



武汉大学
WUHAN UNIVERSITY

基于大语言模型的 Deepin 文档问答机器人

队伍名称：OS 全都队

| | |
|-----|---------|
| 成员： | 龙安迪（队长） |
| | 赵浩宇 |
| | 段信维 |

指导老师：蔡朝晖 钟鸣

赛题导师：吴荣杰

2023 年 05 月

摘 要

本作品为 2024 年全国大学生计算机系统能力大赛——操作系统设计赛（全国）OS 功能挑战赛道中 project225“文档问答机器人”的命题作品。基于大语言模型，团队开发了一个智能文档问答机器人，其能够对 Deepin 操作系统的 Wiki 文档进行理解、分类和检索，进而实现基于 Deepin 文档的问答。

本作品实现了对 Deepin 系统相关的文档预处理、数据库存储、根据用户输入的问题进行实体链接、在云端服务器部署文档问答机器人，用户能够直接在浏览器中通过公网访问 Deepin 问答机器人并向其提问。在后续比赛阶段，团队会进一步开发前端应用，改用 C/S 架构并将项目上传至 Deepin 应用商店，以便更好地将文档问答机器人与 Deepin 操作系统原生环境融合。

我们的赛题完成度如下：

| | 目标内容 | 完成情况 |
|------|----------------------------------|---|
| 基本目标 | 完成一个聊天机器人，能根据 Deepin wiki 内容回答问题 | 完成。我们开发了一个基于大模型的智能文档问答机器人，目前已经能实现 Deepin 内容文档的理解和基于文档的问答。 |
| 基本目标 | 通过训练模型，使回答和问题要有 80% 以上概率的相关性。 | 完成。我们的技术路线相较于传统 RAG 架构进行了改善，融入 LangChain 框架调用，本地部署 Chat GLM3-6b 大语言模型，基于 Deepin 文档数据集进行微调，加强大语言模型的效果。目前团队构建的文档问答机器人在文档检索时能够有 80% 以上的召回率，在答案生成时有 90% 以上的准确性。 |
| 基本目标 | 编写博客，记录开发过程，并投递。 | 完成。按照 Deepin 文档的指导，团队使用 HUGO 编写博客并已申请合并。 |
| 可选扩展 | 添加内容，支持回答玲珑使用的问题。 | 部分实现。团队已尝试通过爬虫爬取网页内容，只要爬取到玲珑使用手册文档，即可实现扩展。 |
| 可选扩展 | 添加可视化界面支持，能在 Deepin 系统上使用。 | 完成。团队搭建了简约的前端界面，前后端均部署于华为云弹性云服务器 ECS 并绑定公网 IP，可以在 Deepin 系统中通过浏览器访问本作品。后续将 |

| | | |
|--|--|---|
| | | 开发原生应用并上传至 Deepin 应用商城，进一步为 Deepin 用户使用文档问答机器人提供便利。 |
|--|--|---|

目 录

1 作品概述..... 1

 1.1 作品简介..... 1

 1.2 作品特色和创新..... 2

 1.3 预期目标..... 3

2 项目背景和相关资料分析..... 4

 2.1 命题背景..... 4

 2.2 需求分析..... 4

 2.3 问题重难点..... 5

 2.4 现有研究分析..... 6

3 系统框架设计..... 8

 3.1 总体技术流程..... 8

 3.2 大语言模型的本地部署与训练..... 9

 3.2.1 模型选择与本地部署..... 9

 3.2.2 模型微调..... 9

 3.3 文档获取与存储..... 10

 3.3.1 文档语料收集..... 10

 3.3.2 文档解析与转换..... 11

 3.3.3 数据库设计和文档存储索引..... 13

 3.4 文档检索与问答实现..... 14

 3.4.1 语音问题输入..... 14

 3.4.2 文档检索..... 15

 3.4.3 提示词构建和答案生成..... 16

 3.5 用户界面设计和系统部署..... 17

4 系统实现..... 19

 4.1 文档获取..... 19

 4.1.1 文本语料获取..... 19

 4.1.2 预处理..... 20

| | |
|-------------------------------|----|
| 4.2 ChatGLM3-6b 模型部署和微调 | 20 |
| 4.3 文档解析与索引存储 | 21 |
| 4.4 用户问题处理与文档检索 | 23 |
| 4.4.1 处理语音问题输入 | 23 |
| 4.4.2 问题实体识别 | 24 |
| 4.4.3 实体链接与文档检索 | 25 |
| 4.5 大语言模型调用与答案生成 | 25 |
| 4.6 系统搭建和部署 | 26 |
| 4.6.1 前端设计和开发 | 26 |
| 4.6.2 系统部署 | 27 |
| 4.7 作品功能呈现 | 28 |
| 4.7.1 文件上传 | 28 |
| 4.7.2 文字问答 | 30 |
| 4.7.3 语音问答 | 31 |
| 4.8 作品安装说明 | 32 |
| 4.8.1 系统访问 | 32 |
| 4.8.2 本地部署 | 32 |
| 5 测试分析 | 33 |
| 5.1 测试概要 | 33 |
| 5.1.1 测试目的 | 33 |
| 5.1.2 测试重点 | 33 |
| 5.1.3 测试环境 | 33 |
| 5.1.4 数据来源与规模 | 34 |
| 5.2 测试结果 | 34 |
| 5.2.1 文件上传功能测试 | 34 |
| 5.2.2 文字问答功能测试 | 35 |
| 5.2.3 语音问答功能测试 | 35 |
| 5.2.4 总体测试 | 36 |
| 5.3 测试结论 | 37 |

| | |
|-----------------|----|
| 6 未来展望 | 38 |
| 7 比赛进展和收获 | 39 |
| 7.1 分工和协作 | 39 |
| 7.2 收获与心得 | 39 |
| 8 参考文献 | 40 |

1 作品概述

1.1 作品简介

本作品为 2024 年全国大学生计算机系统能力大赛--操作系统设计赛（全国）——OS 功能挑战赛道中 project225“文档问答机器人”的命题作品。基于大语言模型，团队开发了一个智能文档问答机器人，其能够在特定场景的中文语料库中实现文档的理解、分类、检索，进而实现基于文档的问答。

本项目开发的文档问答机器人采用大语言模型检索增强生成（Retrieval Augmented Generation, RAG）技术，强化大语言模型对于文档的检索能力，提升问答准确性。同时应用了 LangChain 框架，增强对不同大语言模型的兼容性。

本作品的技术路线框架图如图 1.1 所示，简要设计流程如下：收集到 Deepin 系统文档数据后，对文本文档进行预处理和分块，并调用大模型生成关键词。Deepin Wiki 在文本中内嵌图片链接，用于解释文本内容。为了增加对这部分图片的语义理解，在扫描文本文档时一旦遇到图片链接，则加载对应图片一并提交大模型以识别关键词，加强关键词的准确性。将关键词转化为词向量，与索引信息一并存储在数据库中。根据用户输入的问题进行实体链接，寻找到相关文档块。将相关文档块内容与用户问题一并构建提示语段，遇到图片链接则加载对应图片，共同输入大语言模型，进而获取输出回答。基于上述步骤搭建文档问答机器人系统后，在云端服务器部署，用户能直接在 Deepin 操作系统中使用浏览器，通过公网访问问答机器人并向其提问。

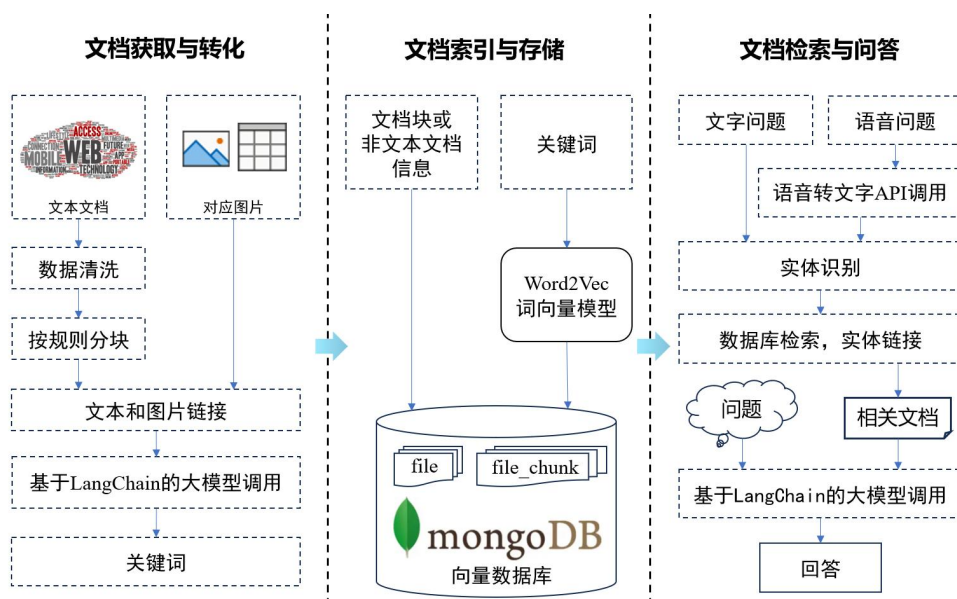


图 1.1 技术路线框架图

1.2 作品特色和创新

在构建针对 Deepin 系统的文档问答机器人时，我们选取的技术路线相较于传统 RAG 架构进行了改善，并融入了 LangChain 框架的调用，较好的完成了命题要求。通过创新点和技术实现，Deepin 系统文档问答机器人能够更好地满足用户对系统知识的需求，提供更准确、更便捷的解答服务。

针对命题要求，本作品进行了如下实现：

①完成 Deepin 文档问答机器人（必须）。我们开发了一个基于大模型的智能文档问答助手，目前已经能实现 Deepin 内容文档的理解和基于文档的问答。

②训练模型使回答和问题有 80%以上概率的相关性（必须）。在本地部署了经过训练的大语言模型，针对 Deepin 系统的数据和语境进行训练，能够更好地理解和生成与 Deepin 系统相关的回答。我们选取了九百多份文档中的一百份在 Deepin 系统中进行测试，目前团队构建的文档问答助手在文档检索时能够有 80%以上的召回率，在答案生成时有 90%以上的准确性。

③编写博客记录开发过程并投递（必须）。我们遵循 Deepin 文档中博客上传手册的相关要求，利用 HUGO 创建了团队的开发博客文件，目前已申请合并上传博客。

④支持回答玲珑使用的问题（选择）。团队已尝试通过爬虫爬取玲珑使用手册的网页内容，在后续可以批量爬取实现文档问答机器人的内容扩展。

⑤添加可视化界面能在 Deepin 系统上使用（选择）。团队搭建了简约的前端界面，前后端均部署于华为云弹性云服务器 ECS 并绑定了公网 IP，可以在 Deepin 系统中通过浏览器访问问答机器人并向其提问。

本作品进行了如下创新：

①支持多种文件类型。在传统 RAG 架构的文本文档基础上，新增了对与文本文档关联的图片文件的支持，丰富了知识库中的文件种类和数量。

②轻量化数据库存储。相较于 LangChain 框架在数据库中存储文档内容，采用 MongoDB 数据库存储文档块索引的方式，加快检索效率。

③文档块索引优化。在文档块之间引入一定量的重叠，增加上下文信息。相比检索后再延伸上下界的做法，在创建记录索引时便延伸上下界能够获得更加准确的关键词信

息，增加检索时的精度和召回率。

④多模态交互支持。除了传统的文本问答方式，我们还支持用户通过语音等多模态形式进行交互。用户可以通过语音输入问题来获取解答，使用户体验更加便捷高效。

1.3 预期目标

本作品预期将基于大语言模型构建一个专注于 Deepin 系统教程和词条的文档问答机器人，设计用户友好的前端 UI 界面并在公网部署，使用户能够便捷地访问问答机器人平台。团队预期在测试阶段收集九百多份以上的与 Deepin 系统相关的多源文档，涵盖从基础安装到高级配置的各个方面，包括一定的文本文档和对应的图片。团队构建的文档问答机器人预期在文档检索时能够达到 80% 以上的召回率，确保用户查询的大部分内容都能被系统检索到。同时，在答案生成方面追求 90% 以上的准确性，以提供用户准确、可靠的答案和建议。通过不断优化模型算法和丰富文档库，团队期待这一文档问答机器人能够成为 Deepin 系统用户的辅助工具，帮助他们更高效地学习和使用 Deepin 系统。

2 项目背景和相关资料分析

2.1 命题背景

随着人工智能技术的飞速发展，大语言模型（Large Language Model, LLM）作为新一轮产业变革的核心技术，正深刻改变着人类的生产生活方式。特别是以 ChatGPT 为代表的 AIGC 技术的崛起，为人工智能通用化开启了新的篇章。在这一背景下，如何有效地将大模型技术应用于具体场景，提升生产力，成为了业界共同探索的方向。

本命题要求设计一款基于大语言模型的 Deepin 系统文档问答助手类应用，该应用可实现在指定文档上进行检索、理解和交互，协助使用者高效阅读和查询。需要实现文档的解析与转化、用户交互界面设计，同时训练大语言模型提取文档中的关键信息和主体，识别重要段落、句子和术语，并将其归纳和总结，为用户提供基于 Deepin Wiki 文档的问答。

2.2 需求分析

在现代社会，信息差的存在导致个人花费更多时间进行知识的搜寻和筛选，增加了学习成本。信息差也导致企业在培养员工时，由于缺乏有效的信息传递和培训机制，需要投入大量的资源来进行员工培训。

同时，随着大数据时代的到来，企业和个人都面临着信息过载的问题。这样的信息爆炸导致用户面对海量的文档资料时，难以快速有效地获取所需信息。基于此，团队搭建的文档问答机器人旨在针对现有的信息差和信息爆炸问题，解决个人和企业两方面的需求。

对于个人而言，文档问答机器人能够帮助其快速找到所需的学习资源和针对问题的解答，减少在海量信息中筛选的时间，解决个人用户高效学习和知识检索的需求。

对于企业而言，文档问答机器人能够帮助其快速提取并整合文档中的关键信息，整理并归纳企业知识，进而建立私有知识库，对员工提供知识问答服务。员

工可以随时通过智能文档问答机器人进行知识问答和学习,大大减少了企业的对于员工的培训成本,解决企业提高知识管理效率、降低培训成本的需求。

2.3 问题重难点

针对命题提出的要求,本作品在设计和实现阶段具有技术实现、性能优化和拓展创新三个方面的重难点:

1. 技术实现

①文档语料收集:收集 Deepin 系统的文本语料信息,并进行清洗和预处理;

②文档解析与转化:使用合适的长度对文档进行分块并提取关键词,进而将关键词转化为词向量,涉及关键词提取算法和词向量模型的调用;

③文档存储:使用合适的数据库软件和结构进行文档索引和关键词词向量的存储,涉及数据库结构设计;

④系统部署与用户界面:将系统部署至云服务器,通过 B/S 架构实现系统运行,涉及云服务器搭建和 UI 前端设计。升级计划利用 DTK 框架在 qt 开发原生应用,并上传系统应用商城。

2. 性能优化

①算力资源:有效利用并获取算力资源,以实现模型的本地部署和微调;

②响应速度:通过优化系统结构和代码算法,提升系统响应速度,优化用户体验;

③回答准确率:通过模型的微调优化大语言模型的回答准确率。

3. 拓展创新

①大语言模型的本地部署和微调:选取合适的开源大语言模型并在本地部署,收集特定领域的语料训练集对模型进行微调,提升回答准确率;

②语音问题输入:集成语音识别技术,申请浏览器的语音权限,设计单独的语音传输流程,使用户能够通过语音提问,提升使用便捷性。

2.4 现有研究分析

针对基于文档的问答任务，目前普遍采用的方法为大语言模型的检索增强生成（Retrieval Augmented Generation, RAG）技术，即基于私有数据库进行信息检索，与问题合并生成提示词（Prompt）输入大模型，以获取基于文档内容的输出。RAG 基本架构如图 2.1 所示，该架构有效解决了大语言模型知识局限性的不足和幻觉问题，在私域部署大模型的情况下也能确保文档数据的私有化和安全性。

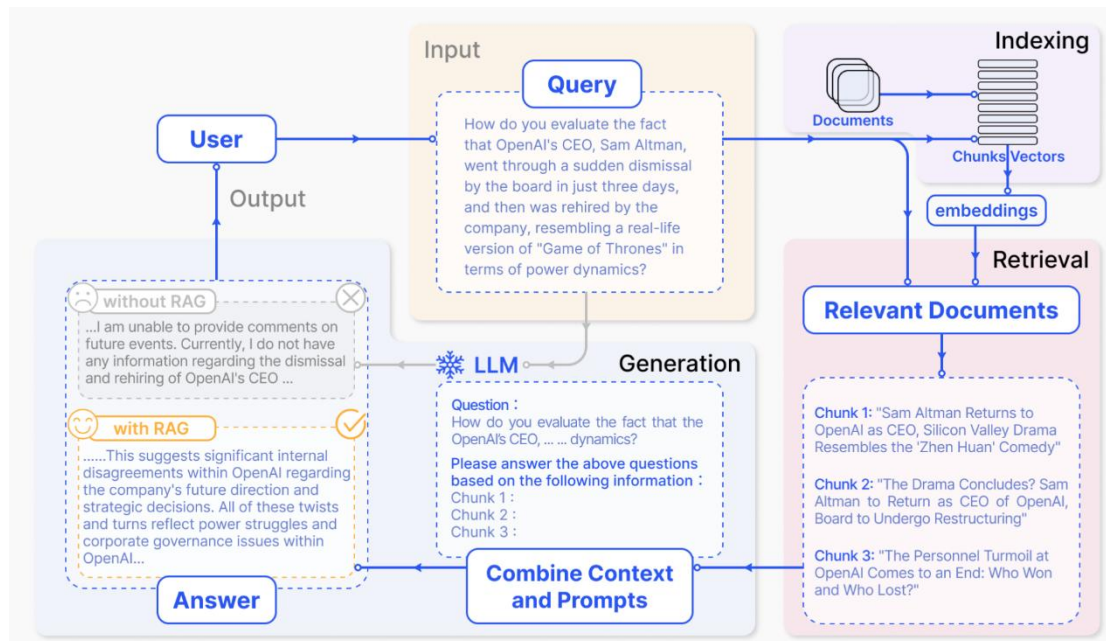


图 2.1 RAG架构图

Gao 等将 RAG 分为初级 RAG、进阶 RAG 和模块化 RAG。初级和进阶的 RAG 流程如图 2.2 所示，对此二者而言，其经典实现流程包括索引、检索和生成三个步骤。其中，索引部分是将原始数据清洗后分块存储，提取关键词，并利用词向量模型将关键词转化为词向量；检索部分是利用相同的词向量模型对问题进行词向量嵌入，与文档块计算相似度以找到相关文档块；生成部分是将问题和相关文档块组合成 Prompt 提示，进而调用大语言模型并获得答案输出。

基于此，研究者提出了进阶 RAG 架构，针对问题输入和文档检索进行了一定的优化。进阶 RAG 架构通过改变块的大小和减小文档索引颗粒度来优化索引结构，其对问题输入进行预处理以优化了检索精度。在提取到对应文档后，通过后处理进行数据重排并改善 Prompt 结构。

此外，有研究尝试针对文档类型进行改进，使 RAG 架构支持更多类型的文

档以存储多模态数据。有研究尝试对问题输入进行更加准确的处理，对提取到的实体赋权重，以支持更长的用户问题输入。有研究尝试优化在生成时考虑联网搜索外部数据，增加答案的丰富性。

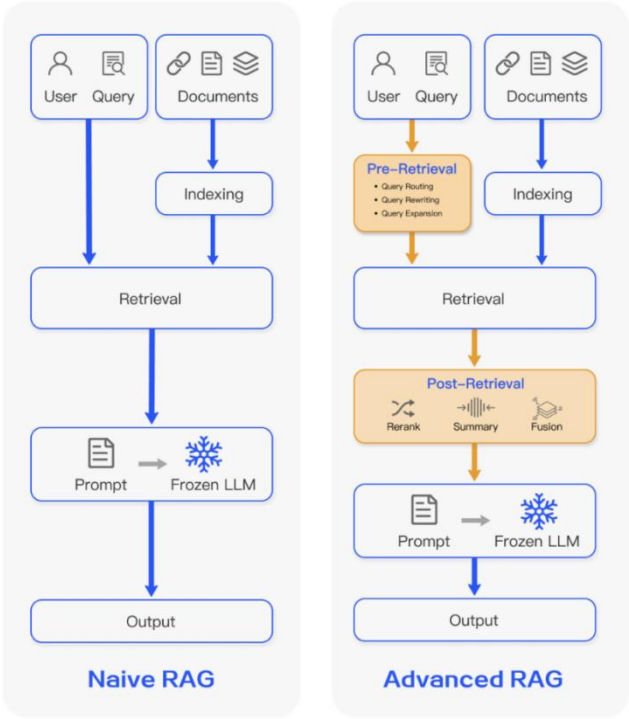


图 2.2 初级RAG和进阶RAG结构

针对大语言模型的调用和应用构建，LangChain 提供了一个基于大语言模型的上层应用开源开发框架。其包含一系列的接口和工具，便于开发者构建和部署基于大语言模型的应用，其基本功能如图 2.3 所示。LangChain 将与大语言模型有关的组件链接在一起，实现了大语言模型接口的统一和标准化。

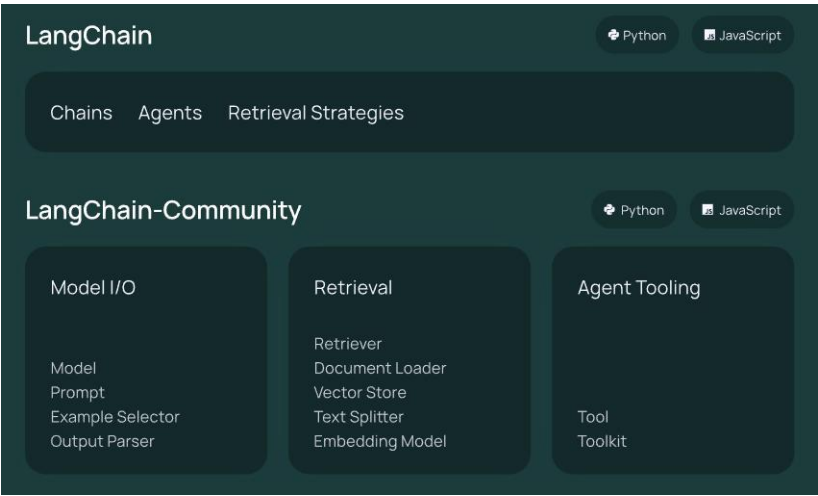


图 2.3 LangChain基本框架图

3 系统框架设计

3.1 总体技术流程

本作品基于 RAG 架构和 LangChain 框架进行优化改善，通过文档获取与解析、文档索引与存储、文档检索和文档问答等步骤实现了一个智能文档问答机器人。项目的技术路线框架图如图 3.1 所示

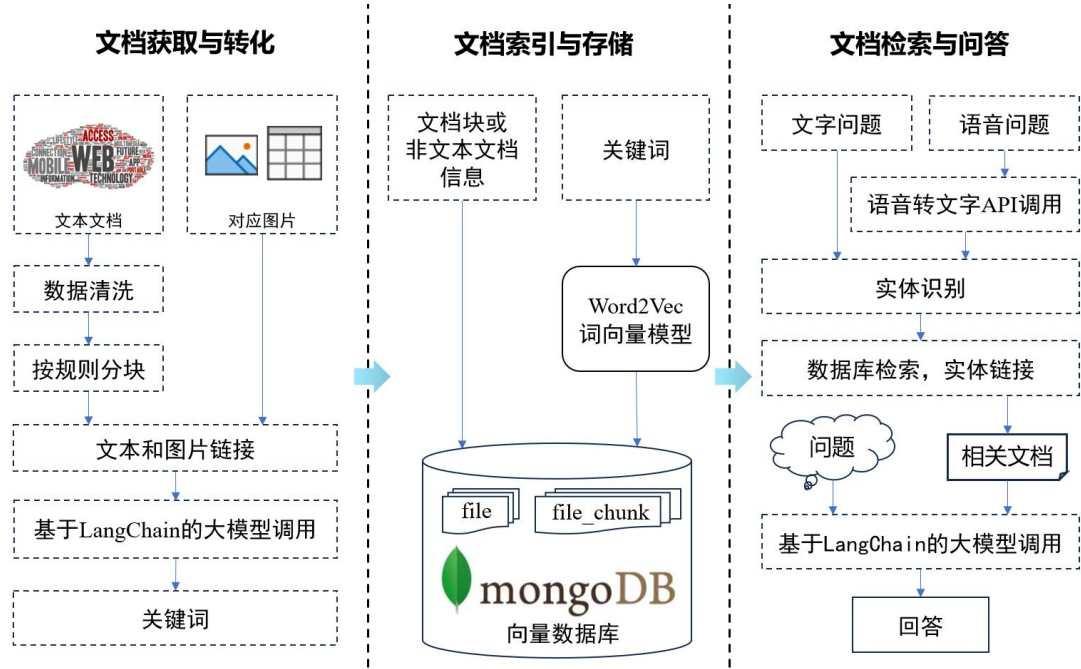


图 3.1 技术路线框架图

团队本地部署大模型并微调，针对 Deepin 系统相关的中文教程和词条进行优化。

本项目支持在私有知识库中存储多模态文档，包括文本文档和图片。团队围绕 Deepin Wiki 开源文档库，收集上述文档并进行解析与存储。对于文本文档而言，对其进行预处理和分块，并对每一个文本块提取关键词。文本文档包含以链接形式内嵌的图片，使用多模态大模型对一个文本块中的文本和图片一并提取关键词。文档问答机器人支持通过文字和语音形式进行提问，问题中的实体被识别后，与数据库中关键词进行实体链接，检索相关文档。基于 LangChain 架构，相关文档内容与问题一同构建提示输入大模型，最终输出回答。

3.2 大语言模型的本地部署与训练

3.2.1 模型选择与本地部署

为了实现大语言模型的本地部署，需要选取合适的开源大语言模型。目前，主流的开源大语言模型有 ChatGLM 和 LLaMA 等，ChatGLM 作为一款由清华大学开发的大模型，具有更好的生态支撑。其训练数据包含了比例为 1:1 的中文和英文数据，其更适应于中国的数据环境和文化背景，能够更好地处理中文文本，并能够在后续微调环节中理解 Deepin 系统相关的知识。

因此，本作品选用 ChatGLM3-6b 版本大模型进行本地部署。该版本的模型部署门槛低、对话流畅且功能支持强大，推理所需显存约为 13G，满足命题要求的 16G 显存条件。同时，该版本模型支持图文理解、文生图等多模态需求，可以帮助处理 Deepin Wiki 文档中涉及的图片本部分。

3.2.2 模型微调

目前，想要提升通用大语言模型对特定领域知识的理解能力，解决大模型“幻觉”问题，主要有两种技术路线：检索增强生成和大模型微调（fine-tuning）。在前文中已经介绍了本项目检索增强生成的实现方案，接下来将介绍微调方法以及微调数据集的制作。

大模型微调是利用特定领域的数据集对已预训练的大模型进行进一步训练的过程，以优化模型在特定任务上的性能。本项目要求大语言模型能够解决的问题范围十分明确，即 Deepin 社区文档相关问答。因此团队基于 Deepin 社区文档，用大模型（文心一言）处理相应的语料生成对应的微调数据集，并选择 ChatGLM3-6B 作为基础大模型进行微调。ChatGLM3-6b 模型支持 SFT 全量微调、P-TuningV2 微调和 LORA 微调等微调方式。针对项目特点，使用 P-Tuning V2 进行低参微调。P-Tuning V2 是一种高效的参数化提示微调方法，它在多个模型层都加入相互独立的连续提示，通过增加可训练参数来引导模型生成符合特定领域风格的输出。

对于大语言模型的微调，数据集是关键。用于微调数据集的质量和数量将决定微调能否取得明显的成果。本项目的微调过程中规定数据集的格式如下：

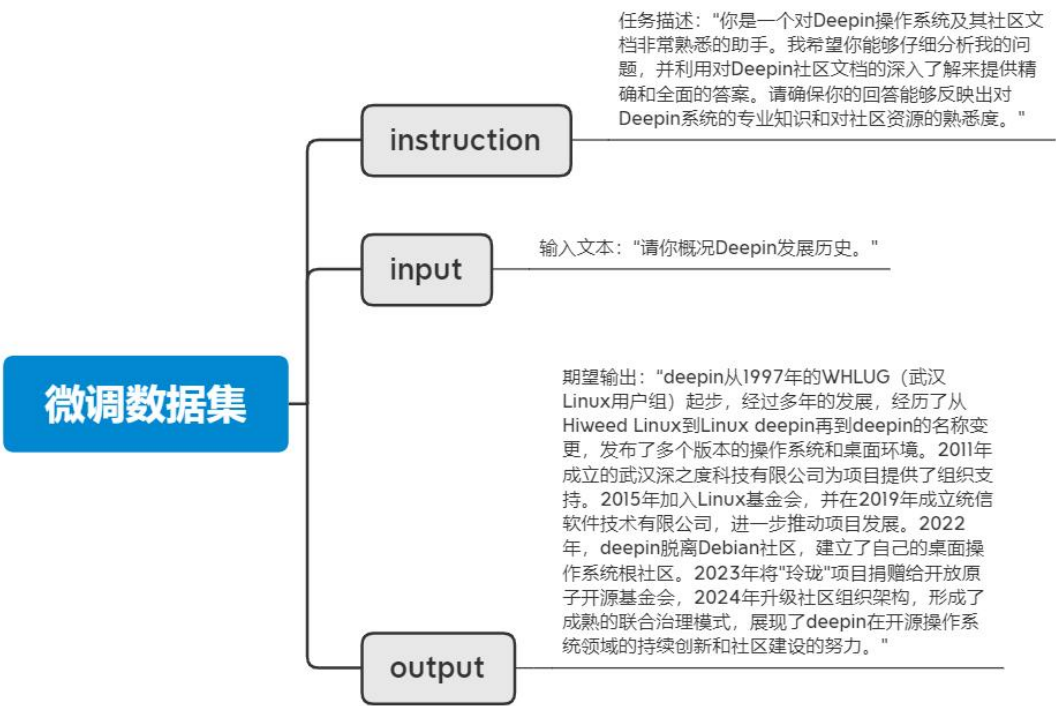


图 3.2 微调数据集格式

项目成员将从 Deepin Wiki 仓库的得到的 Markdown 格式数据进行解析，依据其自带的标题进行章节的划分，然后使用文心一言大模型从每个章节中提取出相关问题以及问题的精准答案，形成如上图所示的数据集条目（例子来源于社区简介章节 1.2 社区历史）。最后经过人工筛选得到高质量的数据集条目，使用 PEFT 技术在 ChatGLM3-6B 的基础上进行参数高效微调。

3.3 文档获取与存储

3.3.1 文档语料收集

语料可以理解为语言材料，包括口语材料和书面材料。语料的定义较为广泛，其来源可能是新闻报刊、纸质书籍、社交平台推文、微信公众号推文、权威百科和开源数据库等，涉及学科也较为复杂。可通过收集特定主题的语料，训练模型和获取期望结果。

本作品中的涉及的语料聚焦 Deepin 系统，通过题目所给网址 <https://github.com/linuxdeepin/wiki.deepin.org> 获得所需文档。

对于文档语料而言，我们通过预处理和解析将其转化，并将结果存储到数据库中，帮助大语言模型回答基于 Deepin 系统知识的问题。

从 github 仓库中下载下来的文档格式为 markdown 文件，我们使用 markdown 库来解析 Markdown 文件并将其转换为 txt 格式文档，方便后续操作。

由于不同的数据源和数据格式，原始文本中可能存在着各种特殊字符、标点符号、数字、链接、HTML 标签等杂乱的因素，这些因素会对文档问答的效果产生负面影响。因此，获得文档数据后，先进行文本数据的预处理工作，以确保文本数据的纯净性和一致性，提高问答效果的准确性、流畅度和自然度。预处理工作包括使用正则表达式进行文本的清洗和去除停用词。正则表达式是一种用来匹配和处理文本的强大工具，可以去除 html 标签、特殊符号和干扰数据。它通过描述字符组成和字符之间的关系，识别、提取和替换文本中的特定模式。在文本清理过程中，正则表达式可以帮助我们高效地处理各种文本中存在的问题，实现自动化的清理过程。除此之外，还使用哈工大的停用词列表去除文本中的常见停用词，如“的”、“是”、“了”等，这样可以减少不必要的噪声和冗余信息，使得文档语料更加紧凑和有效。值得一提的是，使用正则表达式去除的标签不包括图片链接标签，因为图片链接标签在后续步骤中会进行使用。

处理后的文档语料以文本文件的形式存储在服务器中。文件系统中建立了合适的目录结构和索引，以便于后续的检索和访问。

3.3.2 文档解析与转换

进行文档解析与转换的主要目的是提高文档处理的效率和准确性，其主要流程如下：

将文档按照一定的字数进行分块，并在块与块之间设置一定字数的重叠。对于每一个文本块使用大模型提取关键词。由于 Deepin Wiki 文档内嵌图片，为了将这部分图片的语义信息纳入考虑范围，需要对每一个文本块进行扫描。每当遇到图片链接标签（），便从对应路径加载图片一并提供给多模态大模型提取关键词。将提取的关键词转化为词向量。上述过程如图 3.2 所示，其可以有效提高文档检索准确性、提升用户体验。

首先，为了避免文档过长带来的索引关键词覆盖不足的问题、提升文档检索的准确性，需要将文档按一定字数进行分块，并在块与块之间设置一定字数的重

叠，防止关键信息被分裂到两个不同的块中。相较于一些 RAG 架构在检索到文本块后扩充对应文本块的上下界，这种方式能够在保证充分上下文信息的同时，获得更加精准的关键词，进而提升检索时的精度和召回率。

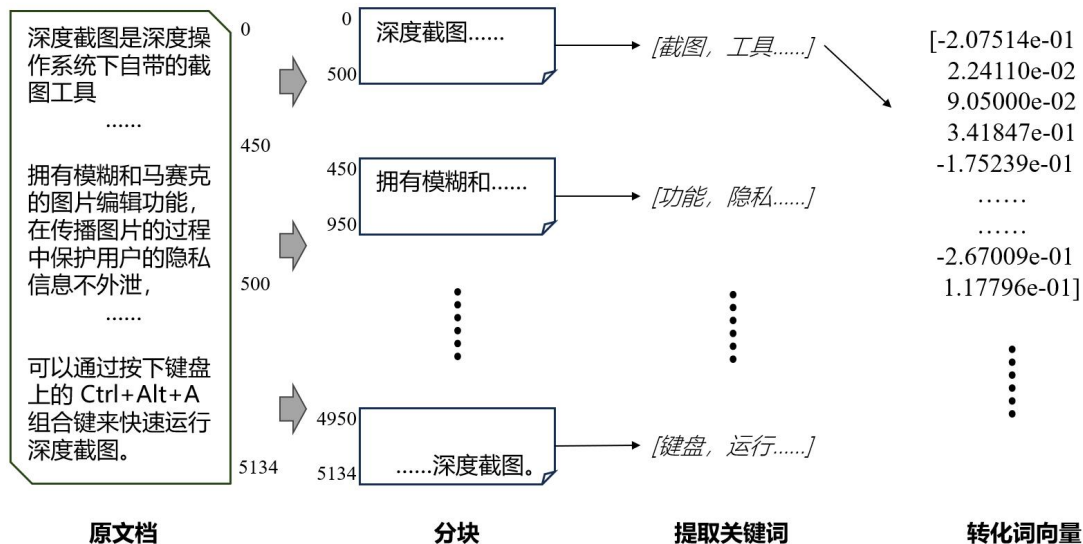


图 3.3 文档分块和词向量提取

考虑到 Deepin Wiki 文档的组织结构，其包含以链接形式内嵌在 Wiki 文档中的图片，这些图片与上下文文本有较强的语义关联。因此，选择使用多模态大模型对一个文本块中的文本和图片一并提取关键词。在给大模型提供文档内容时，检索每一个文档块中的图片链接标签（）并根据路径加载图片构建提示词 Prompt 提供给大模型，以生成对应的关键词。

LangChain 框架是一种利用大语言模型的能力开发各种下游应用的开源框架，为各种大语言模型应用提供通用接口。我们使用 LangChain 框架提供的接口添加 Prompt 指令，要求大语言模型针对输入的内容抽取 5 个长度不超过 3 的关键词。大语言模型能够对文档块的多模态内容进行解析和分析，提取出关键词信息。

然后，调用 Word2Vec 词向量模型将抽取出的关键词转化为词向量，以便后续的文本分析和处理。Word2Vec 是一种将单词嵌入为向量的词向量模型，可以将单词转化为具有语义信息的向量表示，从而提高文本处理的效果。

在上述过程中，为了避免提取到的关键词未在词向量模型中注册，导致关键词无效的问题，需要针对提取的关键词进行检测。若关键词未包含在词向量模型中，则按置信度顺位向后取一个关键词，直到取到五个均在词向量模型中注册的关键词。该操作可以保证提取到的关键词都具有一定的语义信息，能够提高检索

的精度和效率。

3.3.3 数据库设计和文档存储索引

传统的 LangChain 架构将文本内容和索引一同存储在数据库，使文档只需被加载一次，加快获取相关文档的效率，但却导致了数据库的数据量剧增，存储数据冗余，数据库操作的性能也因此受到限制。

本作品摒弃了上述弊端，设计了轻量化的数据库结构，对于文本块仅存储其开始和结束位置，在查询到对应文本块时再读取其内容。

具体而言，采用 MongoDB 文档数据库存储文档基本信息和对应的词向量内容。在数据库中建立 `file` 和 `file_chunk` 两张表，两张表存储的数据信息如表 3.1 所示。`file` 表存储文档信息，其字段信息如表 3.2 所示；`file_chunk` 表存储文本块和词向量等信息，其字段信息如表 3.3 所示。`file_chunk` 表包含文档标识符字段，存储该文本块对应的文档唯一标识，进而实现两表之间的关联。每一个文档都包含一个 `file` 表记录 and 若干个 `file_chunk` 表记录。

表 3.1 数据库表信息

| 表名 | 存储数据 |
|-------------------------|---------------|
| <code>file</code> | 文档基本信息 |
| <code>file_chunk</code> | 文本块信息、关键词和词向量 |

表 3.2 `file`表字段信息

| 字段名 | 数据类型 | 描述 |
|------------------------|-----------------------|--------------|
| <code>_id</code> | <code>ObjectId</code> | 系统自动生成的唯一标识符 |
| <code>file_path</code> | <code>string</code> | 文件相对路径 |
| <code>file_name</code> | <code>string</code> | 文件名称 |

表 3.3 `file_chunk`表字段信息

| 字段名 | 数据类型 | 描述 |
|----------------------|-----------------------|--------------|
| <code>_id</code> | <code>ObjectId</code> | 系统自动生成的唯一标识符 |
| <code>file_id</code> | <code>ObjectId</code> | 对应文档的唯一标识符 |

| | | |
|------------------|-------------|---------------|
| paragraph_number | int | 该文本块在文档中的序号 |
| start_position | int | 该文本块在文档中的起始位置 |
| end_position | int | 该文本块在文档中的结束位置 |
| keywords | string[5] | 对该文本块提取的关键词 |
| word_vectors | int[5][300] | 关键词转化生成的词向量 |

读取到文件后，将其基本信息存储在 `file` 表中并获取该条记录对应的 `_id` 值。对于一个文本文档中的每个文本块，创建一条 `file_chunk` 表记录，将文本块的信息和关键词词向量等数据保存在该记录中。将该记录的 `file_id` 字段设置为对应文档在 `file` 表中的 `_id` 字段值，从而将文档和对应的文本块关联。每一个文本文档包含一个 `file` 表记录和与其文本块数量相等的 `file_chunk` 表记录。

3.4 文档检索与问答实现

3.4.1 语音问题输入

用户通过提出问题与文档问答机器人进行交互。文档问答机器人支持用户通过文字或语音的方式进行提问。对于语音提问，需要将其转化成文字形式的问题，从而进行下一步处理，该过程具体流程如图 3.3 所示。

在用户发出请求后，录音开始，系统通过 `PyAudio` 库来捕获音频数据。需要设置一定的录音取样频率，并定义录音的最小长度和声音保存的阈值。当录音过程中检测到一定数量的高于阈值的样本时，它将开始保存录音。当用户点击结束或声音连续一段时间低于阈值后，录音结束。录音完成后，录音数据将被保存为一个 `WAV` 文件并发送至后端服务器。

为了便于后续的 API 调用，需要对录音文件进行一系列的预处理操作。后端服务器接收到该 `WAV` 文件后，计算 `WAV` 文件的字节大小。接着使用 `base64` 库将其转换为 `Base64` 编码，并将编码后的内容转换为 `UTF-8` 格式的字符串。

为了实现语音转文字，需要将 `Base64` 编码的音频数据发送到百度智能云千帆平台的 API。该 API 要求构建一个 `JSON` 格式的有效载荷 (`payload`)，其中包含了音频的编码格式、采样率、声道数、客户端 ID、令牌以及 `Base64` 编码的语音数据和其长度。通过 `HTTP` 协议的 `POST` 请求将构建的 `payload` 发送到语音识

别服务的 URL。语音识别服务将会处理发送的音频数据，并以 JSON 格式返回识别后的文本结果。后端系统提取其中的问题文本进行下一步处理。

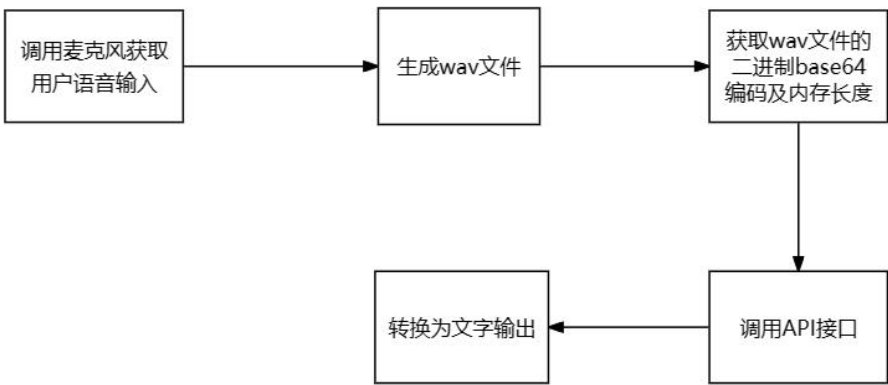


图 3.4 语音转文字实现流程

3.4.2 文档检索

基于上述步骤获得的文本形式的问题，对其进行实体识别。调用词向量模型，将问题中的有效实体转化为词向量，并与 file_chunk 表中每条记录的关键词词向量交叉计算余弦相似度，实现实体链接。

在匹配计算时，存在一个问题的实体列表和若干个关键词词向量列表。遍历每一个关键词词向量列表，与实体列表进行匹配计算。针对这两个列表中的每一个元素，两两计算余弦相似度，选取相似度最高的两个结果取平均值，作为该文本块记录与用户问题的关联度，如图 3.4 所示。

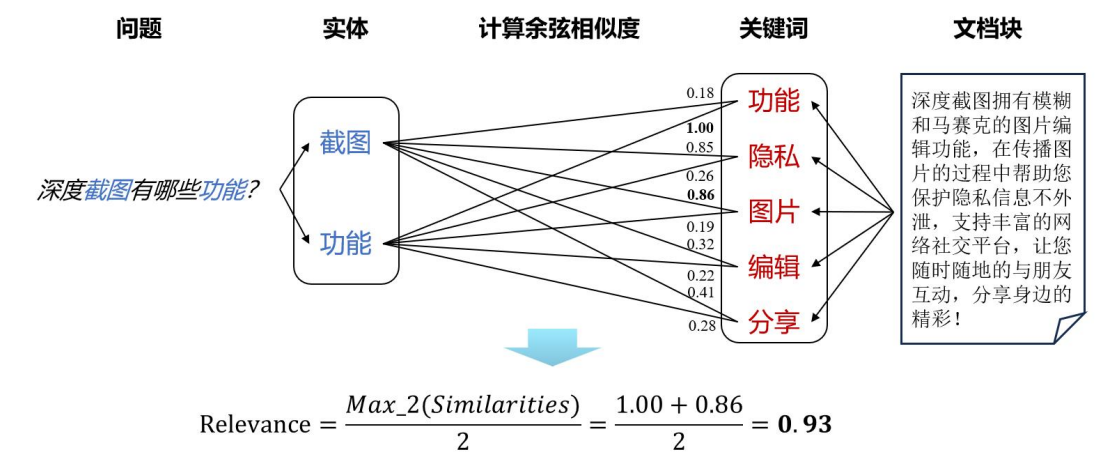


图 3.5 对每个文档块进行关联度计算

余弦相似度计算公式如下：

$$Sim = \frac{V_{w1} \cdot V_{w2}}{\|V_{w1}\| \|V_{w2}\|}$$

其中 V_{w1}, V_{w2} 分别是两个词语的词向量， Sim 值范围为 $[-1,1]$ ，越接近 1 则说明两个词语的语义相似度越高。

对于 `file_chunk` 表中的每一条记录，都通过此方式计算其与问题的关联度，最终寻找两个关联度最高的记录，作为与该问题相关的文本块。该算法综合考虑文本块的关联度排名和大语言模型 `token` 大小限制，有效提升了相关文本块的召回率和准确性。

3.4.3 提示词构建和答案生成

本作品中，提示词的构建和大语言模型（ChatGLM3-6b）的应用通过调用 `LangChain` 接口实现。

提示词包含了用户问题、相关文档和其它信息。当系统根据用户问题输入检索到相关的文档后，需要访问对应文档并获取具体内容，将这些内容作为信息集合。基于此，系统会构建一个包含信息集合的提示词，用于引导大语言模型针对用户提出的问题进行回答。提示词的结构主要通过设定大语言模型的“能力与角色”及“指令”实现，具体内容如图 3.5 所示。

在能力与角色模块，通过提示词将大语言模型设定为一个数据分析师，通过检索用户提供的信息集合中的相关信息以给出用户对应问题的答案。

在指令部分，提示词包含的指令则要求大语言模型完全依赖用户提供的信息来回答用户问题，并要求其在发现用户提供的信息不足时回复：“信息不足”。这样的指令有效减少了大语言模型幻觉的产生，防止其在信息不足时生成错误回答。

将上述步骤构建的提示词输入本地部署的 ChatGLM3-6b 模型中，大语言模型会结合用户问题和相关文档进行回答。如图 3.6 所示，大语言模型将会在信息集合中检索相关内容，然后生成答案反馈给用户。若系统无法从信息集合中找到相关信息，则给出“信息不足”的拒绝回答反馈。

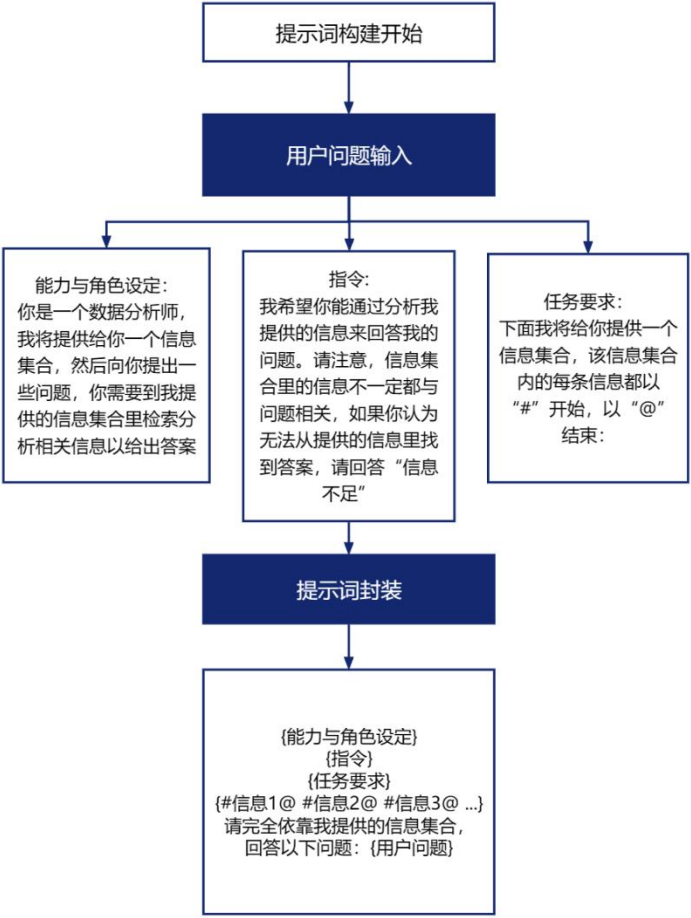


图 3.5 提示词构建

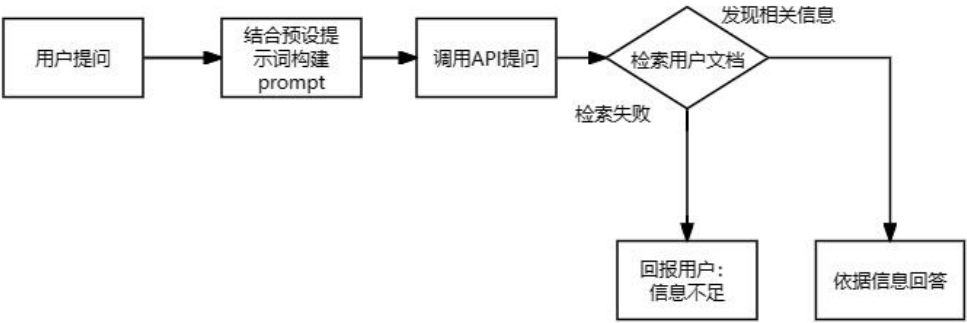


图 3.7 问答实现流程图

3.5 用户界面设计和系统部署

本作品使用前后端工具栈提供与智能文档问答机器人对话的网页服务, 如图 3.7 所示。



图 3.8 前后端技术栈

前端界面通过 Vue3 框架搭建，涉及的组件有：问答显示界面、提问文本框、提问按钮和语音提问按钮。对应的响应事件如表 3.4 所示。

表 3.4 前端组件响应事件

| 组件 | 事件 | 响应函数 |
|--------|----|---------------------------------|
| 提问按钮 | 点击 | 将提问文本框中的文本发送至后端服务器 |
| 语音提问按钮 | 点击 | 开始录音，录音结束后将音频文件以 WAV 格式发送至后端服务器 |

在华为云购买弹性云服务器 ECS 并绑定公网 IP，通过 SSH 协议访问远程服务器并将前端页面相关文件上传至服务器中，通过 Vite 构建工具启动程序，提供公网访问服务。

后续我们将使用 DTK 开发一个用户友好的图形界面，用于用户提问和显示回答。使用 DTK 的 Widget 库和控件库，设计应用程序的用户界面。DTK 扩展了 Qt 的控件库，提供了丰富的 UI 组件和自定义控件。可以在 Qt Designer 中设计界面，拖放控件来构建 UI。在 Deepin 操作系统中使用 DTK 开发一个功能丰富、跨平台兼容的应用程序。DTK 提供了强大的开发工具和丰富的 API，可以帮助我们高效地实现应用需求。

4 系统实现

4.1 文档获取

4.1.1 文本语料获取

为获取 Deepin 系统知识语料，团队选择了相关网站获取文本语料以获取文本文档，如图 4.1 所示。语料来源包括 [linuxdeepin/wiki.deepin.org: wiki content \(github.com\)](https://linuxdeepin/wiki.deepin.org/wiki/content)、[Deepin Wiki - 深度百科 | DeepinWiki](#) 等。同时，团队也获取了部分对应的图片资源，如图 4.2 所示。



图 4.1 文本文档样例

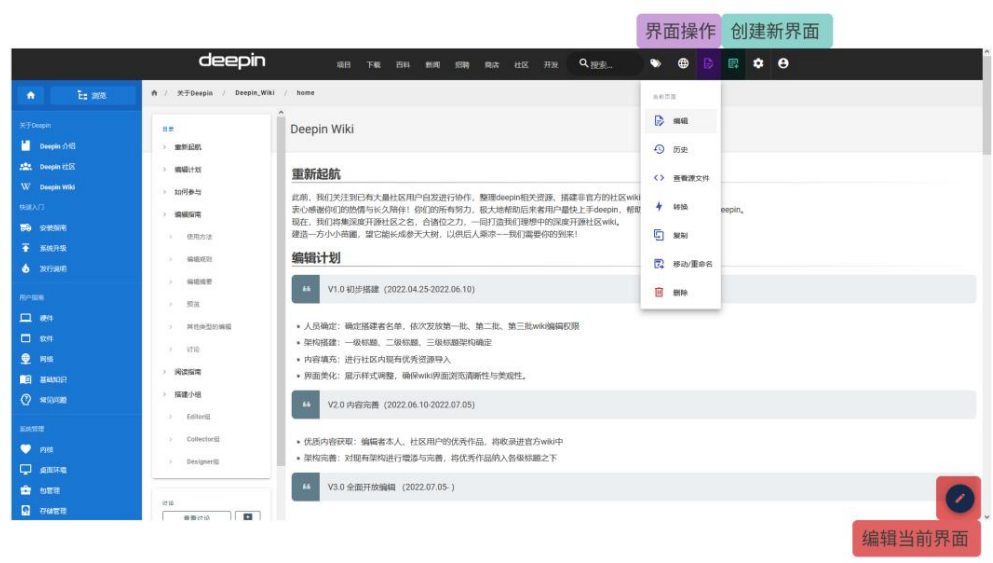


图 4.2 图片样例

4.1.2 预处理

进行关键词提取前，需要文件进行预处理工作。将 markdown 文件转换为 html 格式的文件，对其 txt 文件后，对其中的文本数据进行处理，以确保其语料质量，提高关键词提取的准确性。预处理工作包括使用正则表达式进行文本的清洗和去除停用词。

使用正则表达式进行文本清洗，去除中文、英文、数字和标点以外的所有字符。同时利用正则表达式进行机械压缩，去除连续的重复内容。

使用哈工大的停用词列表去除文本中的常见停用词，如“的”、“是”、“了”等，该停用词表的部分内容如图 4.3 所示。对文本内容进行遍历，如果遇到停用词表中的内容，则对其进行去除，进而提升文本语料质量。

如果文档中插入了图片文件，提取出图片引用，并补充完整 url，方便后续调用。

| | |
|----|------|
| 阿 | 除此之外 |
| 哎 | 除非 |
| 哎呀 | 除了 |
| 哎哟 | 此 |
| 唉 | 此间 |
| 俺 | 此外 |
| 俺们 | 从 |
| 按 | 从而 |
| 按照 | 打 |
| 吧 | 待 |
| 吧哒 | 但 |
| 把 | 但是 |
| 罢了 | 当 |

图 4.3 停用词列表示例

4.2 ChatGLM3-6b 模型部署和微调

本地部署并微调 ChatGLM3-6b 模型需要至清华大学对应的开源仓库下载仓库内容和模型文件，进而建立本地仓库。

分别执行：

```
git clone https://github.com/THUDM/ChatGLM3
git clone https://huggingface.co/THUDM/chatglm3-6b
```

```
pip install -r requirements.txt
```

下载资源创建本地仓库，并建立所需环境。

基于上一步收集到的文档语料，进行人工标注以生成微调所用的训练集和验证集，数据样例如图 4.4 所示。

```
1 {"prompt": "功能#图片编辑*编辑方法#模糊*编辑方法#马赛克",  
2  "response": "深度截图拥有模糊和马赛克的图片编辑功能。"}]
```

图 4.4 微调数据集样例

接着使用仓库中 P-Tuning V2 微调的配置文件，运行低参微调程序，并在验证集上进行验证。

4.3 文档解析与索引存储

在后端运行上传文档的程序后，系统检索对应目录下所有支持的文件，并对每个文件按照其相对路径构建查询条件，并获取更新内容，如图 4.5 所示。使用 update 更新操作访问数据库的 file 表，如果该文件已经存在数据库中，则更新其字段值；如果还未存在数据库中，则新建该记录。存储该记录的_id 字段值，供后续创建 file_chunk 表记录时使用。

```
# 构建查询条件  
file_filter_query = {"file_path": file_rel_path}  
  
# 构建更新记录  
file_update_values = {  
    "$set": {  
        "file_path": file_rel_path,  
        "file_name": os.path.basename(file_abs_path),  
        "type": file_type  
    }  
}
```

图 4.5 file表查询条件和更新记录内容

对于文本文档，按照 500 字的长度进行分块处理，段与段之间引入 50 字的重叠。分段处理文章，对于每一个文本块，根据图片路径标签加载对应的图片，调用本地部署的 ChatGLM3-6b 大模型对包含图片的文本内容提取关键词。使用 LangChain 框架提供的接口添加 Prompt 指令，要求大语言模型针对输入的内容抽取 5 个长度不超过 3 的关键词，如图 4.6 所示。

```
prompt_string = """请对上传的图像或文本提取 \
5个最佳关键词 \
请务必确保每个关键词的长度不超过3个字
"""
```

图 4.6 提取关键词的提示语

对每个文本块或文档获取到 5 个关键词后，使用开源的中文词向量模型 Chinese-Word-Vectors 将关键词转化为词向量，该词向量模型基于百度百科等综合语料进行训练。此处，由于生成的词向量格式不能直接存储至 MongoDB 数据库中，需要将其转化为列表形式再进行存储，如图 4.7 所示。

```
# 词向量转化为列表存储
standard_vectors = []
for word_vector in word_vectors:
    standard_vectors.append(np.array(np.ravel(word_vector)).astype(float).tolist())
```

图 4.7 将词向量转化为列表形式

对每一个文本块，按照其对应的文档_id 和段落号构建查询条件，并获取更新内容，如图 4.8 所示。使用 update 更新操作访问数据库的 file_chunk 表，如果该文本块已经存在数据库中，则更新其字段值；如果还未存在数据库中，则新建该记录。

```
# 构建查询条件
chunk_filter_query = {
    "text_ID": article_id,
    "paragraph_number": paragraph_number
}

# 构建更新记录
chunk_update_values = {
    "$set": {
        "text_ID": article_id,
        "paragraph_number": paragraph_number,
        "start_position": start,
        "end_position": end,
        "keywords": keywords,
        "word_vectors": standard_vectors
    }
}
```

图 4.8 file_chunk表查询条件和更新记录内容

4.4 用户问题处理与文档检索

4.4.1 处理语音问题输入

在录音阶段，使用 `get_voice.py` 脚本来捕获用户的语音。这个脚本利用了 PyAudio 库。PyAudio 库是一个跨平台的音频输入输出库，它允许 Python 代码与音频硬件进行交互。该部分包含一个 `Recorder` 类，负责管理录音的整个过程。

在 `Recorder` 类中，需要设置缓冲区大小等关键参数，如图 4.9 所示。

使用 PyAudio 对象打开音频流，配置为单声道输入。在一个循环中，持续从音频流中读取数据。每一缓冲区的数据都会被读取并转换为 NumPy 数组，以便于后续处理。通过比较音频样本的振幅与预设的阈值 `LEVEL`，确定是否有足够强的声音信号。如果在 `NUM_SAMPLES` 个样本中检测到超过 `COUNT_NUM` 个样本振幅超过阈值，则认为检测到了声音。

```
class Recorder:
    NUM_SAMPLES = 2000 # pyaudio内置缓冲大小
    SAMPLING_RATE = 8000 # 取样频率
    LEVEL = 500 # 声音保存的阈值
    COUNT_NUM = 20 # NUM_SAMPLES个取样之内出现COUNT_NUM个大于LEVEL的取样则记录声音
    SAVE_LENGTH = 8 # 声音记录的最小长度：SAVE_LENGTH * NUM_SAMPLES 个取样
    TIME_COUNT = 60 # 录音时间，单位s
```

图 4.9 Recorder类参数设置

当检测到声音时，开始保存音频数据。保存的音频会累积到预设的长度：`SAVE_LENGTH * NUM_SAMPLES`。当达到录音时间限制 `TIME_COUNT`、没有声音信号或用户主动结束时，结束录音过程。

录音完成后，使用 `transfer_base64.py` 脚本将 WAV 文件转换为 Base64 编码，以便网络传输。以二进制读取模式打开 WAV 文件，并读取其内容。使用 `base64` 库将读取的音频数据进行 Base64 编码。同时记录 WAV 文件的字节大小。

最后，使用 `voice_identify.py` 脚本将 Base64 编码的音频数据发送到百度智能云千帆平台的语音识别服务 API，以获取识别后的文本。构建一个 JSON 格式的有效载荷 `payload`，包括音频文件的编码格式、采样率、声道数、客户端 ID、令牌、Base64 编码的语音数据和其长度，如图 4.10 所示。设置 HTTP 请求头，指定内容类型和接受类型均为 JSON。使用 `requests` 库发送一个 POST 请求到语音

识别服务的 URL，传递构建好的 payload。

```
payload = json.dumps({
    "format": "wav",
    "rate": 16000,
    "channel": 1,
    "cuid": "X8EvHMIZZXtX1myXDqXFTHfrvx6UTWf0",
    "token": "24.ca43c9e9a16cfeea0f339190aff16560.2592000.1715514542.282335-61069881",
    "speech": changeTo64base.base64_output,
    "len": changeTo64base.file_size
})
```

图 4.10 有效载荷内容

收到服务器响应后，将 JSON 格式的响应内容转化为文本形式的问题。

4.4.2 问题实体识别

为了进行文档检索，需要对文本形式的问题进行实体识别。包括预处理、分词、词性标注和关键词提取几个步骤。其中，预处理与 4.1.2 节中对文本语料的预处理方式一致，在此不再赘述。

利用分词工具如 jieba 对预处理后的文本进行分词，将文本分割成有意义的词语。利用 jieba 自带的词性标注工具对分词后的词语进行标注，标注出每个词语的词性，如 a（形容词）、n（名词）、v（动词）等，如图 4.11 所示。在本作品中，根据词性标注结果，保留了名词和动词作为实体识别结果。

ICTCLAS 汉语词性标注集

| 代码 | 名称 | 帮助记忆的诠释 |
|----|------|-------------------------------|
| Ag | 形语素 | 形容词性语素。形容词代码为a，语素代码 g 前面置以A。 |
| a | 形容词 | 取英语形容词adjective的第1个字母。 |
| ad | 副形词 | 直接作状语的形容词。形容词代码a和副词代码d并在一起。 |
| an | 名形词 | 具有名词功能的形容词。形容词代码a和名词代码n并在一起。 |
| b | 区别词 | 取汉字“别”的声母。 |
| c | 连词 | 取英语连词conjunction的第1个字母。 |
| Dg | 副语素 | 副词性语素。副词代码为d，语素代码 g 前面置以D。 |
| d | 副词 | 取adverb的第2个字母，因其第1个字母已用于形容词。 |
| e | 叹词 | 取英语叹词exclamation的第1个字母。 |
| f | 方位词 | 取汉字“方”的声母。 |
| g | 语素 | 绝大多数语素都能作为合成词的“词根”，取汉字“根”的声母。 |
| h | 前接成分 | 取英语head的第1个字母。 |

图 4.11 汉语词性标注集

4.4.3 实体链接与文档检索

调用词向量模型，将上一步中提取到的用户问题中的实体转化为一个问题词向量集合，遍历数据库 `file_chunk` 表中的每一条记录，其 5 个关键词的词向量构成一个记录词向量集合。

对问题词向量集合与每一个记录词向量集合做余弦相似度计算，选取最大的两个结果取平均值，作为该记录与问题的关联程度。

遍历过程中，保存两个关联度最高的记录，作为与该问题相关的文本块。遍历结束后，检查两个记录的关联度是否大于阈值，如图 4.12 所示。将大于阈值的记录进行下一步处理，如果两个记录都不大于阈值，则返回空，提示未找到相关文档。经过实验修正，我们将阈值大小设置为 0.4。

```
# 检查记录的相似度是否达到阈值
linked_entities = [(record, similarity) for record, similarity in relative_records if similarity >= threshold]

if linked_entities:
    return linked_entities
else:
    return None # 如果相似度均低于阈值，则返回空，提示未找到对应文档
```

图 4.12 检查记录是否达到阈值

4.5 大语言模型调用与答案生成

在进行提示词构建时，将大语言模型设定为一个数据分析师，使其依据提供的信息集合，给出用户问题的答案。向提示词中添加指令，要求大语言模型完全依赖用户提供的信息集合来回答问题，并要求其在发现信息不足以回答问题时回复：“信息不足”。如图 4.13 所示，提示词中指明：对于信息集合中的每一条信息，均以“#”开头，以“@”结束，进而提供清晰的信息边界。在实际开发中，提示词构建通过调用 `LangChain` 框架提供的接口实现。

```
for text in texts:
    prompt += "#" + text + "@"
prompt += "请完全依靠我提供的信息集合，回答我以下问题：" + question
```

图 4.13 提示词的部分构建流程

构建信息集合时，每当遇到图片链接标签，同样需要将对应的图片加载至信

息集合中，供大模型参考。

对应的基于构建的完整提示词，调用本地部署的 ChatGLM3-6b 模型进行问答。如图 4.14 所示，在开发相关函数时，包含了历史对话信息 `history` 作为 ChatGLM3-6b 模型的输入和输出，从而使文档问答机器人支持多轮对话。在实际开发中，由于 ChatGLM3-6b 的本地部署提供了十分简洁的调用接口，因此选择直接加载调用，而非使用 LangChain 框架。

```
def glm_caller(prompt, history):  
    response, history = model.chat(tokenizer, prompt, history=history)  
  
    return response, history
```

图 4.14 ChatGLM3-6b模型调用

4.6 系统搭建和部署

4.6.1 前端设计和开发

本作品的前端页面由问答显示界面、提问文本框、提问按钮和语音提问按钮等部分组成。整体页面设计如图 4.15 所示。



图 4.15 问答机器人整体页面设计

问答显示界面模仿传统的大语言模型在线问答界面，左侧显示 AI 生成的回

答，右侧显示用户输入的问题，如图 4.16 所示。

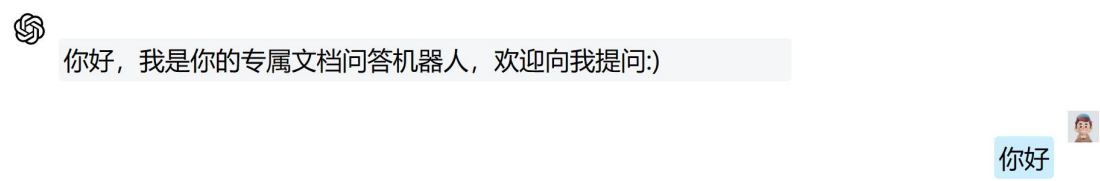


图 4.16 问答显示界面设计

提问文本框支持用户输入不超过 500 个字的问题，如图 4.17 所示。

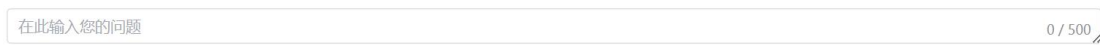


图 4.17 提问文本框设计

提问按钮显示“发送”字样，如图 4.18 所示。用户点击按钮后，提问文本框中的问题将被发送至后端服务器。



图 4.18 提问按钮设计

语音提问按钮显示麦克风的图案，用户点击语音提问按钮后，开始录音，按钮图案将会变化，如图 4.19 所示。待录音结束后，按钮将重新恢复麦克风图案。



图 4.19 点击前后的语音提问按钮设计

4.6.2 系统部署

本作品的前后端均部署于华为云弹性云服务器 ECS。在华为云平台网站购买弹性云服务器，将计费模式选择为按需计费以减小成本，选择 GPU 加速型服务器以提供大语言模型推理的性能，服务器操作系统选择 Ubuntu 22.04，规格选择 32G 内存，16G 显存。

同时购买并绑定公网 IP，选择按流量计费。服务器配置完成后，通过 SSH 协议访问，并将前端页面相关文件上传至服务器中，通过 Vite 构建工具启动程序，提供公网访问服务。文档问答机器人的公网访问在 Google Chrome 浏览器中访问的效果如图 4.20 所示。

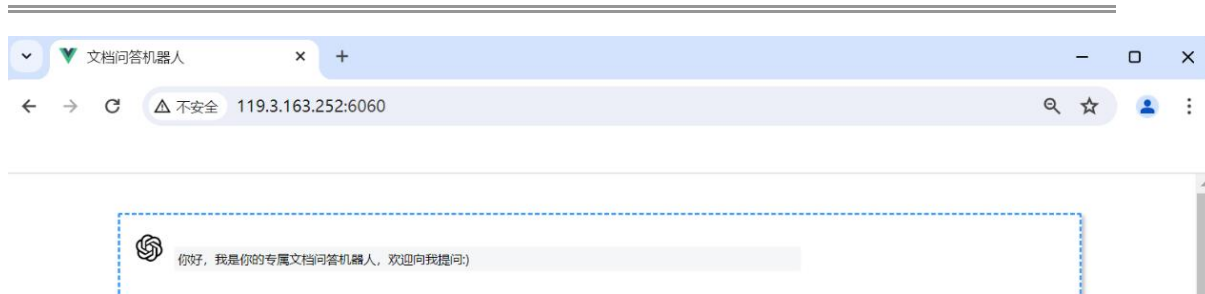


图 4.20 通过Google Chrome浏览器访问本作品网页

4.7 作品功能呈现

4.7.1 文件上传

用户在后端运行上传文档的程序, 系统将会处理目标目录下所有 Wiki 文档。

对于文本文档, 系统进行分块后提取关键词并转化为词向量, 将对应信息存储至 MongoDB 数据库中, 文本文档示例如图 4.21 所示。该示例文本文档有一份内嵌图片“页面截图”, 对应标签为“”, 图片内容如图 4.22 所示。使用大模型对文本和内嵌图片进行关键词提取, 进而存储至数据库中, 对应生成的 file 表记录和其中一条 file_chunk 表记录如图 4.23 和图 4.24 所示。

```
title: 00_wiki来源和共识
description:
published: true
date: 2023-02-24T07:58:57.747Z
tags:
editor: markdown

dateCreated: 2022-10-17T01:20:57.502Z

来源

https://wiki.deepin.org 是 Deepin 社区官方的百科网站, 用于分享 Deepin 与 Linux 相关的知识。

网站搭建框架是由社区初期投票选择出来的 Wiki.js

00_wiki目录下包含了主要的wiki相关内容, 这里不再重复介绍

<img src='./src/2022-10-17_11770.png' alt='页面截图' />
```

图 4.21 文本文档《00_wiki来源和共识》示例

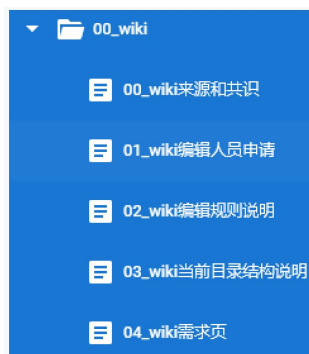


图 4.22 嵌入图片《[页面截图](./src/2022-10-17_11770.png)》示例

```
_id: ObjectId('66537486ad34cc95a6a475ad')
file_path: "../storage/Deepin/txt/00_wiki/00_wiki来源和共识.txt"
file_name: "00_wiki来源和共识.txt"
type: "txt"
```

图 4.23 文本文档生成的file表记录

```
_id: ObjectId('66537486ad34cc95a6a475af')
text_ID: ObjectId('66537486ad34cc95a6a475ad')
paragraph_number: 1
end_position: 451
▼ keywords: Array (5)
  0: "Deepin"
  1: "wiki"
  2: "来源"
  3: "共识"
  4: "社区"
start_position: 0
▼ word_vectors: Array (5)
  ▼ 0: Array (300)
    0: 0.1048860028386116
    1: -0.16104300320148468
    2: -0.1962289959192276
    3: -0.033296000212430954
    4: -0.07669000327587128
    5: 0.22582599520683289
    6: 0.09199599921703339
    7: -0.05930899828672409
    8: 0.15677900612354279
    9: 0.10877999663352966
    10: 0.12018699944019318
    11: 0.022102000191807747
    12: -0.19839200377464294
    13: 0.02530599944293499
    14: 0.10353299975395203
    15: -0.017021000385284424
    16: 0.06138399988412857
    17: 0.2079010009765625
    18: -0.03335399925708771
    19: -0.11051999777555466
    20: 0.0039160000160336494
    21: 0.09592200070619583
    22: 0.055041998624801636
    23: 0.11531999707221985
    24: 0.34915900230407715
```

图 4.24 文本文档生成的其中一条file_chunk表记录

4.7.2 文字问答

通过公网访问搭建的文档问答机器人，可以在线进行基于文档的问答。用户在文本框中输入问题后点击发送按钮，系统将会显示回答，并给出相应回答的参考文件链接，如图 4.25 所示。



图 4.25 系统做出基于文档的回答

如果在提供的信息集合中查询不到对应的信息，系统将会做出信息不足的拒绝回答，如图 4.26 所示。

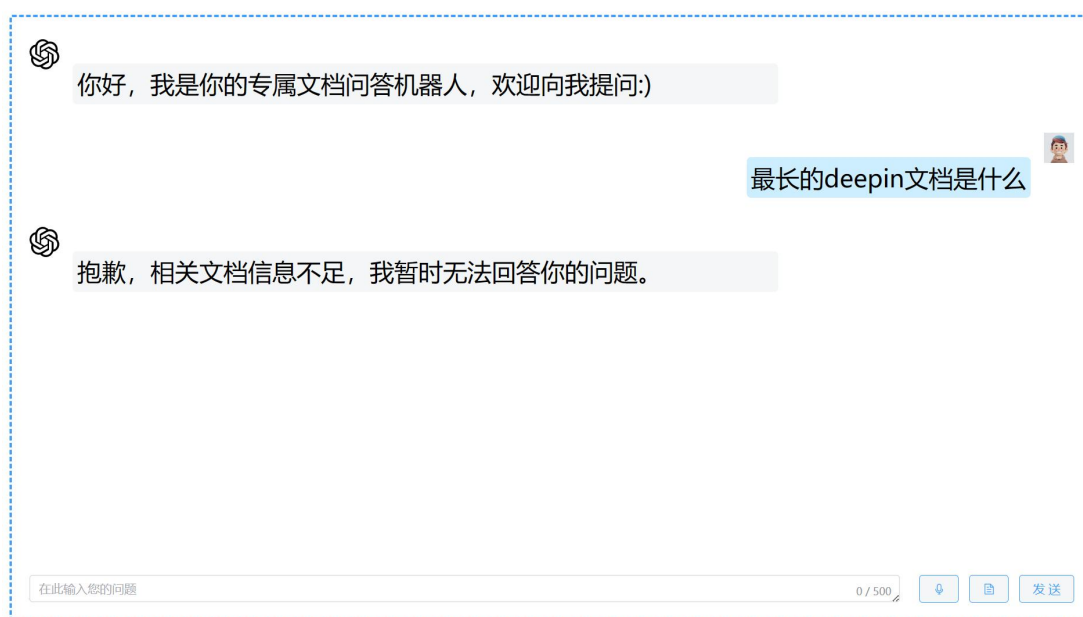


图 4.26 系统做出信息不足的拒绝回答

4.7.3 语音问答

用户点击语音按钮后开始录音，系统在后端调用相关 API 将语音转化为文字，在问题文本框中以打字机的样式提供转换结果反馈，如图 4.27 所示。



图 4.27 系统反馈转换生成的文字

语音提问结束后，系统自动将文本形式的问题发送至后端，并在界面右侧显示为用户问题输入，如图 4.28 所示。



图 4.28 语音输入转化为文本问题

4.8 作品安装说明

4.8.1 系统访问

本作品搭建的智能文档问答机器人以 B/S 架构部署在云端服务器，用户能够通过公网访问问答机器人并与其对话。

4.8.2 本地部署

本作品后端源代码开源于：

<https://g-ngna0222.coding.net/public/oscompetition/DeepinQA/git/files>

用户能够通过命令：`git clone https://e.coding.net/g-ngna0222/oscompetition/DeepinQA.git` 克隆代码仓库并自行在本地部署。如果用户需要进行本地部署，则需要在仓库目录下执行：`pip install -r requirements.txt` 安装必要环境依赖。同时需要自行下载并安装 MongoDB 数据库，再在仓库目录下执行：`python main.py` 运行后端程序。

本作品前端代码开源于

<https://g-ngna0222.coding.net/public/oscompetition/WebQA/git/files>

用户能够通过命令：`git clone https://g-ngna0222.coding.net/public/oscompetition/WebQA.git` 克隆代码仓库并自行在本地部署。如果用户需要进行本地部署，则需要先自行下载并安装 Node.js，再在仓库目录下执行：

```
npm install
```

```
npm run dev
```

运行前端页面。

5 测试分析

5.1 测试概要

5.1.1 测试目的

软件测试的主要目的是找出本作品软件系统与需求规格说明书之间的差异，从而让系统变得更加完善，确保软件在交付给最终用户之前能够满足质量标准和预期需求。针对本作品的系统测试期望发现软件中可能存在的错误和缺陷，以便及时修复和改进。同时期望确保软件的功能和性能在完整性、可靠性和效率等方面均符合需求规格和用户期望。

5.1.2 测试重点

在本作品的测试分析中，我们将重点评估以下方面：

- ①系统稳定性：通过 B/S 架构部署的平台能否通过公网正常访问，在高负荷情况下是否具有稳定性，系统能否在长时间持续稳定提供服务。
- ②问答准确性：对于不同提问类型和不同相关文档的准确回答能力，做出的回答是否能保证以文档内容为参照，问题是否具有客观性和逻辑性。
- ③检索效率：当数据库中包含大量文档时，能否在短时间内找到对应文档并回答用户提出的问题。
- ④用户交互：用户界面设计是否明确相关功能且对用户友好，用户交互是否流畅且符合用户的一般行为逻辑。

5.1.3 测试环境

本测试所使用的环境由表 5.1 所示。

表 5.1 测试环境

| 环境或工具 | 名称与版本 |
|-----------|--------------|
| 操作系统 | Deepin 20.9 |
| Python 环境 | Python 3.8.5 |

| | |
|-------------|-----------------|
| MongoDB 数据库 | MongoDB 7.0.5 |
| NodeJS | Node 16.19.1 |
| 浏览器 | Firefox 120.0.1 |

5.1.4 数据来源与规模

本次测试中，我们语料数据来源包括命题所给网址。数据规模通常根据需求和可获得的资源而定。在本次测试中，我们从 900 余份 Deepin Wiki 文档中选择了 100 份文档并进行存储，其中包括约 34 张图片资源。

5.2 测试结果

5.2.1 文件上传功能测试

文件上传功能测试共测试了爬虫有效性等五个具体项目，测试内容和结果如表 5.2 所示。

表 5.2 文件上传功能测试内容

| 测试项目 | 测试内容 | 测试要求 | 测试方式 | 测试结果 |
|--------------|------------------------------|--|------------------------------------|------------------------------|
| 文本文档分块与关键词提取 | 文本文档能否按照规则分块并链接图片，使用大模型提取关键词 | 对于每一份文本文档，按照规则正确分块，针对每一文本块和图片提取的关键词需充分合理 | 针对收集的文本文档进行分块，对于每一块调用大模型生成关键词并打印输出 | 所有文本文档均正确分块，生成的关键词与文本块内容关系紧密 |
| 关键词有效性 | 提取的关键词能否被有效转化为词向量 | 提取的关键词均需要在词向量模型中注册，如果未注册则按置信度递补下一个关键词，直到关键词均注册 | 针对提取到的关键词检测是否注册，若未注册则调用递补算法 | 最终获得的所有关键词均在词向量模型中注册，关键词有效 |
| 数据库存储 | 数据库能否正常提供新建、更新和查询功能，存储数据是否正确 | 正确存储文档路径、关键词和词向量等数据，能够通过文档路径构建索引，进而 | 针对数据库的两个表分别构建查询条件和更新记录，测试更新和 | 文档信息和关键词正确存储并被更新，通过文档信息正确查询到 |

| | | | |
|--|----------------|------|-------|
| | 实现查询和更新 等操作 | 查询操作 | 对应关键词 |
|--|----------------|------|-------|

5.2.2 文字问答功能测试

文字问答功能测试共测试了实体识别等三个具体项目，测试内容和结果如表 5.3 所示。

表 5.3 文字问答功能测试内容

| 测试项目 | 测试内容 | 测试要求 | 测试方式 | 测试结果 |
|--------------------------------|---------------------------------------|--|--|-------------------------------------|
| 实体识别 | jieba 分词工具能否正确对问题进行分词和词性分析，进而识别问题中的实体 | 分词得到的实体需要合理且有效 | 随机输入多个问题，使用 jieba 分词工具进行实体识别，输出对应实体转化成的词向量 | 识别的实体均合理有意义，输出的词向量均有效 |
| 文档检索 | 系统能否根据问题正确寻找到相关文档 | 寻找到的文本文档块和图片需要与问题有关联，且需要保证对与问题相关的文档有较好的召回率 | 针对输入问题进行文档检索，打印输出检索到的具体内容 | 检索到的内容与问题相关程度较高，且在数量限制内与问题有尽可能高的相关度 |
| 基于 LangChain 的 Prompt 构建和大模型调用 | 能否基于问题和相关文档构建 Prompt，并调用大模型得到合理回答 | 大语言模型根据信息集合生成答案，若无法找到相关信息，则做拒绝回答 | 调用 LangChain 接口构建 Prompt 并调用大模型，打印输出回答 | 回答内容与文档信息密切相关，未出现幻觉现象 |

5.2.3 语音问答功能测试

语音问答功能测试共测试了语音生成 WAV 文件等四个具体项目，测试内容和结果如表 5.4 所示。

表 5.4 语音问答功能测试内容

| 测试项目 | 测试内容 | 测试要求 | 测试方式 | 测试结果 |
|-------------|-----------------------|--------------------|----------------------|------------------|
| 语音生成 WAV 文件 | 系统能否正常录音，录音后能否将音频从前端以 | 后端收到前端发送的非空 WAV 文件 | 多次在前端进行语音提问，在后端检查收到的 | 后端收到非空的 WAV 音频文件 |

| | | | | |
|------------------|---|--|---|------------------------------|
| | WAV 格式发送至 后端服务器 | | WAV 音频文件 | |
| 录音音质 | 检测所生成的 WAV 文件音质是 否达到要求 | 生成的 WAV 文 件录音内容相对 清晰可以辨认， 音质损失在可接 受范围内 | 多次在前端进行 语音提问，将后 端收到的 WAV 文件与直接录音 进行音质对比 | WAV 文件内容清 洗可辨，音质损 失可接受 |
| WAV 文件转 二进制文件 | WAV 文件所转换 导出的二进制文 件是否有效 | 导出的二进制文 件能够用于 API 语音转文字 | 将生成的二进制 文件内容及长度 输入 API，获得文 字输出 | 二进制文件所包 含的语音信息完 整，识别准确 |
| 语音提问总体 功能 | 能否正常通过调 用百度语音识别 API 识别输入的 问题语音 | 语音识别准确， 响应时间合理 | 在前端进行语音 提问，后端打印 输出转化生成的 文本问题 | 语音识别较为准 确，响应时间符 合预期 |

5.2.4 总体测试

本作品系统的总体测试共测试了用户界面交互等三个具体项目，测试内容和结果如表 5.5 所示。

表 5.5 总体测试内容

| 测试项目 | 测试内容 | 测试要求 | 测试方式 | 测试结果 |
|--------|-------------------------|-----------------------------|--|-------------------------------------|
| 用户界面交互 | 用户交互逻辑是否友好，页面反馈是否正确 | 用户能够以正常行为逻辑正常使用系统 | 安排 5 位从未接触过本项目的人员访问网站 | 5 位人员全部能够自主操作页面并使用问答机器人 |
| 系统兼容性 | 系统是否能够在不同的浏览器中提供服务 | 系统需要在主流浏览器中均可访问并提供服务 | 使用 Microsoft Edge, Google Chrome, 360 浏览器，火狐浏览器和 IE 浏览器访问并使用系统 | 所有浏览器均可正常访问并使用本系统 |
| 系统稳定性 | 系统能否在高并发高负荷条件下长时间稳定提供服务 | 系统在高负荷条件下能够访问，且性能减少需要在合理范围内 | 同时在 20 台主机通过浏览器访问网站并尝试进行问答 | 系统能够提供服务，且与单台主机访问相比产生一定延迟，延迟在可接受范围内 |

| | | | | |
|--------|------------|----------------|--------------------|---|
| 系统响应时长 | 系统响应时长是否合理 | 系统在较短时间内进行回答响应 | 在系统前端界面进行提问，记录响应时长 | 系统开始响应时长约为 3 秒，生成 500 字回答的总时长约为 15 秒，符合预期 |
|--------|------------|----------------|--------------------|---|

5.3 测试结论

经过对系统的全面测试与深入分析，本作品通过 B/S 架构部署的平台能够在公网正常访问，并且在高负荷情况下表现出了良好的稳定性。系统对于不同提问类型和不同相关文档的准确回答能力较强，能够参照文档内容提供客观、逻辑性的回答，证明了系统在文档检索和答案生成方面的有效性。系统的用户界面设计功能明确且交互流畅，对用户友好，符合用户的一般行为逻辑，在用户体验方面具有良好表现。

因此，经过测试验证，项目开发完成了预定目标 and 需求，可以进行交付使用。

6 未来展望

未来，我们期望从系统性能、模型架构、文档类型和安全性四个方面进一步完善本作品。

- 进一步优化系统架构、提升服务器性能、优化数据库查询和存储策略等方式来减少响应延迟，提升用户问答体验。
- 通过引入更先进的模型架构并优化算法，以提高模型的文本理解和生成能力，从而增强问答机器人的准确性和效率。利用用户反馈和自动评估机制对模型进行持续学习与更新。
- 采取一定的安全防护措施，建立完善的用户授权和访问控制机制，以确保系统的安全性并防止数据泄露。对系统中的敏感数据进行加密存储和传输，以防止数据泄露和非法访问。
- 补充扩展要求实现玲珑相关的问题回答。
- 完善系统前端界面，实现更加完备的文档问答。

7 比赛进展和收获

7.1 分工和协作

团队的系统性开发主要分为四个部分：

- 大模型构建与优化。基于最新的深度学习技术构建和优化了文档问答系统的大模型，对于提升系统的文本理解和回答准确性至关重要。
- Deepin 内容整合。整合 Deepin wiki 或其他相关文档资源，为大模型提供丰富的训练数据和知识库，确保数据的准确性和多样性，以满足不同查询场景的需求。
- 文档问答网页端实现。基于大模型和 Deepin 内容，实现了文档问答系统网页端的各项功能，不断优化系统性能，提升用户体验。
- 测试与评估。对系统进行测试和评估工作，确保系统的稳定性和准确性，通过自动化测试和人工测试相结合的方式，对系统进行全面的验证。

在协作方面，我们团队通过定期的线下项目会议、在线协作平台等方式，确保信息的及时传递和共享。各团队成员之间紧密配合，共同推进项目的进展。

7.2 收获与心得

本次比赛不仅让我们在技术上取得了显著的进步，更让我们在团队协作和行业应用方面获得了宝贵的经验，为我们未来的学习和工作提供有力的支持。

- 通过比赛，我们深刻认识到大模型在文档问答系统中的关键作用。大模型能够通过对海量数据的学习，提升对文本的理解和生成能力，从而提高问答系统的准确性和效率。Deepin wiki 等文档资源为文档问答系统提供了丰富的知识库，从而提升系统的性能。
- 在比赛过程中，我们遇到了许多技术挑战，如模型训练的效率、数据处理的准确性等。通过不断地尝试和实践，我们找到了相应的解决方案，并不断优化系统性能。比赛让我们更加深刻地认识到团队协作的重要性。只有团队成员之间紧密配合、相互支持，才能克服各种困难，取得成绩。

- 通过本次比赛，我们看到了文档问答系统在各行业中的广泛应用前景，并将在未来更多的领域进行实践与运用。

8 参考文献

- [1] Gao Y, Xiong Y, Gao X, et al. Retrieval-augmented generation for large language models: A survey[J]. arXiv preprint arXiv:2312.10997, 2023.
- [2] Ma X, Gong Y, He P, et al. Query rewriting for retrieval-augmented large language models[J]. arXiv preprint arXiv:2305.14283, 2023.
- [3] Ilin I. Advanced rag techniques: an illustrated overview[J]. 2023.
- [4] Lewis P, Perez E, Piktus A, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks[J]. Advances in Neural Information Processing Systems, 2020, 33: 9459-9474.
- [5] Zeng A, Liu X, Du Z, et al. Glm-130b: An open bilingual pre-trained model[J]. arXiv preprint arXiv:2210.02414, 2022.
- [6] Du Z, Qian Y, Liu X, et al. Glm: General language model pretraining with autoregressive blank infilling[J]. arXiv preprint arXiv:2103.10360, 2021.
- [7] Qiu Y, Li H, Li S, et al. Revisiting correlations between intrinsic and extrinsic evaluations of word embeddings[C]//Chinese Computational Linguistics and Natural Language Processing Based on Naturally Annotated Big Data: 17th China National Conference, CCL 2018, and 6th International Symposium, NLP-NABD 2018, Changsha, China, October 19–21, 2018, Proceedings 17. Springer International Publishing, 2018: 209-221.
- [8] Mihalcea R, Tarau P. Textrank: Bringing order into text[C]//Proceedings of the 2004 conference on empirical methods in natural language processing. 2004: 404-411.
- [9] Borgeaud S, Mensch A, Hoffmann J, et al. Improving language models by retrieving from trillions of tokens[C]//International conference on machine learning. PMLR, 2022: 2206-2240.
- [10] Ovadia O, Brief M, Mishaelli M, et al. Fine-tuning or retrieval? comparing knowledge injection in llms[J]. arXiv preprint arXiv:2312.05934, 2023.