.2　任务分解模块
Figure 4.2　Task Decomposition Module

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，并显式表示子任务之间的依赖，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　工具选择与编排模块

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工具选择算法

图 4.3为本文提出的工具选择算法的效果展示。

首先，在上一步任务分解模块中，我们将整个任务构建为一个有向无环图，并且用边来表示任务之间的依赖关系。那么我们在选择每个子任务的工具时，需要考虑到前序任务的所调用的工具。

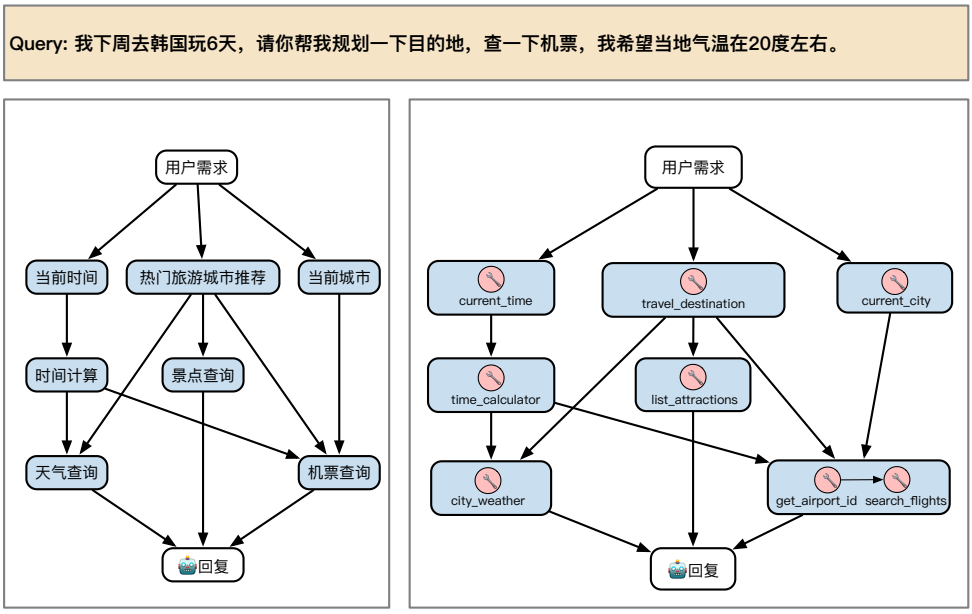

图 4.3　基于深度优先遍历的工具选择算法
Figure 4.3　Dynamic Tool Selection based on DFS

因此算法流程如下所示，对没有前序任务的节点，我们通过语义相似度算法得到一组候选的工具集合，然后通过提供 LLM 这些工具的名称、描述和参数等信息，直接让 LLM 来选择对应的工具。

对于有前序任务的节点，我们通过寻找前序任务的工具节点的邻居节点来得到一组候选工具。不同前序任务之间的邻居节点取并集提供给 LLM。在筛选邻居时，我们会根据任务分解器的“类别”字段筛选特定工具类别的邻居，以过滤掉噪声工具节点，提高工具选择的有效性。若取并集之后的工具节点仍过多，我们将会计算每个工具与子任务描述的相似度，并进行排序，选择前 K 个工具节点提供给大语言模型。我们设计了简单实验来选择这个 K 值以及选取前 K 个的有效性，具体见第五章的实验部分。

对于搜索得到的邻居节点，我们在提示词中对不同类别的依赖关系设置了不同程度的重要性，引导大语言模型选择合适的工具节点来完成子任务。

在处理具有前序任务的工具节点时，我们通过寻找这些工具节点的邻居节点生成候选工具集合。对于多个前序任务节点，其邻居节点取并集，形成综合依赖关系的候选工具集供大语言模型（LLM）使用。

为了提高工具选择的有效性，我们依据任务分解器的“类别”字段筛选相关工具节点，过滤掉与任务无关的噪声节点。如果候选工具仍然过多，则计算工具与子任务描

述的语义相似度，并选择前 K 个最相关的工具作为候选。

提示词中引入了依赖关系优先级规则，以引导大语言模型选择合适工具完成子任务。依赖关系的优先级为：强依赖 > 弱依赖 > 时序依赖。

以下为提示词示例：

Example Prompt for Tool Selection

Instructions: You are tasked with selecting the most appropriate tools to solve the given sub-task. Based on the provided tool descriptions and their dependencies, choose tools in the following order of priority: **Strong dependency > Weak dependency > Sequential dependency**. Use the chosen tools to complete the sub-task.

Sub-task: Calculate the time difference between the current time and a target time.

Available Tools:

1. *Current Time Retrieval Tool:* No dependency, Time Information.
2. *Time Difference Calculator Tool:* Weak dependency (requires `current_time` as input).
3. *Time Zone Converter Tool:* Sequential dependency (depends on output from Time Difference Calculator, Time Adjustment).

通过该算法可以迭代得到子任务图上每个子任务所需要调用的工具节点，并且形成一个工具之间的有向无环图，为后续调用提供调用顺序和依赖关系。

4.3.2.2　工具召回器

在上述算法中，初始工具集以及后续候选前 K 个工具集都是根据工具和子任务描述的语义相似度来选择的，将子任务描述和工具描述转为向量的嵌入模型对选择正确工具有很大的影响。

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工具召回器，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。

通常工具召回是通过向量模型和相似度算法来进行的，具体而言，我们将子任务描述和每个工具的信息都转化为向量表示，并通过计算查询向量与工具向量之间的相似度来检索相关工具。这个过程首先使用预训练的语言模型将工具的文本描述转

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工具向量之间的相似度，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工具，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，对召回的工具进行排序，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工具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 4.4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。

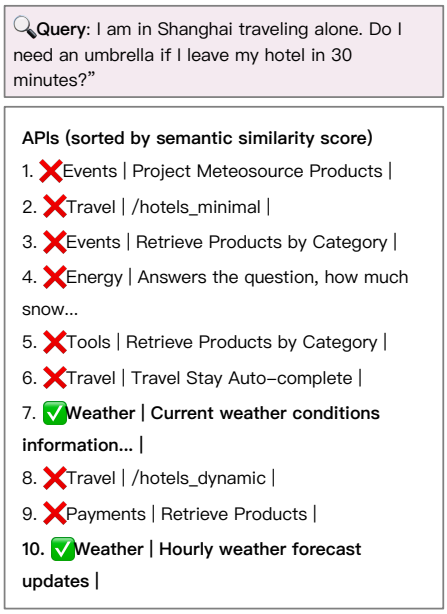

图 4.4　使用开源向量模型得到的工具排序结果
Figure 4.4　工具 Ranking Using Open-source Embedding Models

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工具列表。

对于负样本的部分，如图 4.5，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）。

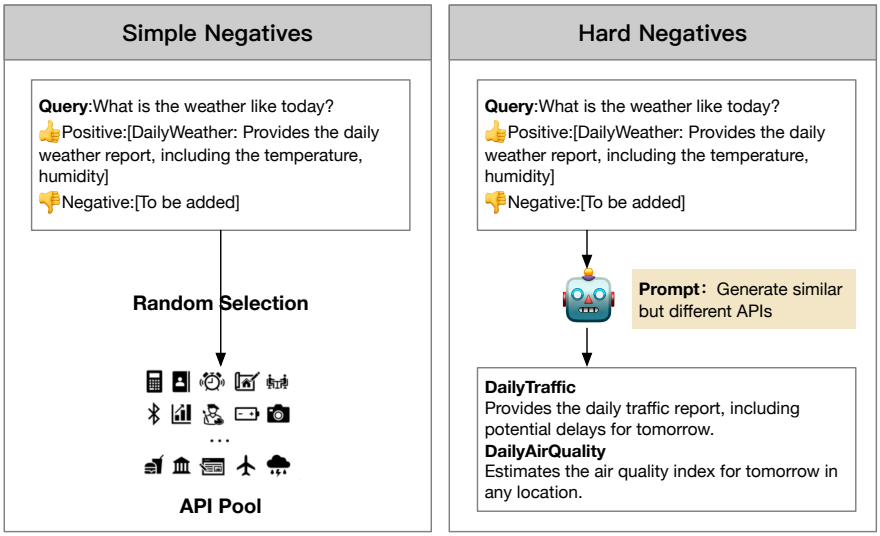

图 4.5　负样本构造的两种方式
Figure 4.5　Negative Sample Generation

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，不同类别的工具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 > 的数据，然后我们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，我们直接进入工具调用模块进行调用。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　工具调用路径记忆框架

本节提出了一种增强模型规划和推理能力的记忆框架，该框架分为短期记忆和长期记忆两个部分。短期记忆聚焦于动态推理过程中的实时信息记录，长期记忆则存

储历史调用路径以支持未来推理任务的优化。

短期记忆　短期记忆是模型在图上动态推理时保存的状态信息，旨在实时支持当前推理过程。其内容包括用户任务、当前遍历节点、历史遍历节点以及调用路径等关键数据。在推理过程中，短期记忆会被动态更新，以反映当前状态和推理环境。为确保模型能够感知这些状态信息，我们将短期记忆存储于内存中，并在每次推理时通过构建提示词直接将其注入大语言模型的上下文。这种机制不仅能帮助模型理解当前推理场景，还能提高其规划和决策的准确性。

长期记忆　长期记忆用于存储历史工具调用路径及其对应的推理结果，并随着调用次数的增加不断扩展。这些信息存储在数据库中，并在后续推理任务中通过相似度检索为模型提供支持。长期记忆的核心目标是利用历史经验优化模型的规划与工具选择能力。

长期记忆模块基于检索增强生成（Retrieval-Augmented Generation, RAG）框架，将历史上成功的 < 子任务描述, 工具调用路径 > 转化为向量存储。对于新的任务需求，系统会将其嵌入为向量，并与长期记忆库中的向量进行相似度计算。通过排序，系统检索出与当前需求最相似的 K 个历史任务及其工具调用路径。最终，这些历史任务的调用路径和描述以自然语言的形式被注入大语言模型的上下文，作为提示词辅助工具选择模块的推理与执行。

这种结合短期与长期记忆的框架，既能动态适应实时推理的需求，又能通过历史知识积累，增强模型对复杂任务的规划能力，从而显著提高系统的效率与准确性。

4.3.3　工具调用与总结模块

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工具调用 DAG，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，最终调用 LLM 来进行总结，生成符合用户需求的答案。

工具调用部分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工具调用的参数，然后通过代码生成请求体并通过工具调用接口将请求体发送给目标工具，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，要求模型根据工具响应输出最终结果。在系统设计中，为了方便流程的管理和设计，我们将工具总

结智能体作为一种特殊的工具，在所有工具都执行完毕之后，调用该工具对其他工具响应结果进行总结和输出，作为系统的最终输出。

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工具，可能单个工具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，通过对响应的压缩来缓解大语言模型的上下文限制，以便于工具参数生成和工具总结器的正确执行。

4.3.3.2　工具并行调用模块

工具并行调用模块的目标是优化工具调用 DAG 的执行效率，特别是在多个工具之间没有依赖关系的情况下，通过并行处理显著缩短整体运行时间。

并行调用逻辑　在工具调用 DAG 中，每个工具节点可能存在两种关系：

1. 独立节点：工具之间无参数或时序依赖，可同时执行。
2. 依赖节点：工具的输入参数依赖于其他工具的输出，必须按照依赖关系依次执行。

为了实现并行调用，我们首先对工具调用 DAG 进行拓扑排序，以明确工具的依赖关系。然后根据以下步骤划分工具调用组：

1. 将无前序依赖的工具分组为一个批次，并行启动调用。
2. 等待当前批次所有工具完成后，获取其输出结果并解析。
3. 将当前批次的输出作为后续工具的入参，继续处理下一批次工具。

这种逻辑确保了工具的调用顺序既满足依赖关系，又尽可能实现并行化处理，从而提高整体执行效率。如图4.6所示，

工具并行调用的主要优势包括以下三个方面：首先，通过将无依赖关系的工具分阶段并行启动，避免了工具调用之间的相互阻塞，显著提高了整体处理效率；其次，并行逻辑优先优化关键路径的工具调用时间，同时使独立路径上的工具可以并行完成，从而有效缓解关键路径的瓶颈问题，减少关键路径上的总耗时；最后，并行调用支持多个工具同时调用大语言模型（LLM）进行处理，降低因串行调用而累计的 LLM 响应延迟，大幅提升系统的整体运行效率。

时间复杂度与性能分析　假设工具调用 DAG 包含 x 个工具，其中每个工具的平均调用耗时为 t，最长依赖路径包含 L 个工具节点。则顺序调用的总耗时为：

$$T_{\text{顺序}} = \sum_{i=1}^{x} t_i \approx x \cdot t$$

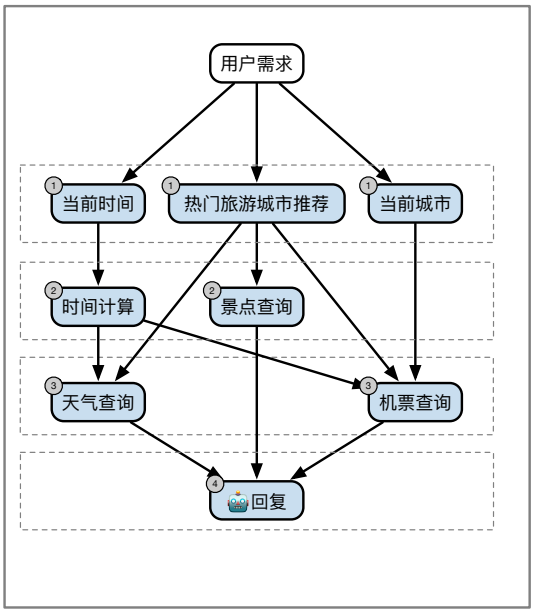

图 4.6　工具并行调用逻辑

而并行调用的总耗时由最长依赖路径 L 决定：

$$T_{\text{并行}} = \sum_{i=1}^{L} t_i \approx L \cdot t$$

在无依赖情况下，所有工具均可并行调用，此时整体耗时仅为单个工具中耗时最长的调用时间：

$$T_{\text{无依赖并行}} = \max_{i=1}^{x} t_i$$

然而，在实际应用中，工具调用的总时间还需考虑 LLM 的延迟因素：

$$T_{\text{实际}} = T_{\text{工具调用}} + T_{\text{LLM 延迟}}$$

其中：

- $T_{\text{工具调用}}$ 是工具执行的时间，由上述拓扑排序及并行调用决定。
- $T_{\text{LLM 延迟}}$ 包括请求体生成时间与响应压缩时间，具体取决于工具数量和响应数据的规模。

以一个具体的示例说明：假设工具调用 DAG 有 10 个工具，每个工具的平均调用耗时为 $t = 1$ 秒，最长依赖路径包含 4 个节点（$L = 4$）。如果顺序调用，总耗时为：

$$T_{\text{顺序}} = 10 \cdot 1 = 10\,\text{秒}$$

而通过并行调用，总耗时降至：

$$T_{\text{并行}} = 4 \cdot 1 = 4 \text{秒}$$

实际系统中需加上 LLM 的延迟时间，但并行调用的节省逻辑依然成立，尤其在依赖关系较少或任务分布均匀的情况下。

并行调用通过优化工具的执行顺序，有效缓解了因依赖关系导致的等待时间长的问题。尽管 LLM 的生成延迟和响应压缩带来额外时间开销，但并行逻辑依然能够充分利用资源，显著提升系统的整体效率。

4.3.3.3　工具入参生成模块

工具入参生成模块的主要任务是为工具调用动态生成请求体。请求体中固定的信息（如请求 URL、请求方式、Key 等）可以从工具图谱和配置文件中直接获取，而动态生成部分主要包括工具的请求参数列表。

根据工具调用的流程，每个工具可能存在零到多个前序工具，这些前序工具的输出可能会直接影响当前工具的参数生成。

为了正确生成当前工具的入参，系统在所有前序工具调用完成后，会将以下信息提供给参数生成智能体：当前工具的名称和描述、参数列表及其描述、前序工具的调用响应以及参考格式。

随后，系统会对生成的 JSON 字符串进行解析，得到实际的 JSON 对象作为参数。解析后的参数需要经过严格校验，以确保其符合预期要求。校验内容包括三个方面：一是**参数完整性**，验证是否缺少必选参数；二是**参数类型**，确保每个参数的类型（如整数、字符串、数组）与定义一致；三是**参数名称**，检查生成的参数名称是否与工具的参数定义匹配。校验过程可以形式化表示为：

$$\text{validate}(P_{\text{parsed}}) = \begin{cases} \text{True}, & \text{若参数满足所有校验规则；} \\ \text{False}, & \text{若参数校验失败。} \end{cases}$$

若校验未通过，系统将返回固定的错误信息，并要求参数解析器重新生成参数。这一过程会循环执行，直到满足以下条件之一：参数生成成功且通过校验，或者达到最大尝试次数，系统放弃当前工具的调用。

当参数生成成功并通过校验后，系统会调用固定的函数来执行工具请求。工具调用的响应通常以 JSON 格式返回，随后会被提供给响应解析模块，进行进一步的处理和利用。

通过结合工具依赖关系、大语言模型生成能力和严格的参数校验机制，工具入参生成模块能够确保工具调用请求的准确性和鲁棒性，同时提高整体系统的可靠性。

4.3.3.4　工具响应解析模块

生成请求体后，我们通过工具调用接口将其发送给目标工具，并获取对应的响应数据。工具的响应数据通常以 JSON 格式返回，其中可能包含大量信息。然而，这些信息并不总是直接相关。通过实验分析发现，许多工具的返回内容中包含冗余信息，这使得响应数据长度过长，难以直接输入到大语言模型中进行处理。直接依赖大语言模型从中提取重要信息在某些情况下会导致效率下降。因此，我们引入了响应压缩模块，旨在尽可能保留关键信息的同时减少响应数据的长度，使其能够更高效地适配大语言模型的上下文限制。

由于工具响应的格式通常不固定，难以预先确定哪些字段应保留或舍弃，因此我们采用大语言模型分析工具响应的示例，仅保留与用户需求相关的字段，从而减小响应长度。

如图 4.7 所示，响应压缩模块的逻辑流程如下：

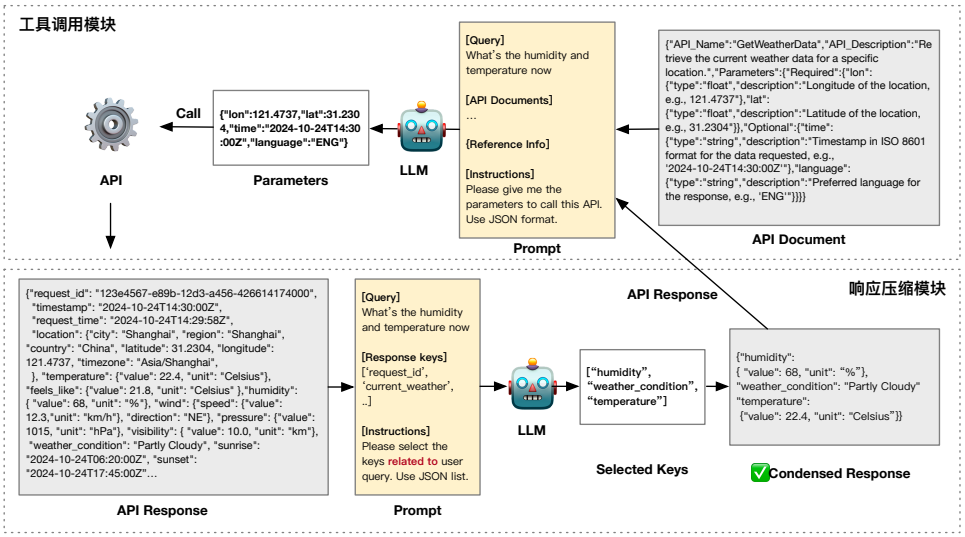

图 4.7　响应压缩模块

Figure 4.7　Compression of API responses

响应压缩的流程可以分为以下三个阶段：

首先，**提取工具文档信息**。所有工具均来自 ToolBench 等开源数据集，这些数据集包含每个工具的详细描述和示例响应信息。我们将工具名称、功能描述、参数信息

以及工具响应示例作为文本输入大语言模型。这些信息构成了压缩模块的基本提示词。

其次，**生成压缩提示词**。我们为大语言模型提供详细的规则，要求模型根据工具的功能描述提取最相关的信息。例如，工具版本、调用时间以及无效信息等与实际功能无关的内容会被舍弃，而关键字段将被保留。

最后，**添加上下文学习示例**进行强化提示。我们选取三个上下文学习示例，每个示例包含一个原始响应和对应的专家撰写的压缩响应。通过这些示例，我们要求压缩器以自然语言输出保留的字段，并使用正则表达式匹配生成字段列表，最终用代码逻辑筛选出需要的字段及其内容。

在实际应用中，当工具响应长度超过 1024 个字符时，系统会优先移除不重要的信息进行压缩。如果压缩后仍然超过 1024 个字符，则进一步截取压缩结果的前 1024 个字符。这种方法有效减少了工具响应的长度，避免冗余信息的干扰，并缓解了大语言模型上下文窗口的限制，从而确保系统调用的正常运行。

4.3.3.5　工具总结器

在所有的工具都调用完毕之后，我们通过调用工具总结器对工具的调用结果进行总结，以生成符合用户需求的答案。工具总结器并不会获取所有工具的响应，而是会根据在子任务分解时的依赖关系选择部分工具的响应结果来进行总结，即仅选择与回答有直接关联的工具响应。

这部分方法的设计类似于传统的检索增强生成（Retrieval-Augmented Generation, RAG）。在这里，通过工具调用得到的结果可以看作是类似于检索阶段的信息，而基于这些结果生成对用户需求的回答则对应于生成阶段。在这一框架中，核心目标包括：1）减少幻觉现象，确保生成的内容基于真实信息；2）确保回答的稳定性，即相同的输入能够生成一致的答案；3）提升回答的相关性，降低无关信息（噪声）的干扰。

为实现上述目标，我们设计了一个用于工具总结模块的提示词模板，通过提示词工程引导模型生成回答。如下所示，该模板为模型提供了一种结构化的思考和表达方式，使得生成的内容更加稳定、有逻辑，并能够准确满足用户的需求，从而有效提升生成质量。

An Example for Response Generation with GPT-4

Instruction Prompt: Generate a response to the user's question based on the results from the tools and your internal knowledge.

Task: What's the most popular travel destination in Europe? And what's the weather like there?

Return from Tool Calling: {"popular_travel_destination": {"destination": "Paris, France","popularity_score": 9.8},"global_weather_search": {"location": "Paris, France","weather": {"temperature": "15°C","condition": "Partly Cloudy","humidity": "60%",},"timestamp": "2024-11-17T14:00:00Z"}}

Instructions: 1. If the tool's output is incorrect, or if the information from the tools and your internal knowledge is insufficient to answer, do not respond. Avoid fabricating any untrue information.

2. Think step by step before answering. Plan the key points to address before formulating your response.

3. Only answer the user's request. Do not include any additional or irrelevant information.

4. Use the same language as the Task (in this case, English).

5. Provide the output in JSON format.

Output Format: {"Steps": ["Find the most popular travel destination in Europe.","Describe the current weather at that destination."],"Result": "The most popular travel destination in Europe is Paris, France. The weather in Paris is partly cloudy with a temperature of 15°C, a humidity level of 60%."}

4.4　本章小结

本章提出了一种基于工具图谱和深度优先遍历的动态工具编排与调用方法，通过任务分解模块将复杂需求拆解为子任务，并利用深度优先遍历算法动态规划工具调用路径，以应对工具动态性和复杂依赖关系。结合记忆框架，有效利用历史经验优化路径选择效率，同时通过语义相似度优化的工具召回器提升工具选择的准确性。

工具调用模块支持并行处理和响应压缩，减少大语言模型的上下文压力，提高执行效率。最终，通过工具总结模块对调用结果进行整合与回答生成，确保输出的准确性和相关性。本方法有效结合了工具图谱的结构化信息与大语言模型的动态决策能力，为复杂任务的灵活、高效、精准解决提供了系统化的支持。

第 5 章　实验分析

5.1　实验环境

本实验系统的配置由环境配置和系统库配置两部分组成。环境配置方面，实验使用的中央处理器为 Intel Xeon Gold 5318Y，具备 48 核心、96 线程，主频为 2.10 GHz（最高 3.40 GHz），内存为 251 GB；图形处理器型号为 NVIDIA GeForce RTX 3090，共计 8 张显卡，每张显存为 24 GB。系统库配置方面，向量数据存储使用 qdrant-client 1.11.3，图数据库采用 5.26.0 版本的 Neo4j，深度学习框架 torch 为 2.4.1 版本，并结合 FastAPI 0.111.0 和 Streamlit 1.34.0 进行前端交互，整体运行环境基于 Python 3.10。

表 5.1　Configuration of Environment

Attribute Name	Attribute Value
CPU 型号	Intel Xeon Gold 5318Y @ 2.10GHz
核心数	48 核
线程数	96 线程
主频	3.40 GHz
内存	251 GB
GPU 数量	8
GPU 类型	NVIDIA GeForce RTX 3090
GPU 显存	24 GB / 每张卡

表 5.2　Configuration of Library Environment

Library Name	Version
Python	3.10
torch	2.4.1
fastapi	0.111.0
streamlit	1.34.0
qdrant-client	1.11.3
neo4j	5.26.0
FlagEmbedding	1.2.11

5.2 API 召回向量模型实验与评估

5.2.1 数据集介绍

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 5000 条微调数据，用于微调向量模型以构建 API 召回器。

其中数据格式如下所示，由于在原始数据集中已包含 Query 和正样本，我们实现了负样本的构造。在简单负样本构造中，负样本为随机抽取的工具；在困难负样本构造中，负样本为大语言模型生成得到的负样本。

5.2.2 模型训练及超参数设置

我们选择了 BGE-m3[77] 作为 API 召回器的基模型，主要基于其多功能性、多语言性和多粒度性的特性。首先，BGE-m3 具备多功能性，能够同时执行嵌入模型的三种常见功能：密集检索、多向量检索和稀疏检索。这种多功能性使模型在处理不同类型的检索任务时具有更高的灵活性。其次，BGE-m3 支持超过 100 种工作语言的多语言性，使得模型能够适应本文中的中英双语环境下的 API 召回下游任务，满足不同语言场景的需求。最后，BGE-m3 具有多粒度性，能够处理不同粒度的输入，从短句到长达 8192 个标记的长文档。这种特性使模型在处理各种长度的文本时都能保持良好的性能，体现了其优秀的泛化能力。

通过上述构建的数据集，我们采用对比学习来进行模型微调。对比学习（Contrastive Learning）是一种无监督表征学习方法，通过构建相似样本对（正样本对）和不相似样本对（负样本对），使模型学习到特征空间中语义相关样本彼此靠近，而无关样本彼此分离。InfoNCE（Information Noise-Contrastive Estimation）是一种对比损失函数，旨在将正样本对的相似性最大化，同时在多分类任务中最小化负样本对的相似性，以实现高质量的特征表征。假设我们有一组输入数据 $\{x_i\}_{i=1}^N$，每个样本 x_i 通过数据增强生成一组正样本（positive samples）x_i^+，负样本（negative samples）$\{x_j^-\}$ 则从批次中的其他样本中采样。定义映射函数 f_θ，将输入数据映射到一个潜在的特征空间，即 $f_\theta(x_i) \in \mathbb{R}^d$。InfoNCE 损失函数定义如下：

$$\mathcal{L}_{\text{InfoNCE}} = -\log \frac{\exp(\text{sim}(f_\theta(x_i), f_\theta(x_i^+))/\tau)}{\sum_{j=1}^N \exp(\text{sim}(f_\theta(x_i), f_\theta(x_j^-))/\tau)} \tag{5.1}$$

其中 $\text{sim}(u, v) = \frac{u \cdot v}{|u||v|}$ 表示余弦相似度，τ 为温度参数，控制相似度的平滑性和对负样本难度的敏感性。

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，

在一台带有 V100 芯片的 GPU 上训练，最终获得了微调后的工具召回器模型。

5.2.3　评估指标

我们使用召回率（Recall）[78-79]和 NDCG 分数[80]来衡量微调后的工具召回器的效果。

召回率用于衡量模型对正样本的识别能力，表示在所有实际正类样本中正确识别出的比例。召回率的计算公式为：

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5.2}$$

其中，TP 表示模型正确预测为正类的样本数量，FN 表示实际为正类但被模型错误预测为负类的样本数量。较高的召回率意味着模型能够识别出更多的正类样本，但可能导致误报率增加。

NDCG（Normalized Discounted Cumulative Gain）用于评价排序效果，通过对结果的排序位置进行权重衰减，衡量模型在不同相关性得分下的排序质量。NDCG 先计算 DCG（Discounted Cumulative Gain）：

$$\text{DCG}_p = \sum_{i=1}^{p} \frac{\text{rel}_i}{\log_2(i+1)} \tag{5.3}$$

其中，rel_i 是位置 i 上结果的相关性评分，i 是该结果在列表中的位置，排名越靠前权重越大。为了归一化，NDCG 将 DCG 除以理想状态下的 IDCG（Ideal DCG）：

$$\text{NDCG}_p = \frac{\text{DCG}_p}{\text{IDCG}_p} \tag{5.4}$$

其中 IDCG 是将结果按照最高相关性得分排序后的 DCG 值。NDCG 的值范围在 0 到 1 之间，越接近 1 表示排序效果越理想。本研究中，使用 NDCG@3 和 NDCG@8 分别评估排名前 3 和前 8 个结果的排序效果。

5.2.4　基线模型

为了进一步评估本文提出的面向工具领域的工具召回器微调方法的有效性，本工作选用开源的其他向量召回模型作为基线模型，包括面向通用领域的向量召回模型和面向工具召回的向量召回模型，其中面向通用领域的向量召回模型为 BGE-m3[77]，m3e-base[81]，经过工具数据微调的模型为 APIRetriever[2]。

表 5.3 API 工具召回实验结果

方法	Recall@5			NDCG@1			NDCG@5		
	I1	I2	I3	I1	I2	I3	I1	I2	I3
m3e-base	00.0	00.0	00.0	00.0	32.7	29.4	33.8	37.0	34.5
BGE-m3	53.4	50.2	51.5	36.1	38.4	37.5	43.2	48.1	45.2
APIRetriever	00.0	00.0	00.0	84.2	68.2	81.7	89.7	77.9	87.1
Ours (Simple Negatives)	68.9	66.7	67.5	52.5	63.0	55.7	59.5	71.0	65.2
Ours (Hard Negatives)	**71.3**	**69.1**	**70.4**	**55.0**	**66.2**	**58.4**	**61.8**	**73.1**	**69.0**

5.2.5 实验结果

表 5.4 显示了不同方法在召回率（Recall@5）、NDCG@1 和 NDCG@5 指标上的表现。

针对改实验结果的分析如下：

- **召回率 (Recall@5) 的提升**：我们的模型在 Recall@5 上达到了 71.3%，相比 BGE-m3 提升了 17.9 个百分点。ToolBench 的 API 召回器的召回率为 57.8%，比本方法低 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@1 的提升**：在关注最前排名结果的 NDCG@1 指标上，我们的方法在 I1 和 I2 数据集上分别达到了 55.0 和 66.2，相比 ToolBench 的 API 召回器平均提升约 16 个百分点。这表明我们的模型在首位召回结果的相关性排序上更为精准，使得用户更有可能在首个推荐结果中获得最相关的工具。
- **NDCG@5 的提升**：在关注前 5 个排序结果的 NDCG@5 指标上，本方法在 I1 和 I2 数据集上分别达到了 61.8 和 73.1，相比于 ToolBench 的 API 检索器平均提升约 19.1 个百分点。这表明本方法在扩大范围至前 5 个结果时，仍然能够保持较高的相关性排序质量，帮助用户在较大候选集中找到更相关的工具。
- **任务难度对模型表现的影响**：所有方法在跨任务的数据集 I1（单工具任务）到 I2（工具链任务）时都表现出一些差异，I2 数据集的召回任务对相关性排序提出了更高的要求。尽管如此，本方法在 I2 数据集上的表现仍然优于其他基线方法，说明在更高任务复杂度的情境下，我们的模型具有更强的适应性和鲁棒性。
- **召回率与排序质量的平衡**：从 Recall@5 和 NDCG 指标的综合结果可以看出，我们的模型在保证高召回率的同时，也能够有效地将高相关性的工具排在前列。这对于工具召回任务尤为重要，特别是在需要优先呈现最佳工具的应用情境下。

相比基线方法，我们的方法在前 5 个结果中提供了更高的 NDCG 值，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。特别是在更高复杂度任务的用户需求中表现突出，进一步验证了基于对比学习微调的 BGE-m3 模型在本文的工具召回场景的适用性。

5.3　基于工具图谱与深度优先遍历的 API 编排与调用方法实验

5.3.0.1　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。

5.3.0.2 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。

5.3.0.3　数据集评估和筛选

为了评估和验证数据集的质量，我们对生成的数据进行了深入的评估，并根据评估进行了数据集的筛选，以下为评估的具体过程与评估结果。

我们参考了[39]中的三个评估指标：1. 自然性 2. 复杂性 3. 一致性，用于衡量数据集的质量，每项目指标评分为 1-5 分。

我们选择了两种基线方法与该构造的数据进行对比和打分，分别为:。

在人工评估中，我们随机抽取了 50 个样本并对这些样本进行质量评估。为了确保评估的公平和客观性，所有样本都是匿名进行评估的。同时我们提供了详细的打分标准，以校准评估标准，最终结果为三位专家评分的平均值，结果如下表所示5.4。

表 5.4　数据集评估结果对比（评分范围：1-5 分）

方法	自然性 ↑	复杂性 ↑	一致性 ↑	综合评分 ↑
构造数据集（我们的方法）	3.82	3.90	3.87	3.86
方法 A（简单生成路径）	3.34	3.40	3.28	3.34
方法 B（人工设计路径）	3.72	3.75	3.84	3.77

最终结果表情该数据集与基线数据集在打分上较为相似，证明了该数据集的有效性和优越性。

5.3.1　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[82]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。

- **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:

$$PR = \frac{\#(\text{Solved})}{\#(\text{Solved}) + \#(\text{Unsolved})}. \tag{5.5}$$

- **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：

$$WR = \frac{\#(\text{Won})}{\#(\text{Won}) + \#(\text{Lost}) + \#(\text{Tie})}. \tag{5.6}$$

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，与 GPT-4 的标注结果进行了对比。

表 5.5 Consistency between Evaluator and Human Annotator

Annotation Type	Number of Human-Annotated Samples	Consistency
Pass Rate	100 samples	87%
Win Rate	100 samples	76%

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 87%，在胜率上该数字达到了 76%，这表明基于大语言模型的标注与人工标注的标准基本吻合。

5.3.2 基准线

本文选用下列方法作为实验的基准方法。

- **基本提示方法（Vanilla）**。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（CoT）**[16]。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**[21]。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。

5.3.3 实验结果

在大语言模型的选择上，我们选择了 Qwen2.5-7b 和 GPT-3.5 作为基础模型，选择原因是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。两个模型的实验均在构建的测试集上进行，并通过通过率 (Pass Rate) 和胜率 (Win Rate) 作为评判指标。实验结果如表 5.6 所示。

Model	Method	I1-Single Tool		I2-Tool Chain		Average	
		Pass	Win	Pass	Win	Pass	Win
ChatGPT	Vanilla	32.3	28.7	28.4	24.5	30.4	26.6
	CoT	42.6	38.3	38.9	35.1	40.8	36.7
	ReACT	46.2	44.8	42.3	40.4	44.2	42.6
	Ours	54.1	51.9	50.7	48.3	52.4	50.1
	Ours+ToolRetriever	56.4	53.2	52.8	50.5	54.6	51.9
Qwen-2.5	Vanilla	28.2	24.9	24.5	20.7	26.4	22.8
	CoT	38.3	34.6	34.7	30.8	36.5	32.7
	ReACT	42.7	40.3	38.4	36.1	40.6	38.2
	Ours	50.8	48.5	46.9	44.4	48.9	46.5
	Ours+ToolRetriever	52.5	50.2	48.6	46.3	50.6	48.2

表 5.6　Results for I1 and I2 with Pass and Win metrics for different methods

实验结果　表 5.6 展示了不同方法在两类任务 (I1-Single Tool 和 I2-Tool Chain) 上的通过率和胜率。从结果可以看出，本研究方法（Ours 和 Ours+ToolRetriever）在所有指标上均优于其他基线方法，具体分析如下：

- **通过率 (Pass Rate) 的提升**：在 I1 数据集上，Ours 方法的通过率为 54.1%，相比于 ReACT 方法的 46.2%，提高了 7.9 个百分点；相比于思维链方法 (CoT) 提高了 12.1 个百分点。在进一步优化的 Ours+ToolRetriever 方法中，通过率提升至 56.4%，再次提升了 2.3 个百分点。在更具挑战性的 I2 数据集上，Ours 方法的通过率为 50.7%，相比于 ReACT 方法的 42.3%，提高了 8.4 个百分点；相比于思维链方法提高了 11.8 个百分点。Ours+ToolRetriever 在 I2 数据集上的通过率为 52.8%，比 Ours 方法进一步提升 2.1 个百分点。这表明我们的改进方法在应对更复杂任务时表现更为稳健。
- **胜率 (Win Rate) 的优势**：在 I1 数据集上，Ours 方法的胜率为 51.9%，高于 ReACT 方法的 44.8%，提升了 7.1 个百分点；相比于思维链方法的 38.3%，提升了 13.6 个百分点。在 Ours+ToolRetriever 方法中，胜率进一步提升至 53.2%，略高于 Ours 方法。在 I2 数据集上，Ours 方法的胜率为 48.3%，相比于 ReACT 方法的 40.4%，提高了 7.9 个百分点；相比于思维链方法的 35.1%，提高了 13.2 个百分点。而 Ours+ToolRetriever 方法则将胜率提升至 50.5%，相较 Ours 方法提升了 2.2 个百分点。这表明我们的方法不仅在通过率上具有优势，在胜率的指标上也显现出对路径合理性的强适应性。
- **不同任务复杂度的对比**：可以观察到，所有方法在更复杂的 I2 数据集上的通过

率和胜率均低于 I1 数据集。然而，本方法（尤其是 Ours+ToolRetriever）在 I2 数据集上依然保持了较高的通过率和胜率，与其他基线方法相比表现出了显著的优势。这说明本方法在复杂任务上的适应性更强，能够有效处理多工具场景中的复杂性。

5.3.4 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 26% 是由于 API 接口调用错误造成的。这类错误包括参数缺失、API 调用结果错误等。
2. **初始节点候选集问题**：有约 15% 的失败案例源于初始节点候选集的选择错误。这是由于根据相似度的 API 召回器未召回到合适的初始节点，导致后续节点选择出现寻找方向错误，或者是有限空间内无法搜索得到目标工具。在本文中已经对 API 召回器进行了微调，未来可以通过更深入的困难负样本挖掘或者是将用户需求拆解为更细粒度、具体的子任务来解决。
3. **工具幻觉现象**：有 28% 的出错用户 query 出现了“工具幻觉”现象，即模型选择了并不存在或不适合当前任务的工具节点。这种错误选择不仅增加了不必要的 token 消耗数量，导致了额外计算资源和时间的浪费，还显著降低了任务完成率。
4. **其他错误**：剩余的错误都是由于以上错误组合而成，或是由于其他原因造成的，如路径寻找错误或者是在有限时间内未能找到合适的 API 路径。

5.4 本章小结

本章全面分析了基于工具图谱和深度优先遍历策略的 API 调用方法以及 API 召回模型的实验结果。在工具召回方面，我们提出了一种基于 BGE-m3 模型的对比学习微调方法，通过简单和困难负样本生成有效优化了召回模型的性能。本文通过在公开工具数据集上开展的详细实验，并与通用向量模型和面向工具召回场景的向量模型等基线模型进行对比，验证了本文方法的有效性。

此外，我们还对基于工具图谱的 API 编排与调用方法进行了详细的实验分析，并与其他基于提示词工程和流程设计的方法进行了比较，通过在不同难度任务的测试集上进行测试，我们验证了该方法的优越性。同时，为了检验不同部分在方法中的贡

献度，我们设计了有关工具图谱的消融实验。实验表明，工具图谱能够有效减少无效调用和优化任务路径，而边权值和点权值的设计进一步提升了模型对工具依赖关系和优先级的判断能力。

总体而言，本章通过多个实验验证了方法的有效性，该方法为复杂任务中的工具选择与调用提供了高效的解决方案。

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。

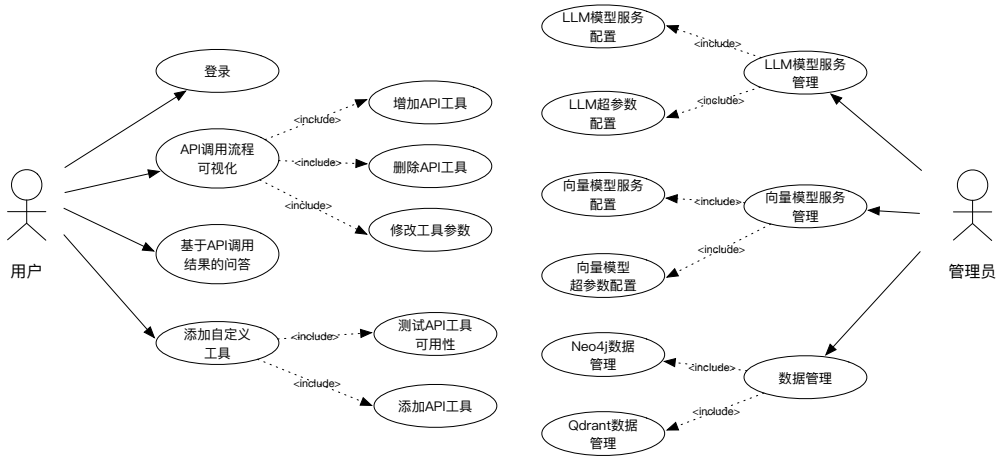

图 6.1 系统用例图
Figure 6.1 System Usecase Diagram

我们采用面向对象的需求分析方法，绘制了系统的用例图。在该用例图中，我们定义了两种主要角色：系统用户和管理人员。系统用户包含普通用户和开发者，而管

理人员则具有更高权限的管理功能。这样的角色划分帮助系统满足不同用户的需求，支持工具调用、数据管理、模型管理等任务。根据需求分析，系统应具备以下核心功能模块：

1. 用户与权限管理：系统应提供用户注册、登录、身份验证和权限分配功能。系统用户可根据权限访问相应模块，而管理人员具有配置用户权限的能力，以保障系统安全性和合理性。

2. API 调用流程管理与问答支持：系统支持用户通过可视化界面查看并编辑 API 调用流程，并能基于调用结果进行问答。API 调用流程管理模块可展示调用步骤、输入输出等信息，用户可根据需要进行简单调整。问答模块则负责对编排好的工具流程进行调用和解析，生成总结性描述或回答用户问题。

3. 自定义工具与工具库管理：系统支持用户根据需求添加自定义工具，丰富工具库的扩展性。用户通过填写工具的配置信息并测试后，系统将自动集成到工具库，供用户选择和调用。同时，工具库还提供常用工具模板，方便用户浏览和直接调用。

4. 模型服务管理：系统为管理人员提供大语言模型和向量模型服务的管理功能。该模块支持模型的配置、更新和监控，以满足多种任务需求并优化系统性能。管理人员可以根据需要对模型进行更新，以确保系统提供高效的模型服务。

5. 数据库管理：系统应提供高效的图数据库和向量数据库管理功能，以支持管理员对不同类型的数据进行配置和管理。对于图数据库管理，平台应支持 Neo4j 等图数据库的配置，允许管理员查看数据库状态、管理数据结构和优化查询性能，以确保关系数据的高效存储和检索。对于向量数据库管理，平台应支持 Qdrant 等向量数据库的配置，允许管理员调整存储策略和搜索参数，以优化向量数据的存储和检索性能。系统支持大规模向量数据的导入、分片配置和索引更新，确保能够高效完成相似度计算，满足推荐系统和语义搜索等场景的需求。

6.2　交互方案设计

图 6.3为该系统的交互设计图。

本系统主要有以下三个部分，前端交互界面、系统后端和其他服务模块，集成了不同模型的调用接口以及知识图谱查询功能，为用户提供了低门槛、用户友好的界面，支持用户通过自然语言的方式进行交互和提问。整体交互流程如图 6.2 所示：

1. 前端页面：前端页面为用户提供了交互界面，支持大模型对话、历史调用记录以及自定义 API 工具添加等功能。用户可以通过这些页面与系统进行自然语言交

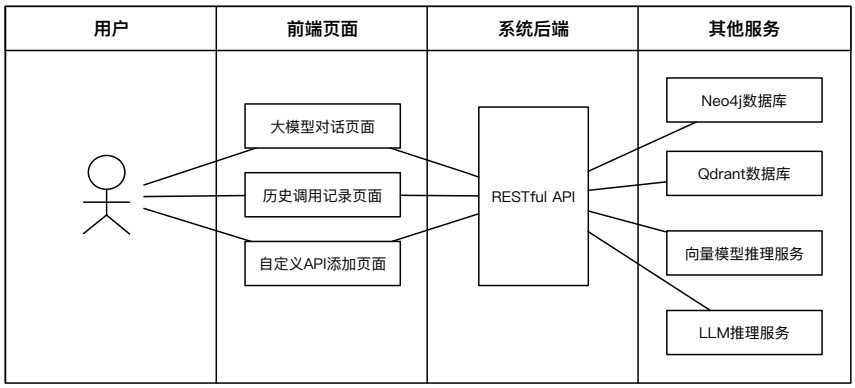

图 6.2 系统交互图
Figure 6.2 System Usecase Diagram

互，例如通过对话界面向大模型提问、查看历史调用模板获取 API 使用信息，或在自定义页面添加新的 API 工具以扩展系统功能。2. 系统后端：系统后端作为核心控制层，通过 RESTful API 接口将前端用户请求传递至其他服务模块，并负责业务逻辑的处理。后端系统不仅管理用户请求的权限和数据存储，还充当 API 编排的核心，确保前端请求的安全性和规范性。在与其他服务模块的交互中，后端可以灵活调用图数据库、向量数据库以及模型推理服务，实现对知识图谱的查询和不同模型的协同调用，以满足复杂的查询和推理需求。3. 其他服务模块：其他服务模块包括 Neo4j 图数据库、Qdrant 向量数据库、向量模型推理服务和 LLM（大语言模型）推理服务。这些模块分别用于存储和查询图数据库、管理和检索向量化数据、支持基于向量相似度的搜索需求，以及提供大语言模型的对话和推理功能。系统后端根据用户需求对这些服务进行编排调用，确保用户问题得到高效而准确的解答。

通过以上三部分的协同工作，系统能够为用户提供全面的 API 编排调用智能问答、历史调用模板查询和自定义工具配置等服务，满足用户的需求，并提供了易用、易扩展的系统和良好的交互体验。

6.3 系统架构设计

本系统架构分为五个层次：存储层、访问层、功能层、接口层和展示层。以下对各层的功能和组成模块进行详细介绍。

图 6.3为该系统的系统软件架构设计图。

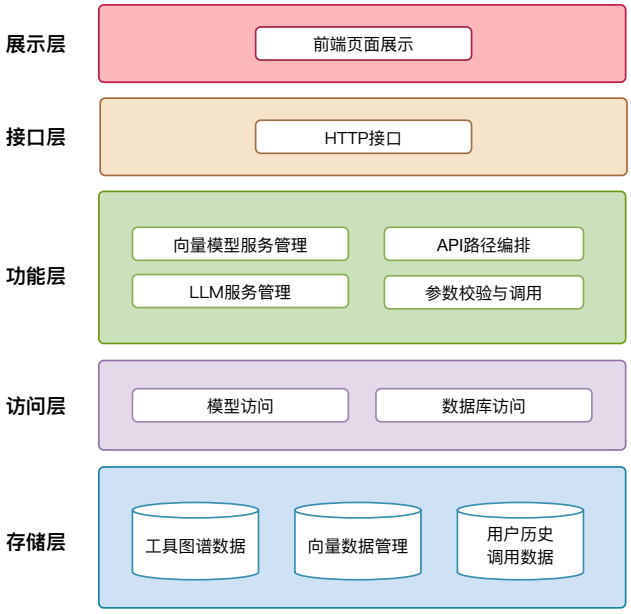

图 6.3　系统用例图
Figure 6.3　System Usecase Diagram

6.3.1　存储层

存储层负责系统核心数据的存储，包括以下内容：

- **API 知识图谱**：用于存储 API 的知识关联信息，方便后续的关系检索和动态编排。
- **API 详细信息的向量存储**：将 API 的详细信息预先计算并存储在向量数据库中，以便于快速计算相似度。
- **历史调用路径**：记录系统使用过程中产生的调用路径，作为经验数据供后续参考和优化。

6.3.2　访问层

访问层负责封装对存储层的访问功能，提供统一的数据访问接口，便于功能层的调用。其主要功能包括：

- 访问图数据库和向量数据库的数据，进行知识图谱检索和相似性计算。
- 访问和解析 JSON 格式的文件数据，确保数据在各层之间的顺畅传递。

6.3.3　功能层

功能层是系统的核心，实现动态 API 编排和调用。主要模块包括：

- **任务分解模块**：输入用户的复杂需求，输出为拆解后的固定格式的一组子任务，子任务之间相互独立。
- **长期记忆检索模块**：根据当前任务描述，在历史调用路径中检索相似任务描述和历史调用路径，提供给当前任务的动态 API 编排模块作为参考。
- **API 知识图谱检索模块**：按照不同的检索模式搜索 API 节点及其关联的邻居节点信息。
- **基于图谱的 DFS 动态编排算法模块**：在知识图谱上执行深度优先搜索和回溯操作，生成最优的调用路径。
- **API 调用模块**：获取目标 API 的参数信息，并结合用户需求生成 API 调用的输入参数。使用生成的参数调用 API，同时设置自动重试机制，以确保调用的稳定性；若调用失败则进行重试，若多次重试仍失败则返回错误信息。
- **API 调用结果总结模块**：将 API 调用结果转换为自然语言格式，并结合用户的原始需求生成最终的总结性回复。

6.3.4 接口层

接口层负责为系统各功能提供访问接口，采用 RESTful API 的 HTTP 接口形式：

- 系统后端通过 RESTful API 接口与前端交互，供前端进行数据访问和操作。
- 模型服务与平台后端之间也通过 HTTP 接口进行通信，以确保数据传输的安全性和高效性。

6.3.5 展示层

展示层是面向用户的 web 前端页面，为用户提供直观友好的交互界面。主要页面包括：

- **信息问答页面**：用户可通过自然语言输入需求，系统将自动编排 API 调用路径并执行，最终以自然语言格式返回结果，模拟多轮对话的交互体验。
- **自定义 API 添加页面**：用户填写 API 名称、调用链接、参数类型和描述信息等来添加新 API。新 API 需经过测试验证后方可加入 API 仓库。
- **API 历史调用参考页面**：用户可以查看历史 API 调用记录，并支持通过选择和配置参数重新使用历史 API 调用链。

6.4　模块设计

图 6.4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。

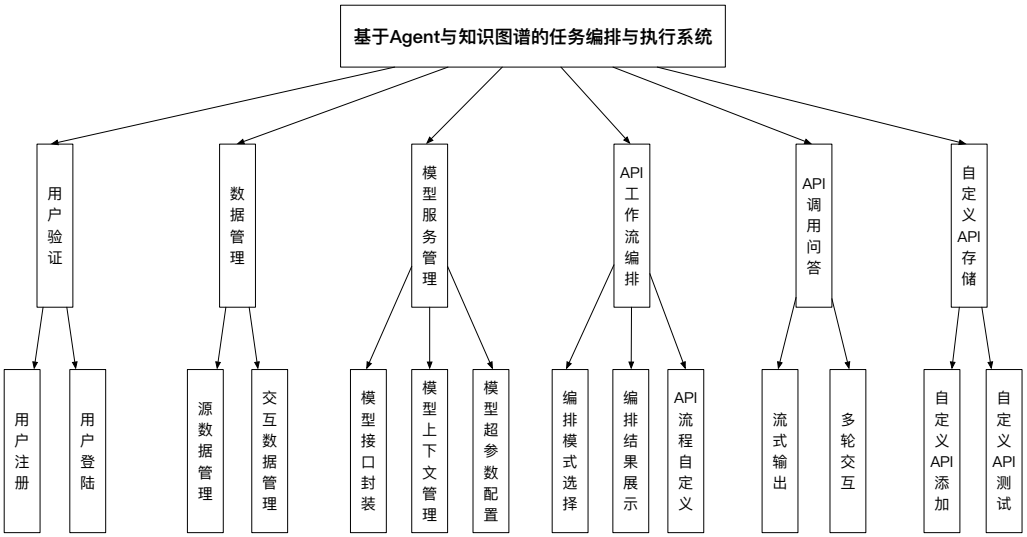

图 6.4　系统模块图
Figure 6.4　System Module Diagram

6.4.1　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。

6.4.2　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。

6.4.3　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。

6.4.4 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。

6.4.5 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。

6.4.6 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。

6.5 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。

在本地大语言模型部署时，我们选择 vLLM 来部署 Qwen 模型，vLLM 能提供业内顶尖的推理效率、高效的注意力管理机制和对模型量化的支持，还能够将模型部署为 OpenAI API 格式，提供了方便的交互。我们在该系统采用了前后端分离的方式，选择了 FastAPI 作为后端框架，将除模型推理服务外的服务都以 API 接口的形式提供，前后端通过 HTTP 进行通讯。在前端页面部分，我们选用了目前大语言模型有关应用有关最流行的前端框架之一———Streamlit 前端框架来实现。

6.6 本章小结

本章设计并实现了一个基于知识图谱和大语言模型智能体机制的 API 编排与调用系统。系统通过集成知识图谱查询、模型管理和向量相似度计算功能，为用户提供自然语言交互的低门槛界面，并支持复杂需求的 API 调用和回答生成。系统架构分

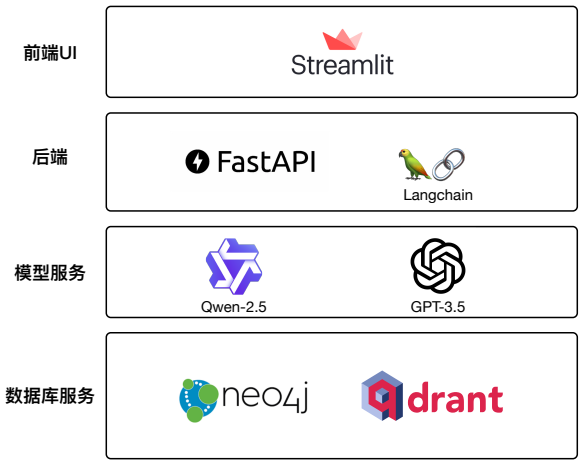

图 6.5　系统模块图

Figure 6.5　System Implementation Diagram

为存储层、访问层、功能层、接口层和展示层，涵盖从数据存储与检索到用户交互的全链路设计。主要功能包括任务分解模块与 API 动态编排、API 调用流程管理、历史调用参考、自定义工具扩展以及多轮问答支持。系统实现采用 FastAPI 作为后端框架，结合 Streamlit 前端和 Neo4j、Qdrant 等数据库，实现了模块化、用户友好、可扩展的交互体验，满足了多样化的用户需求并提供高效的 API 编排与调用服务。

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

本文从大语言模型在复杂任务场景中的应用挑战出发，提出了一套完整的解决方案，包括工具图谱构建、动态工具编排与调用方法，以及系统设计与实现，旨在探索如何通过大语言模型与知识图谱的结合实现高效的工具调用和任务解决方案。具体内容涵盖以下三方面：

1. 本文提出了一种基于工具调用路径的工具图谱构建方法，系统性地完成了从数据筛选到图谱验证的全流程。通过提取高质量工具组和调用路径，设计了工具知识图谱的概念模型，以描述工具的层次关系与依赖逻辑，构建了工具图谱。该图谱为复杂任务中的工具选择与规划提供了重要支撑，显著提升了工具调用的准确性和效率。
2. 为应对工具调用过程中的动态性和复杂依赖关系，本文设计了一种基于知识图谱的动态编排与调用方法。通过任务分解模块将复杂需求拆解为可执行的子任务，并结合深度优先搜索算法在图谱中动态规划工具路径，实现了工具调用的灵活性与精确性。引入记忆机制和自我反思机制，进一步提高了调用过程的稳定性和适应性，为多样化任务需求提供了高效解决方案。
3. 本文基于知识图谱和大语言模型智能体机制，设计并实现了一个用户友好的 API 编排与调用系统。系统采用模块化架构，涵盖工具管理、任务规划与执行、以及智能问答功能，为用户提供了低门槛的交互体验。通过自然语言解析需求，系统能够自动完成 API 调用流程，实现了工具调用的自动化和可视化，展现了高效性与扩展性。

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调用路径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] HUANG S, ZHONG W, LU J, et al. Planning, Creation, Usage: Benchmarking LLMs for Comprehensive Tool Utilization in Real-World Complex Scenarios[J]. arXiv preprint arXiv:2401.17167, 2024.

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

[3] QU C, DAI S, WEI X, et al. Tool Learning with Large Language Models: A Survey[J]. arXiv preprint arXiv:2405.17935, 2024.

[4] 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.

[5] THEUMA A, SHAREGHI E. Equipping Language Models with Tool Use Capability for Tabular Data Analysis in Finance[J]. arXiv preprint arXiv:2401.15328, 2024.

[6] HAO Y, CHEN Y, ZHANG Y, et al. Large Language Models Can Plan Your Travels Rigorously with Formal Verification Tools[J]. arXiv preprint arXiv:2404.11891, 2024.

[7] YE J, LI G, GAO S, et al. Tooleyes: Fine-grained evaluation for tool learning capabilities of large language models in real-world scenarios[J]. arXiv preprint arXiv:2401.00741, 2024.

[8] SCHICK T, DWIVEDI-YU J, DESSÌ R, et al. Toolformer: Language models can teach themselves to use tools[J]. Advances in Neural Information Processing Systems, 2024, 36.

[9] HAO S, LIU T, WANG Z, et al. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings[J]. Advances in neural information processing systems, 2024, 36.

[10] PARISI A, ZHAO Y, FIEDEL N. Talm: Tool augmented language models[J]. arXiv preprint arXiv:2205.12255, 2022.

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

[12] 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.

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

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

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

[16] WANG X, WEI J, SCHUURMANS D, et al. Self-Consistency Improves Chain of Thought Rea-

soning in Language Models[EB/OL]. arXiv. 2023. http://arxiv.org/abs/2203.11171.

[17] 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.

[18] WANG X, WEI J, SCHUURMANS D, et al. Self-consistency improves chain of thought reasoning in language models[J]. arXiv preprint arXiv:2203.11171, 2022.

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

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

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

[26] ANANTHA R, BANDYOPADHYAY B, KASHI A, et al. ProTIP: Progressive Tool Retrieval Improves Planning[J]. arXiv preprint arXiv:2312.10332, 2023.

[27] 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.

[28] HSIEH C Y, CHEN S A, LI C L, et al. Tool documentation enables zero-shot tool-usage with large language models[J]. arXiv preprint arXiv:2308.00675, 2023.

[29] ZHANG Y, CAI H, SONG X, et al. Reverse chain: A generic-rule for llms to master multi-api planning[J]. arXiv preprint arXiv:2310.04474, 2023.

[30] YUAN S, SONG K, CHEN J, et al. Easytool: Enhancing llm-based agents with concise tool instruction[J]. arXiv preprint arXiv:2401.06201, 2024.

[31] SHI Z, GAO S, CHEN X, et al. Learning to use tools via cooperative and interactive agents[J]. arXiv preprint arXiv:2403.03031, 2024.

[32] WANG Z, CHENG Z, ZHU H, et al. What are tools anyway? a survey from the language model perspective[J]. arXiv preprint arXiv:2403.15452, 2024.

[33] SHEN Y, SONG K, TAN X, et al. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face[J]. Advances in Neural Information Processing Systems, 2024, 36.

[34] XU F, SHI W, CHOI E. Recomp: Improving retrieval-augmented lms with compression and selective augmentation[J]. arXiv preprint arXiv:2310.04408, 2023.

[35] 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.

[36] 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.

[37] 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.

[38] WU X, SHEN Y, SHAN C, et al. Can Graph Learning Improve Planning in LLM-based Agents? [C]//The Thirty-eighth Annual Conference on Neural Information Processing Systems.

[39] SHEN Y, SONG K, TAN X, et al. Taskbench: Benchmarking large language models for task automation[J]. arXiv preprint arXiv:2311.18760, 2023.

[40] SCHNEIDER E W. Course modularization applied: The interface system and its implications for sequence control and data analysis[C]//Proceedings of ADIS. 1973.

[41] SINGHAL A. Introducing the Knowledge Graph: Things, not strings[Z]. Google Blog. Retrieved from https://www.blog.google/products/search/introducing-knowledge-graph-things-not/. 2012.

[42] ZOU X. A survey on application of knowledge graph[C]//Journal of Physics: Conference Series: vol. 1487: 1. 2020: 012016.

[43] NOY N, GAO Y, JAIN A, et al. Industry-scale Knowledge Graphs: Lessons and Challenges: Five diverse technology companies show how it's done[J]. Queue, 2019, 17(2): 48-75.

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

[45] ACHIAM J, ADLER S, AGARWAL S, et al. Gpt-4 technical report[J]. arXiv preprint arXiv:2303.08774, 2023.

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

[47] Anthropic. Claude 3 (Oct 8 version) [Large language model][Z]. https://www.anthropic.com/. Accessed: 2024-11-19. 2023.

[48] TEAM G, ANIL R, BORGEAUD S, et al. Gemini: a family of highly capable multimodal models [J]. arXiv preprint arXiv:2312.11805, 2023.

[49] TEAM G, GEORGIEV P, LEI V I, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context[J]. arXiv preprint arXiv:2403.05530, 2024.

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

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

[52] YANG A, YANG B, HUI B, et al. Qwen2 technical report[J]. arXiv preprint arXiv:2407.10671, 2024.

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

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

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

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

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

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

[59] 郭先会, 张梦姣, 马军. 基于大语言模型的智能体构建综述[J]. 通信技术, 2024, 57(09): 873-879.

[60] 李国鹏, 吴瑞骐, 谈海生, 等. 面向大语言模型驱动的智能体的计划复用机制[J]. 计算机研究与发展, 2024, 61(11): 2706-2720.

[61] 叶国林, 郭家文, 张晓宇, 等. 基于开源大语言模型的智能体算法研究[J]. 中国信息界, 2024(04): 161-163.

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

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

[64] FAN W, DING Y, NING L, et al. A survey on rag meeting llms: Towards retrieval-augmented large language models[C]//Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. 2024: 6491-6501.

[65] 赵静, 汤文玉, 霍钰, 等. 大模型检索增强生成（RAG）技术浅析[J]. 中国信息化, 2024(10): 71-72+70.

[66] 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.

[67] 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/2210.12925v1. DOI: 10.18653/v1/2022.emnlp-main.555.

[68] 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.

[69] JI Z, LEE N, FRIESKE R, et al. Survey of hallucination in natural language generation[J]. ACM

Computing Surveys, 2023, 55: 1-38.

[70] 张鹤译, 王鑫, 韩立帆, 等. 大语言模型融合知识图谱的问答系统研究[J]. 计算机科学与探索, 2023, 17(10): 2377-2388.

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

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

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

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

[75] 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.

[76] LIU W, HUANG X, ZENG X, et al. Toolace: Winning the points of llm function calling[J]. arXiv preprint arXiv:2409.00920, 2024.

[77] CHEN J, XIAO S, ZHANG P, et al. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation[J]. arXiv preprint arXiv:2402.03216, 2024.

[78] BUCKLAND M, GEY F. The relationship between recall and precision[J]. Journal of the American society for information science, 1994, 45(1): 12-19.

[79] POWERS D M. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation[J]. arXiv preprint arXiv:2010.16061, 2020.

[80] WANG Y, WANG L, LI Y, et al. A theoretical analysis of NDCG type ranking measures[C]// Conference on learning theory. 2013: 25-54.

[81] Moka AI. M3E-base: Multilingual Embedding Model[EB/OL]. 2024. https://huggingface.co/moka-ai/m3e-base.

[82] 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.

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。

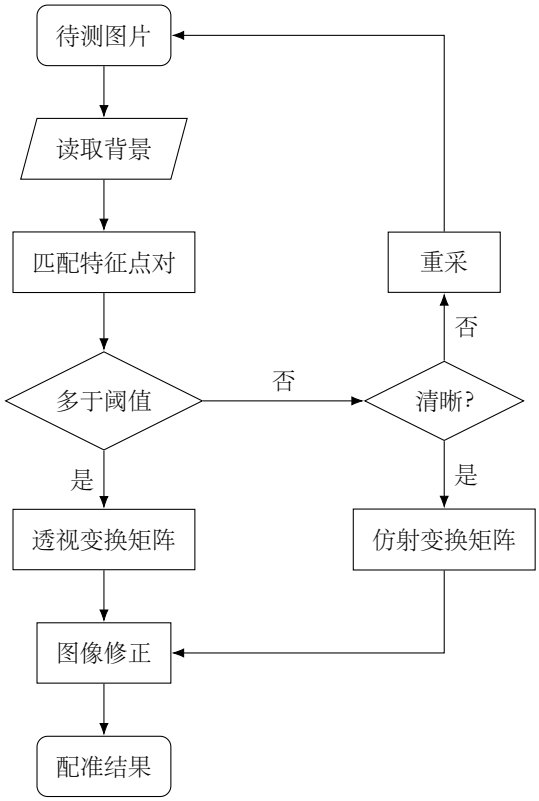

图 B.1 绘制流程图效果

Figure B.1 Flow chart

致　谢

研究生时光悄然流逝，转眼间已经到了毕业的时刻。回望这两年多的学习和生活，我不仅学到了知识，更收获了许多关心和帮助。在这个特别的节点，我想真诚地感谢一路上支持和陪伴我的人。

首先，感谢我的导师吴刚老师。从大四开始，吴老师就一直是我的指导老师。在科研方向选择、论文写作、思路调整等方面，吴老师始终耐心地帮助我，为我解答疑惑、指明方向。这让我能够迅速融入交大的校园生活，顺利开启研究生阶段的学习和科研任务。我从吴老师身上学到了科研的严谨态度，也感受到了师者的无私关怀。

感谢我的行业导师赵俊峰老师。赵老师在学术和生活上都给予了我许多帮助，她让我有机会接触到各种工程项目和前沿技术，开阔了视野，也让我对行业有了更深的理解。赵老师的指导让我在实践中成长，是我研究生阶段的一笔重要财富。

感谢我的父母，他们始终是我最坚强的后盾。在求学路上，父母总是默默支持着我，即使对我的研究领域并不熟悉，也从不吝啬鼓励和陪伴。在我遇到挫折和困难时，他们的安慰和理解让我有了重新出发的勇气。我知道，没有他们的支持，就没有今天的我。

感谢南湖实验室。在这两年的联合培养中，实验室为我们提供了良好的科研和生活环境，让我能够专注于学术研究。这一年多的时间不仅让我学到了许多专业知识，也积累了许多美好的回忆。南湖实验室将永远是我研究生生活中最特别的一部分。

感谢我的朋友们，你们让我的研究生生活变得更加丰富和温暖。感谢我的室友李梦瑶同学，尽管我们性格迥异，但我们求同存异，始终彼此支持和帮助。感谢丁鸿馨同学，你的乐观和认真让我倍感鼓舞，认识你是我的幸运。感谢志远、瑞庆、方越师兄，在我遇到学术难题时，你们的耐心解答和宝贵建议为我指明了方向，让我感受到来自实验室的温暖。感谢朱润川同学。你的乐观和热心给身边的人带来了许多正能量。感谢江长江同学，你豁达的人生态度对我有许多启发，我也因你对人生有了不一样的理解，能够更加成熟地面对生活。

最后，我想感谢自己。不管是科研还是实习，这段路并不总是轻松，但我依然坚持到了最后，感谢自己未曾松懈。也感谢自己一直相信自己。

未来，我将带着这些珍贵的回忆与感激之情，继续前行，迎接人生新的篇章。

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.