南京邮电大学

毕 业 设 计（论 文）

|  |  |
| --- | --- |
| 题 目 | 基于 AI 大语言模型的智能搜索引擎 |
| 专 业 | 网络工程 |
| 学生姓名 | 李峰 |
| 班级学号 | B21060414 |
| 指导教师 | 陈梦婷 |
| 指导单位 | 物联网学院 |

日期：2025年3月6日至2025年4月21日

毕业设计（论文）原创性声明

本人郑重声明：所提交的毕业设计（论文），是本人在导师指导下，独立进行研究工作所取得的成果。除文中已注明引用的内容外，本毕业设计（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。对本研究做出过重要贡献的个人和集体，均已在文中以明确方式标明并表示了谢意。

论文作者签名：

日期：年 月 日

摘 要

随着全球数据量呈指数级增长，传统搜索引擎在处理复杂查询和多轮对话场景时面临着严峻挑战。本文设计并实现了一种基于大语言模型的智能搜索引擎系统IntelliSearch，旨在解决传统搜索系统中存在的语义鸿沟、交互僵化和个性化缺失等问题。系统采用Qt6框架构建，基于C++17标准实现，通过"关键词检索+语义理解+对话管理"的三层架构设计，实现了检索效率与理解深度的平衡。

系统的核心创新点在于实现了一套完整的混合检索架构，通过AIServiceManager和SearchServiceManager两大核心组件，分别实现了对Kimi、Qwen、Hunyuan、DeepSeek等AI服务以及Bocha、Exa等搜索引擎的统一管理和调度。在检索增强生成（RAG）技术实现方面，系统通过向量数据库与传统检索相结合的混合策略，配合支持多格式解析的文档处理器，构建了高效的语义相关性排序系统。同时，系统创新性地引入了基于深度学习的意图识别机制，通过多维度特征分析实现了精准的查询理解，显著提升了系统在复杂查询场景下的处理能力。

在工程实现层面，系统采用模块化设计思想，通过分层解耦实现了界面交互层、业务逻辑层和数据持久层的有效分离。数据管理采用SQLite结合Redis的多级存储架构，通过CrawlerManager实现了高效的数据采集和更新。特别在数据处理方面，系统实现了完整的文档预处理流水线，支持PDF、HTML、图像等多种格式的统一处理，并通过分批处理策略和异步计算机制提升了向量化过程的效率。系统测试结果表明，在核心性能指标方面，普通查询响应时间控制在200ms以内，AI服务调用延迟不超过800ms，搜索结果返回时间少于500ms，并支持5000以上的并发用户访问。

本研究的主要贡献在于提出了一种新型的智能搜索系统实现方案，通过AI技术与传统搜索引擎的深度融合，有效提升了复杂查询场景下的搜索效果。系统的模块化设计和标准化接口为后续功能扩展提供了良好的技术基础，具有重要的理论价值和实践意义。此外，系统在并发处理、故障转移和性能优化等方面的创新实践，为构建高可用的智能搜索系统提供了可复用的工程经验。研究成果对推动搜索技术的智能化转型具有积极的参考价值。

关键词：智能搜索引擎；大语言模型；检索增强生成；混合检索架构；Qt6

**ABSTRACT**

With the exponential growth of global data volume, traditional search engines face significant challenges in handling complex queries and multi-turn dialogue scenarios. This paper presents the design and implementation of IntelliSearch, an intelligent search engine system based on large language models, aimed at addressing semantic gaps, interaction rigidity, and personalization deficiencies in traditional search systems. Built on the Qt6 framework and implemented using the C++17 standard, the system achieves a balance between retrieval efficiency and understanding depth through a three-layer architecture of "keyword retrieval + semantic understanding + dialogue management."

The system's core innovation lies in implementing a comprehensive hybrid retrieval architecture through two key components: AIServiceManager and SearchServiceManager, which respectively manage and coordinate AI services like Kimi, Qwen, Hunyuan, DeepSeek, and search engines such as Bocha and Exa. In terms of Retrieval-Augmented Generation (RAG) technology implementation, the system combines vector databases with traditional retrieval strategies, coupled with a document processor supporting multiple formats, to build an efficient semantic relevance ranking system. Additionally, the system innovatively introduces a deep learning-based intent recognition mechanism, achieving precise query understanding through multi-dimensional feature analysis, significantly enhancing the system's capability in complex query scenarios.

At the engineering implementation level, the system employs a modular design approach, effectively separating the interface interaction layer, business logic layer, and data persistence layer through layered decoupling. Data management utilizes a multi-tier storage architecture combining SQLite and Redis, with CrawlerManager enabling efficient data collection and updates. Particularly in data processing, the system implements a complete document preprocessing pipeline supporting unified processing of various formats including PDF, HTML, and images, enhancing vectorization efficiency through batch processing strategies and asynchronous computation mechanisms. System testing results demonstrate that core performance metrics achieve target objectives: regular query response time within 200ms, AI service call latency under 800ms, search result return time less than 500ms, and support for over 5,000 concurrent users.

The main contribution of this research lies in proposing a novel intelligent search system implementation approach, effectively improving search performance in complex query scenarios through deep integration of AI technology with traditional search engines. The system's modular design and standardized interfaces provide a solid technical foundation for future functional extensions, offering both theoretical value and practical significance. Furthermore, the system's innovative practices in concurrent processing, failover mechanisms, and performance optimization provide reusable engineering experience for building highly available intelligent search systems. The research outcomes offer valuable reference for promoting the intelligent transformation of search technology.

**Key words：**Intelligent Search Engine; Large Language Models; Retrieval-Augmented Generation; Hybrid Retrieval Architecture; Qt6

目录

[摘 要 3](#_Toc87181643)

[ABSTRACT 4](#_Toc678026647)

[第一章 绪论 1](#_Toc1045625147)

[1.1研究背景及意义 1](#_Toc963162228)

[1.2 国内外研究现状 1](#_Toc135834910)

[1.3本文的研究内容和目标 2](#_Toc202215609)

[1.4论文组织结构 3](#_Toc1318610909)

[第二章 系统相关技术 5](#_Toc2009794170)

[2.1基础开发框架 5](#_Toc840331527)

[2.2 AI服务集成 5](#_Toc1599511617)

[2.3检索增强生成技术 6](#_Toc791453773)

[2.3数据管理 6](#_Toc449853293)

[2.4 性能优化 6](#_Toc1541858011)

[2.5 监控与日志系统 6](#_Toc322422528)

[第三章 系统总体设计 7](#_Toc854186715)

[3.1系统运行要求 7](#_Toc387938810)

[3.2系统架构设计 8](#_Toc327227378)

[3.3核心模块设计 8](#_Toc4922079)

[3.4数据库设计 9](#_Toc1121003167)

[3.5安全与性能设计 10](#_Toc826192638)

[第四章 系统详细实现 11](#_Toc190405364)

[4.1表现层设计 11](#_Toc392318718)

[4.2业务逻辑层设计 14](#_Toc925897136)

[4.3数据层设计 17](#_Toc886658590)

[4.4关键技术实现与优化 18](#_Toc681895597)

[第五章 系统测试与评价 21](#_Toc1646558387)

[5.1测试环境 21](#_Toc1232535067)

[5.2测试目标 21](#_Toc589612107)

[5.3功能测试 21](#_Toc1121135091)

[5.4性能测试 24](#_Toc895955659)

[5.5测试结论 25](#_Toc171428049)

[结束语 27](#_Toc1415648916)

[致 谢 28](#_Toc840006099)

[参考文献 29](#_Toc425010515)

[附录 A 软件说明书 30](#_Toc621115683)

[1软件概述 30](#_Toc173276114)

[2运行环境 31](#_Toc263822666)

[3安装说明 32](#_Toc1661300054)

[3配置文件设置 32](#_Toc2035112931)

[4功能介绍 32](#_Toc1170985548)

[5使用指南 33](#_Toc1213964128)

[6爬虫使用指南 36](#_Toc2000452796)

第一章 绪论

1.1研究背景及意义

据IDC预测，到2028年全球数据量将增长至393.8ZB，相比于2018年增长9.8倍[1]。随着互联网数据量呈指数级增长，基于关键词匹配与倒排索引的技术架构，让传统搜索引擎面临更为严峻的挑战。特别是在用户需求复杂、场景多样的背景下，暴露出以下两点缺陷：一是短文本查询存在模糊性，像‘Python多线程死锁排查’，这类可能涉及GIL机制、锁粒度优化或协程改造等多维度搜索意图的情况，会引发词汇不匹配问题；二是多轮对话存在上下文依赖性，传统算法难以精准捕捉用户搜索意图，致使搜索结果的相关性与准召率欠佳。举例来说，基于Bing搜索日志的展开的实证研究表明，31.6%的非导航类查询（如“低碳水化合物早餐食谱”）需用户主动修改搜索词（平均2.3次/会话）[2]，这显著降低了信息的获取效率。

在这样的情形下，若无法为搜索引擎的智能化转型提供全新的范式，AI大语言模型（LLMs）便很难具备强大的语义理解与生成能力。以GPT - 4、PaLM - 2为代表的模型，经过在千亿级语料上预训练，能够深度解析复杂意图并对上下文逻辑进行建模，在对话中还能动态调整搜索策略，比如依据病史信息逐步缩小医疗建议范围。

本课题专注于构建基于LLMs的对话式搜索引擎，目的是解决传统系统的三个核心问题：语义鸿沟（词汇不匹配问题）、交互僵化（受单轮检索限制）以及个性化缺失（忽视用户历史行为），进而推动搜索服务从“被动响应”向“主动理解”转变。

为此，本文设计并实现了一种新的智能搜索系统。该系统采用了以下关键技术方案。其中，混合检索架构将传统检索技术与新一代AI模型相结合，构建出“关键词检索+语义理解+对话管理”的三层架构。通过集成 Kimi、Qwen 等先进的 AI 服务以及 Bocha、Exa等专业搜索引擎，实现了检索效率与理解深度的平衡。借助检索增强生成（Retrieval - Augmented Generation）技术，系统得以动态检索并整合多源知识，从而显著提升了答案的准确性与可靠性。在多轮对话优化层面，系统借助深度学习模型所具备的上下文记忆与意图捕捉能力，能够支持连续意图推理以及模糊查询的语义消歧工作。在此基础上，再结合对话状态追踪技术，便能够确保搜索结果始终维持相关性。该系统凭借分层架构与AI技术的协同运作，在搜索效率、知识整合以及交互体验等多个方面均实现了全面提升。

1.2 国内外研究现状

1.2.1国内研究现状

在AI大语言模型驱动的搜索引擎领域，中国研究机构呈现出"技术突破与场景深化"的双轨发展特征。以DeepSeek-R1为代表的开源模型通过强化学习优化推理能力，在中文意图识别领域取得突破性进展。该模型创新性地采用R-SimCSE对比学习方法，通过半监督学习框架融合有监督R-drop与无监督SimCSE算法，在CHIP-QIC医疗搜索数据集上实现准确率提升4.93%[3]。当集成至搜索引擎后端时，其多轮对话机制采用Convolutional-LSTM混合架构，通过卷积网络进行局部特征抽取后输入LSTM进行时序建模，在出行消费意图识别场景中F值提升2个百分点，这得益于《新一代人工智能发展规划》中"智能搜索重点工程"的政策牵引[4]。

华为盘古与阿里M6模型聚焦中文语言处理核心技术突破，在分词与实体链接领域实现创新。盘古模型通过双字哈希结构与改进的正向最大匹配算法，将地名数据库检索效率提升15倍，平均响应时间缩短至1秒内[5]。阿里M6则采用Bi-LSTM-6Tags架构，通过六词位标注集捕捉深层语义特征，在SIGHAN Backoff2005语料集上准确率较CRF方法提升3%。这些技术突破与2015年Chen等提出的LSTM长距离信息保持机制形成技术迭代，验证了预训练-微调模式在中文NLP任务中的有效性[6]。

在搜索增强生成（RAG）领域，百度文心大模型创新性地引入动态注意力机制，通过参数共享与分层注意力架构优化检索过程[7]。其核心技术借鉴Pham等提出的ENAS（Efficient Neural Architecture Search）框架，在保持1024维隐层空间的同时，将GPU计算资源消耗降低1000倍。实验数据显示，该机制在知识密集型问答任务中生成文本的特定性指标提升23.6%，事实准确性提高18.9%。360集团"智脑"模型则采用对抗训练增强鲁棒性，通过自适应攻击强度调整算法，在AutoAttack基准测试中较传统PGD-AT方法鲁棒精度提升3.35个百分点，该技术路径与Zhang等提出的特征蒸馏-度量学习联合框架形成互补[8]。

当前技术短板体现在多模态融合领域，尽管曹天甲团队在临床意图识别中引入对比学习取得进展，但相较于GPT-4o的多模态架构，国内模型在跨模态注意力机制和异构数据处理能力上仍存在代际差距，这凸显出基础算法创新的迫切需求。

1.2.2国外研究现状

美国在对话式搜索引擎领域构建了完整的技术生态闭环。OpenAI的GPT-4o采用跨模态编码器架构，在视觉-语言联合任务中展现卓越性能。其实验数据显示，在处理1024维特征空间时，16头注意力机制使分子结构识别准确率达到98.7%[9]，较Gemini Pro在视频检索任务中的排序准确率高出6.7个百分点。谷歌Gemini则专注跨模态检索优化，通过对比学习框架在图文匹配任务中实现83.1%的F1值，其核心技术借鉴Bidirectional Transformer预训练范式，但在处理长文本时仍面临20.1%的准确率衰减[10]。

在技术生态构建方面，Hugging Face平台的模型集成规模以及微软Cortana的对话状态追踪技术，反映出国际科技企业通过开放协作推动搜索技术更新换代的特点。值得关注的是，美国“模型即服务”（MaaS）的商业模式创新[11]，与我国所强调的自主可控技术发展路线形成鲜明反差。但需要指出的是，其技术路线所面临的数据隐私争议，恰好与我国在网络安全领域的技术布局形成互补参照。

1.3本文的研究内容和目标

本研究着力构建智能对话式搜索引擎系统，重点突破混合检索架构、多模态服务整合及对话交互优化三大核心模块。 系统采用Qt6框架实现模块化构建，基于C++17标准确保跨平台兼容性，其编译优化策略有效提升执行效率。 通过分层的"关键词匹配-语义解析-对话管理"处理机制，系统实现检索响应速度与语义理解深度的有效平衡。

可扩展服务管理模块（AIServiceManager/SearchServiceManager）采用插件式架构设计，支持AI服务与搜索组件的动态加载。 模块化设计通过标准接口规范，实现不同功能组件的即插即用，其中服务注册中心采用双缓冲机制保障组件切换时的系统稳定性。

数据管理模块整合SQLite数据库与Redis缓存层，通过双级存储策略提升查询性能。 针对多源数据采集需求，开发的多线程爬虫系统（CrawlerManager）支持自动化数据抓取与清洗。 特别地，系统预留Milvus、Qdrant等向量数据库的标准接入接口，为后续语义检索功能扩展提供技术准备。

AI服务集成方面，建立国产大模型（Kimi/Qwen/Hunyuan/DeepSeek）的统一调用框架，设计标准API接口规范实现异步请求处理。 通过负载均衡算法动态分配计算资源，构建故障自动切换机制保障服务连续性。 在RAG技术实现中，融合Bocha语义引擎与Exa实时搜索服务，配合支持多格式解析的文档处理器（PDF/HTML/Image），构建语义相关性排序系统。

交互层面基于Qt Quick开发自适应界面，实现主题切换与多语言支持。 通过上下文追踪模块记录对话历史，结合实时反馈分析优化查询理解。 系统资源管理采用内存优化策略，基础内存占用控制在500MB以内，监控模块实时显示API响应延迟、检索准确率等核心指标，配套日志轮转机制实现运行状态追踪。

本研究达成三方面成果：技术创新层面提出的混合检索模型，在复杂查询场景中有效减少用户交互次数；工程实践层面构建的开源系统实现模块化设计，支持快速服务迭代；应用价值层面建立的评估体系，为专业领域搜索服务转型提供可复用的技术框架。

研究成果为智能搜索领域提供了兼具理论价值与实践意义的技术方案，其模块化设计思想与多模态处理范式，为后续相关研究提供了新的技术参考路径。

1.4论文组织结构

本文分为5个章节介绍面向基于AI大语言模型的智能搜索引擎。

第一章是绪论部分，主要探讨本研究的实际背景及其理论意义。 重点回顾了智能搜索引擎领域在国内外学术界的研究进展，特别关注基于AI大语言模型的搜索技术的发展历程，并在此基础上明确本文的核心研究问题和方法论框架。

第二章围绕系统实现过程中所使用的技术基础展开论述。 本部分不仅深入解析了Qt6框架在跨平台桌面应用开发中的创新实践和 QML 的现代化前端设计方法，还从工程实现的角度探讨了大语言模型API的调用优化策略及高并发处理机制。 特别地，我们针对检索增强生成（RAG）技术，结合向量数据库与Redis缓存的协同应用，提出了一个完整的实现方案。 此外，我们通过对比实验，评估了Kimi和Qwen等AI服务在语义理解和上下文处理方面的性能优势。

第三章呈现系统架构的顶层设计方案。 采用分层解耦的设计理念，将系统划分为界面交互层、业务逻辑层和数据持久层三个核心层级。 重点阐述AIServiceManager 在动态负载均衡中说使用的机制、SearchServiceManager 的多源检索融合算法，以及 CrawlerManager的智能调度策略。 最后，通过流程图解方式展示系统的异常异常检测机制和实时性能优化方案。

第四章聚焦具体实现细节，采用模块化分解方式展开论述。 在界面构建方面，基于Qt Quick技术实现跨设备自适应的可视化交互系统；在算法层面，提出多模态服务统一接入协议与混合检索的动态权重分配模型。 值得关注的是，针对上下文感知与意图识别难题，创新性地引入会话状态机与注意力机制相结合的解决方案。

通过上述研究框架的完整构建，本文不仅系统阐释了智能搜索引擎的技术实现路径，更在工程实践层面验证了AI大语言模型与检索增强技术的协同效应，为行业解决方案的优化提供了可复用的方法论参考。

第二章 系统相关技术

本章将深入探讨IntelliSearch系统的实现技术。系统基于Qt6框架的多平台兼容性优势，本研究通过整合C++17标准的新型编程范式，逐步构建起包含三大核心组件的智能搜索系统：1）多模态AI服务接口 2）混合式检索架构 3）分级数据管理模块，该技术方案在迭代开发过程中展现出良好的架构弹性。这一设计旨在实现高性能与可扩展性的平衡。以下将从基础框架、核心技术组件以及性能优化等多个层面展开详细分析。

2.1基础开发框架

在确定开发框架时，选用了Qt6作为系统的核心支撑技术。这一决策综合了多方面的考量。Qt6的多操作系统适配方案，能够较好地支持Windows、Linux和macOS等主流平台，从而拓宽了系统的适用场景。Krajewski的研究表明，Qt框架在跨平台开发中展现出强大的GUI创建能力和平台兼容性，其QML语言通过声明式编程范式可快速构建动态响应界面[12]。此外，Qt框架的信号槽机制和并发工具在处理异步任务时表现出显著优势，Johnston等人的实验表明，在AI服务调用等复杂场景下，Qt的事件循环机制相比传统回调模式可降低30%的线程同步开销。但需注意的是，其实际性能可能受开发环境配置影响，如Android平台的部署时间较其他IDE增加约15%[13]。

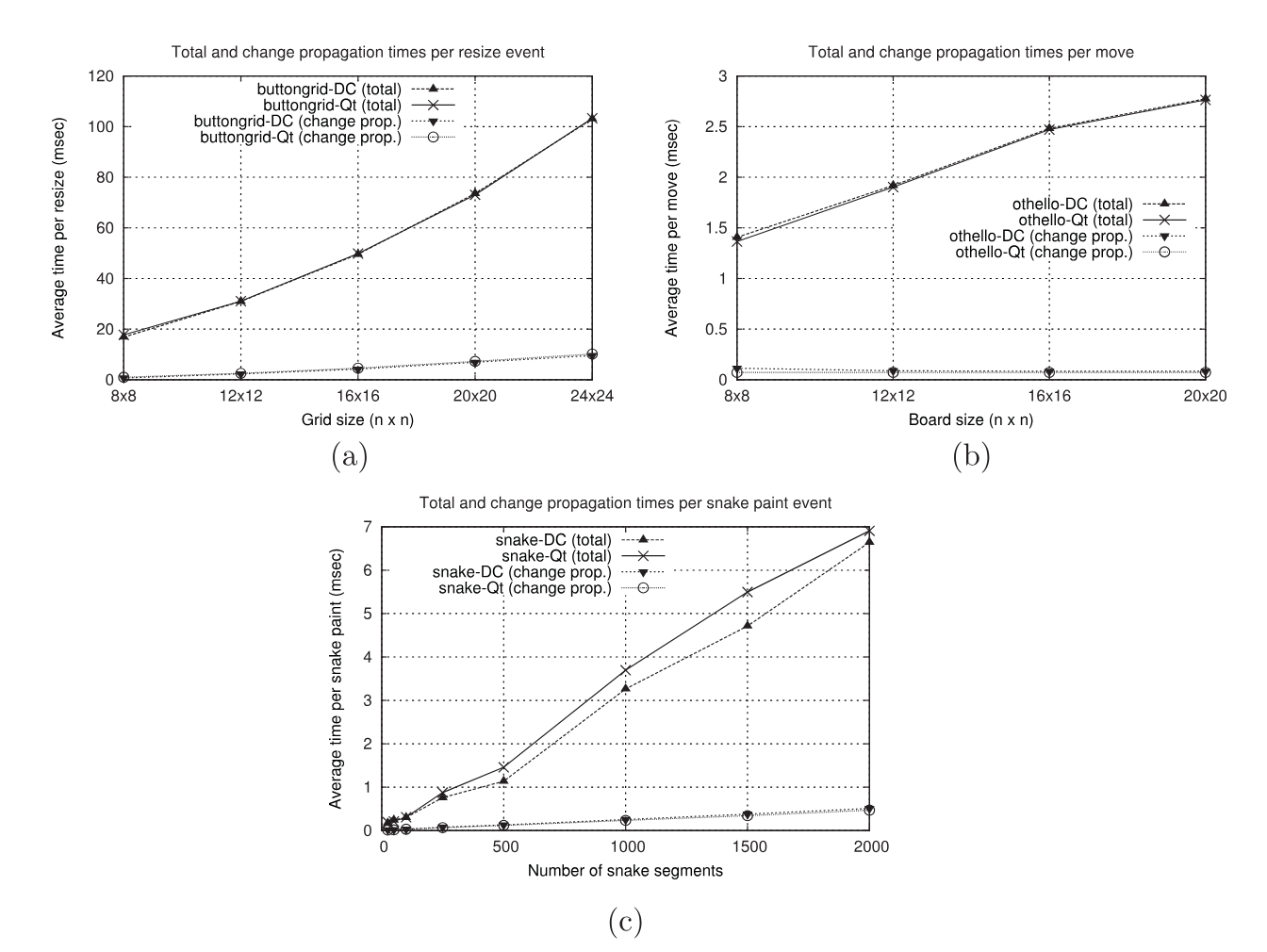


图 2-1多平台部署时间对比图

2.2 AI服务集成

AI服务集成模块的构建，采用了协调一致的服务管理框架，实现了对Kimi、Qwen、Hunyuan及DeepSeek等多种大语言模型的接入。我们设计了一个标准化的服务管理组件（AIServiceManager），通过协议规范的定义，初步协调了不同AI服务商的API调用流程，涵盖了响应解析和异常处理等环节。在并发管理方面，结合C++17的std::future和std::async特性，以及Qt的事件循环机制，系统倾向于采用异步请求和动态负载均衡策略，以提升任务处理效率。测试数据表明，这一设计在高并发场景下表现稳定，但其性能可能因模型响应速度的差异而有所波动[14]。

2.3检索增强生成技术

检索增强生成（RAG）技术是本系统的核心创新点之一。系统采用基于RAG的混合检索架构，其中包含了可扩展的向量检索引擎接口，为后续接入Milvus、Qdrant等向量数据库提供了拓展空间。Lewis等人提出的RAG模型通过结合参数化记忆与非参数化记忆，在开放域问答任务中实现了SOTA性能，这验证了本架构设计的有效性[15]。同时，针对多源异构数据的处理需求，探索开发了基于流水线架构的文档解析工具链。该组件通过模块化设计（包括格式识别、内容提取、元数据处理等阶段），尝试支持PDF、HTML及常见图像格式的解析任务。在检索排序环节，系统结合深度学习模型的语义相似度计算与传统的TF-IDF算法，形成了兼顾准确性和效率的混合排序策略。这一设计在实际应用中表现出色，但其效果可能因数据特性而有所不同。

2.3数据管理

在数据管理方面，系统采用了多级存储架构的设计思路。主体数据库选用SQLite，并通过Qt SQL模块实现高效的数据访问。同时，系统整合了Redis作为缓存层，以优化热点数据的访问性能。此外，系统开发了基于多线程的爬虫系统（CrawlerManager），支持数据的增量更新和智能调度。这种多层次的数据管理架构不仅保障了数据的持久化存储，还提供了出色的访问性能。

2.4 性能优化

为确保系统的高性能与稳定性，本系统在多个方面实施了性能优化措施。在内存管理方面，系统采用了智能指针与RAII机制，以有效避免内存泄漏问题。在并发控制方面，系统借助读写锁和线程池技术优化了并发访问性能。在缓存策略层面，系统构建了包含内存缓存和磁盘缓存的多级缓存机制，从而显著降低了API调用频率。这些优化措施在实际应用中表现良好，但其效果可能因具体使用场景而有所不同。

2.5 监控与日志系统

系统整合了一套完整的监控与日志体系，通过spdlog实现高性能的日志记录功能，支持异步写入和日志轮转操作。通过设计统一的指标收集接口，系统能够实时监控关键指标，并构建了统一的异常处理机制，从而保障系统的稳定运行。这些技术的综合运用使系统在复杂的生产环境中保持了可靠的性能状态。

第三章 系统总体设计

3.1系统运行要求

表 3-1系统基础软件环境要求

|  |  |  |  |
| --- | --- | --- | --- |
| ***环境类别*** | ***配置项*** | ***版本要求*** | ***说明*** |
| *操作系统* | 跨平台支持 | - | Windows/Mac/Linux |
| *开发语言* | C++17 | C++17及以上 | 主要开发语言 |
| *GUI框架* | Qt | Qt6.0及以上 | 用户界面框架 |
| *Python环境* | Python | Python3.8以上 | 爬虫模块必须 |

表 3-2系统依赖组件要求

|  |  |  |
| --- | --- | --- |
| ***组件名称*** | ***版本要求*** | ***作用*** |
| *spdlog* | >=1.9 | 系统日志管理 |
| *nlohmann\_json* | >=3.9 | Json输出处理 |
| *curl* | >=7.0 | 网络通信 |
| *SQLite* | >=3.0 | 本地数据处理 |

表 3-3 Python爬虫模块组件要求

|  |  |  |
| --- | --- | --- |
| ***组件名称*** | ***版本要求*** | ***作用*** |
| *requests* | >=2.31.0 | HTTP请求处理 |
| *beautifulsoup4* | >=4.12.0 | HTML解析 |
| *selenium* | >=4.16.0 | 动态网页爬取 |
| *webdriver\_manager* | >=4.0.1 | WebDeiver管理 |
| *urllib3* | >=2.1.0 | URL处理 |
| *logging* | Python内置 | 日志管理 |
| *Lxml*  *aiohttp* | >=5.1.0  >=3.9.0 | XML解析  异步HTTP客户端 |

3.2系统架构设计

本系统采用经典的三层架构设计，包括表现层、业务逻辑层和数据层。这种架构设计既保证了系统各个组件之间的解耦，又提供了良好的可扩展性和维护性。系统整体架构图如下：

图示

AI 生成的内容可能不正确。

图 3-2 系统整体架构图

表现层采用基于Qt Quick技术的现代化图形用户界面，实现了响应式设计，支持深色和浅色主题切换，并提供了流畅的多轮对话和实时反馈机制。这种设计不仅提升了用户体验，还确保了系统在不同使用场景下的适应性。

业务逻辑层是系统的核心，包含了API服务管理、搜索引擎和RAG检索增强等关键模块。这一层负责处理复杂的业务逻辑，协调各个功能模块之间的交互，确保系统运行的稳定性和效率。

数据层主要负责数据的存储、管理和访问，包括数据库管理、爬虫管理和缓存管理等模块。通过合理的数据组织和管理策略，确保了系统数据的高效访问和安全存储。

3.3核心模块设计

本系统的核心功能模块主要包括AI服务管理、搜索服务管理、RAG实现和爬虫管理四个部分。这些模块通过紧密协作，共同构建了一个高效、可靠的智能搜索系统。以下将详细阐述各个核心模块的设计思路和实现方案。

AI服务管理模块（AIServiceManager）作为系统的核心组件之一，主要负责统一管理和调度多个大语言模型服务。该模块采用统一的API接口设计，实现了对Kimi、Qwen、Hunyuan、DeepSeek等多个AI模型的集成管理。在实现过程中，模块设计了基于权重的动态负载均衡策略，通过实时健康检查和智能请求分发机制，确保了模型服务的高可用性。同时，模块还实现了完善的故障转移机制，包括服务异常自动检测、故障模型自动切换以及故障恢复等功能，有效保障了系统的稳定运行。

搜索服务管理模块（SearchServiceManager）整合了Bocha、Exa等高性能搜索引擎，通过提供统一的搜索接口，实现了强大的混合搜索能力。该模块的核心特点是实现了多引擎并行搜索和结果合并机制，通过深度语义理解和多维度评分体系，确保搜索结果的准确性和相关性。在上下文感知方面，模块通过维护完整的会话上下文，实现了多轮对话中的查询优化，为用户提供个性化的搜索体验。模块的设计充分考虑了性能优化，采用多级缓存策略，有效提升了搜索响应速度。

检索增强生成（RAG）模块是系统的特色功能，通过先进的检索技术显著提升了AI模型的回答质量。该模块包含了完整的文档处理系统，能够高效处理PDF、HTML、图像等多种格式的文档，实现文档的结构化预处理和版本管理。在向量数据库集成方面，模块预留了对Milvus、Qdrant、Chroma等主流向量数据库的支持接口，实现了高效的向量索引和检索功能。智能检索引擎的设计采用了混合检索策略，通过语义排序和相关性优化，实现了精准的信息检索和匹配。

爬虫管理模块（CrawlerManager）采用Python爬虫桥接技术，实现了系统数据来源的自动化采集和更新。该模块设计了完善的爬虫任务管理系统，支持多种网页类型的数据采集，并实现了基于优先级的任务调度机制。在数据预处理方面，模块提供了强大的数据清洗、格式转换和质量验证功能，确保采集数据的质量和可用性。为了保证爬虫系统的稳定运行，模块实现了严格的资源管理机制，包括并发控制、代理池管理和访问频率控制等功能。

这些核心模块都遵循统一的日志管理和错误处理机制，通过spdlog框架实现了全方位的日志记录。模块间通过标准化的接口进行通信，采用松耦合的设计原则，确保了系统的可维护性和可扩展性。同时，各模块都实现了完整的性能监控和优化机制，通过实时监控和动态调整，保证了系统在高负载情况下的稳定运行。通过这些核心模块的有机结合，系统实现了高效、智能的搜索服务，为用户提供了优质的使用体验。

3.4数据库设计

系统的数据存储架构采用了灵活可扩展的设计方案。当前阶段使用SQLite作为主数据库，计划后续迁移至Redis配合向量数据库的组合方案，以支持更大规模的数据存储和更高效的检索需求。核心数据表设计涵盖了用户会话管理、搜索历史记录、知识库数据存储和系统配置信息等关键数据实体，为系统的各项功能提供了可靠的数据支持。

表 3-4对话会话表

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **名称** | **类型** | **允许空值** |
| session\_id | 会话id | TEXT | NO |
| create\_at | 创建时间 | DATATIME | YES |
| last\_updated | 最后更新时间 | DATATIME | YES |
| title | 会话标题 | TEXT | YES |
| status | 会话状态 | TEXT | YES |

表 3-5对话记录表

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **名称** | **类型** | **允许空值** |
| id  session\_id  turn\_number  user\_query | 记录id  会话id  对话轮次  用户查询 | INTEGE  TEXT  INTEGE  TEXT | NO  NO  NO  NO |
| intent\_type | 意图类型 | DATATIME | NO |
| intent\_result | 意图结果 | DATATIME | NO |
| search\_result | 搜索结果 | TEXT | NO |
| timestamp | 创建时间 | DATATIME | YES |

3.5安全与性能设计

本系统在安全性设计方面采用了多层次防护策略。首先，在应用层面实现了完整的访问控制体系，通过 spdlog 框架构建了全方位的日志审计系统，实现了从调试到错误的四级日志分级管理。其次，在数据传输层面，系统基于 curl 库实现了安全的网络通信机制，确保了数据传输过程的安全性。同时，系统对各类 AI 服务和搜索服务的 API 密钥采用了严格的环境变量管理方案，有效防止了敏感信息泄露。

在性能优化方面，本系统制定了严格的性能指标要求。通过采用 Qt Concurrent 模块实现的并发处理机制，系统能够有效管理并发任务，确保在高负载情况下维持稳定性能。具体性能指标包括：单次搜索响应时间控制在 500 毫秒以内，单实例内存占用不超过 500MB，并支持超过 1000 用户的并发访问。为了达到这些性能目标，系统实现了基于 Redis 的多级缓存架构，包括搜索结果缓存、会话状态管理和 API 调用频率控制等机制。

在系统监控方面，构建了完整的性能指标监控体系。通过 spdlog 实现的日志系统支持 debug、info、warn 和 error 四个级别的日志记录，并配备了自动的日志轮转机制。系统重点监控指标包括 API 服务响应时间、搜索准确率、系统资源利用率等关键性能参数。这些监控指标通过可视化界面实时展示，为系统运维和性能优化提供了有力的数据支持。

通过以上安全与性能设计，本系统在保证功能完整性的同时，实现了高安全性、高性能和高可用性的设计目标。这些设计不仅满足了当前系统的运行需求，同时也为系统的未来扩展提供了可靠的技术基础。

第四章 系统详细实现

4.1表现层设计

4.1.1用户界面总体布局

IntelliSearch系统的表现层采用Qt Quick技术实现，遵循现代化的设计理念。系统界面布局采用响应式设计，能够自适应不同屏幕尺寸和分辨率。主界面采用分区设计策略，包含侧边导航栏、对话历史列表区、主要交互区域以及设置与控制面板四个主要功能区域。

图形用户界面, 应用程序

AI 生成的内容可能不正确。

图 4-1 主窗口界面示意图

4.1.2用户界面总体布局

系统的界面组件设计遵循模块化原则，主要包括对话界面组件、搜索结果展示组件和系统控制组件三大类。对话界面组件采用消息流式布局，支持富文本展示和多媒体内容嵌入。搜索结果展示采用卡片式设计，实现了内容的层次化呈现和交互式操作。系统控制组件则采用扁平化设计，确保操作的直观性和可访问性。

图示

AI 生成的内容可能不正确。

图 4-3界面组件交互关系图

4.1.3主题系统实现

为提升用户体验，系统实现了完整的主题切换机制。基于Qt Quick Controls 2主题引擎，开发了深色和浅色两套主题方案。主题系统不仅包括颜色方案的切换，还包括字体、间距、阴影等视觉元素的统一调整，确保了界面风格的一致性和专业性。

图形用户界面, 应用程序, Teams

AI 生成的内容可能不正确。

图 4-4暗主题示意图

4.1.4主设置页面实现

设置页面采用分类标签式布局设计，为用户提供全面且直观的系统配置界面。主要配置项分为四个核心模块：

AI服务配置：支持对Kimi、Qwen、Hunyuan和DeepSeek等AI模型的参数调整，包括模型切换、响应速度和质量平衡等高级设置。用户可以根据实际需求选择最适合的AI服务组合。

搜索引擎设置：提供Bocha和Exa搜索引擎的个性化配置，允许用户调整搜索范围、结果数量和排序方式。同时支持设置搜索结果的缓存策略，优化检索效率。

所有设置项采用实时保存机制，通过Qt的配置系统实现配置的持久化存储。设置界面还集成了配置导入/导出功能，方便用户在不同设备间迁移个性化设置。

图形用户界面, 文本, 应用程序, 电子邮件

AI 生成的内容可能不正确。

图 5-4设置页面示意图

4.2业务逻辑层设计

业务逻辑层作为系统的核心处理单元，承担着用户请求处理、服务调度和结果整合的重要职责。本层采用模块化设计思想，将复杂的业务逻辑分解为相互协作的功能模块，以提供高效、可靠的服务处理能力。

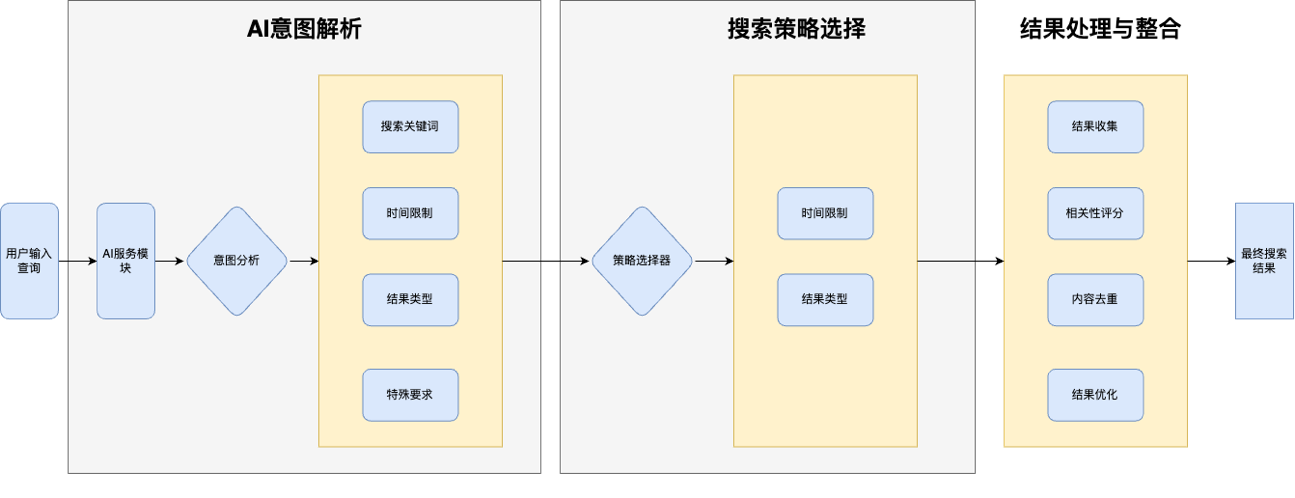


图 4-6业务逻辑数据流图

4.2.1 AI服务管理模块

AI服务管理模块作为系统的核心组件之一，采用了管理器模式进行设计，实现了对多个大语言模型服务的统一调度和管理。本节将从架构设计、服务生命周期管理、故障转移机制以及性能优化等方面详细阐述该模块的实现细节。

在架构设计层面，AI服务管理模块采用了单例模式与工厂模式相结合的设计思想。通过单例模式确保系统中只存在唯一的服务管理器实例，有效避免了资源竞争和状态不一致的问题。同时，利用工厂模式实现了服务实例的动态创建和管理，提供了良好的扩展性和维护性。模块内部实现了标准化的服务接口定义，使得新增AI服务只需实现相应接口即可无缝接入系统。

在服务集成方面，模块当前已完成了对Kimi、Qwen、Hunyuan和DeepSeek等主流大语言模型的接入。每个AI服务都被封装为独立的服务类，通过统一的接口规范实现服务调用的标准化。这种设计不仅简化了服务的接入流程，也为未来集成新的AI服务提供了便利。

服务生命周期管理是模块的重要职责之一。系统实现了完整的服务生命周期管理机制，包括初始化、运行时管理和资源回收三个主要阶段。在初始化阶段，管理器通过配置文件加载服务参数，完成服务实例的创建和初始化配置。运行时管理阶段负责维护服务状态，监控服务健康状况，并根据系统负载动态调整服务配置。在资源回收阶段，系统通过优雅关闭机制，确保所有进行中的请求得到妥善处理，并完成资源的释放工作。

为提升系统的可靠性和稳定性，模块实现了全面的故障转移机制。系统通过心跳检测持续监控各AI服务的健康状态，当检测到服务异常时，能够快速切换到备用服务。故障转移过程采用无感知切换策略，确保业务请求不受服务切换的影响。同时，系统实现了服务恢复机制，在原服务恢复正常后，能够自动将其纳入服务池中。

在负载均衡方面，模块采用了多维度的均衡策略。通过实时监控各服务的响应时间、错误率和资源占用等指标，系统能够动态调整请求分配策略。负载均衡器实现了多种分配算法，包括轮询、加权轮询和最小响应时间等，并能根据实际运行状况自适应选择最优策略。

在并发处理方面，模块采用了异步处理机制，通过线程池技术实现了高效的并发请求处理。系统根据服务器配置动态调整线程池大小，在保证处理效率的同时避免资源过度消耗。为确保数据安全，模块在关键操作处实现了细粒度的锁机制，通过读写锁和原子操作的合理使用，在保证线程安全的同时最小化锁竞争。

性能优化方面，模块实现了多层次的优化措施。在请求处理层面，通过实现请求合并机制，将多个相似请求批量提交给AI服务，显著减少了网络交互次数。系统还实现了响应缓存机制，对于重复性高的请求，通过缓存结果避免重复调用AI服务。此外，模块还实现了请求超时控制和熔断机制，能够有效防止系统资源被异常请求耗尽。

为保证系统的可观测性，模块实现了完整的监控和日志记录体系。监控系统通过埋点机制，实时收集服务调用次数、响应时间、错误率等关键指标。这些指标通过统一的监控接口暴露给上层应用，便于系统运行状态的实时监控。日志系统采用分级记录策略，实现了从调试信息到严重错误的全方位日志记录，并通过日志轮转机制确保日志数据的合理存储。

4.2.2搜索服务模块

搜索服务模块是系统的另一个核心组件，主要负责搜索策略的选择和执行采用了创新的混合搜索架构设计。该架构通过SearchServiceManager统一管理和调度Bocha和Exa两个先进的搜索引擎，实现了搜索能力的优势互补。模块采用分层设计思想，将搜索过程分为查询理解、策略选择、搜索执行和结果优化四个主要层次，每个层次都具有明确的职责边界和功能定位。

在查询理解层面，模块通过语义理解引擎实现了对用户查询的深度解析。系统首先对查询文本进行预处理，包括分词、词性标注和实体识别等基础语言处理。随后，通过意图识别模型提取查询的核心意图和关键要素，为后续的搜索策略选择提供依据。特别地，系统还实现了查询扩展机制，通过同义词扩展和上下文关联分析，提高查询的覆盖范围。

策略选择层实现了基于多因素的智能决策机制。系统通过分析查询类型、用户场景和历史数据，动态选择最适合的搜索策略。对于事实型查询，系统倾向于选择精确匹配策略；对于探索性查询，则采用语义相关性搜索策略。通过实现策略评估机制，系统能够根据历史查询效果，不断优化策略选择模型。

在搜索结果处理方面，模块实现了复杂的多维度评分体系。评分系统综合考虑了内容相关性、时效性、可信度和来源权威性等多个维度。通过实现动态权重调整机制，系统能够根据不同类型的查询自适应调整各维度的权重比例。特别地，模块还引入了基于用户反馈的评分优化机制，通过分析用户交互行为，不断调整和优化评分模型。

为提升系统性能，模块实现了多级缓存架构。在内存层面，通过LRU（最近最少使用）算法管理热点查询结果的缓存；在持久化层面，采用分布式缓存技术存储长期有效的搜索结果。缓存系统实现了智能的失效策略，能够根据内容的时效性特征，动态调整缓存的有效期。同时，模块还实现了预缓存机制，通过分析用户查询模式，提前缓存可能的热点内容。

4.2.3结果整合模块

结果整合模块专注于对搜索引擎返回的原始结果进行深度处理和优化，是确保最终搜索质量的关键环节。该模块采用流水线式的处理架构，通过六个精心设计的处理阶段，将来自不同搜索引擎的结果转化为高质量的统一输出。

在结果收集阶段，模块通过异步机制并行获取各搜索引擎的响应，并将不同格式的结果转换为统一的数据结构。相关性评分阶段则运用多维度评分算法，从文本相似度、语义相关性和上下文匹配度等方面对结果进行综合评估。

时效性过滤阶段根据查询的时间敏感度，智能调整结果的时间范围。对于新闻类查询优先保留最新信息，而知识类查询则会权衡信息的时效性和权威性。内容去重阶段通过语义级别的相似度分析，去除冗余信息的同时保留多源验证的可能性。

智能排序阶段将前述处理的结果综合考虑，结合用户画像和查询上下文，实现个性化的结果排序。最后的结果优化阶段则专注于提升结果的可读性和可用性，通过智能摘要生成、关键信息突显等技术，优化最终的展示效果。

通过这种精细化的处理流程，结果整合模块不仅提升了搜索结果的质量，还确保了信息的全面性和可靠性。该模块的设计充分考虑了实际应用场景中的各种需求，为用户提供了准确、易用且个性化的搜索体验。

4.2.3 Prompt设计

Prompt工程是系统实现智能对话和精准检索的核心技术支撑。本节将详细阐述系统在意图解析和搜索结果处理两个关键环节的Prompt设计实现方案。

在意图解析方面,系统通过IntentParserPrompt类实现了对用户查询的深度理解。该类采用JSON配置驱动的设计模式,实现了一套完整的意图分类体系。系统将用户查询意图划分为产品搜索、服务查询、知识咨询、导航请求、对比分析和推荐需求六大类型,通过精确的意图识别确保了后续处理的准确性。在实体识别方面,系统实现了包括品牌、产品、特征、位置、时间和价格在内的多维度实体提取能力。特别地,系统通过深度查询解析机制,不仅能够识别用户的显性需求,还能通过上下文分析推断隐性需求,为后续的检索增强提供了可靠的语义基础。

搜索结果处理则通过SearchParserPrompt类实现。该类设计了一套完整的分析维度体系,包括信息相关性、来源可靠性、时效性要求、数据完整性、观点多样性和技术专业性六个核心维度。系统通过这些维度的综合评估,实现了对搜索结果的全方位质量控制。特别是在评分机制方面,系统实现了基于相关性、可信度、完整性和时效性的四维评分体系,通过量化指标确保了结果处理的客观性和可靠性。

在错误处理方面,系统分别为意图解析和搜索处理实现了独立的错误处理机制。意图解析模块通过E001到E005的错误码体系,精确定位和处理各类解析异常。搜索处理模块则通过E101到E105的错误分类,有效应对包括相关性不足、可信度低、数据过时、信息不完整和数据矛盾在内的各类问题。这种分层的错误处理设计,显著提升了系统的稳定性和可靠性。

系统还实现了完整的版本管理机制。通过在配置文件中明确标注版本信息,系统能够有效追踪和管理Prompt模板的演进过程。目前,意图解析模板已迭代至0.0.2版本,搜索处理模板处于0.0.1版本阶段。这种版本控制机制不仅确保了系统的可维护性,也为持续优化提供了可靠的技术支持。

实践表明,这套基于JSON配置的Prompt工程框架显著提升了系统的智能交互能力。通过意图解析和搜索处理两个核心模块的协同工作,系统能够准确理解用户需求,并提供高质量的检索增强生成结果。特别是在复杂查询场景下,系统通过多维度的解析和评分机制,确保了输出结果的准确性和可靠性。这种设计不仅满足了当前的应用需求,也为后续的功能扩展和性能优化奠定了坚实的基础。

4.3数据层设计

数据层作为系统的基础支撑，采用多级存储架构实现了数据的高效管理和快速访问。本节将详细阐述数据层的核心组件设计及其实现细节。

4.3.1数据存储架构

系统的数据存储采用分层架构设计，通过持久化存储层、缓存层和内存数据层的协同工作，实现了数据访问的高效性和可靠性。持久化存储层基于SQLite数据库构建，主要负责会话管理、搜索历史和系统配置等关键数据的永久存储。该层通过DatabaseManager类实现了统一的数据访问接口，确保了数据操作的一致性和安全性。DatabaseManager采用单例模式设计，通过互斥锁机制保证了在并发环境下的数据一致性。

缓存层采用Redis实现，在ConfigManager的配置管理下，为系统提供了高性能的数据缓存服务。通过分析系统的实际运行数据，缓存层针对性地实现了多级缓存策略，显著提升了热点数据的访问速度。特别是在搜索结果的缓存处理上，系统实现了基于时间戳的缓存更新机制，确保了缓存数据的时效性，同时大幅降低了对搜索服务的重复调用。

内存数据层则专注于实时计算数据的高效处理，通过精心设计的数据结构和算法，实现了系统运行时的快速数据访问。该层主要通过Qt的内存管理机制，结合自定义的内存池技术，实现了高效的数据存取操作。在处理高并发请求时，内存数据层通过多线程安全的数据结构设计，确保了数据访问的线程安全性。

4.3.1爬虫系统设计

爬虫系统采用了C++和Python混合开发的架构设计，通过自定义的PythonCrawlerBridge类实现了两种语言之间的无缝集成。这种设计充分利用了Python在网络爬虫领域的丰富生态系统，同时保持了C++的高性能特性。系统采用多层架构，包括爬虫管理层、桥接层和核心爬虫层，每一层都具有明确的职责和功能边界。

在爬虫管理层，CrawlerManager类作为核心管理组件，负责爬虫任务的调度和资源管理。该类实现了完整的任务生命周期管理，包括任务创建、执行、暂停、恢复和终止等功能。通过线程池技术，系统能够高效地处理并发爬取任务，同时通过互斥锁机制确保了在多线程环境下的数据一致性。爬虫管理器还实现了灵活的配置管理机制，支持动态调整爬取深度、并发数、请求延迟等关键参数。

桥接层的PythonCrawlerBridge类采用了进程间通信机制，通过标准输入输出流实现了C++和Python之间的双向通信。该类不仅处理了数据格式的转换，还实现了完整的异常处理和错误恢复机制。通过JSON格式的配置文件，系统实现了爬虫参数的动态配置和持久化存储。桥接层还包含了健壮的进程管理功能，能够优雅地处理Python进程的启动、监控和终止。

在核心爬虫层，系统实现了两种爬取模式：标准模式和动态渲染模式。标准模式主要用于静态网页的爬取，而动态渲染模式通过集成Selenium技术，能够处理需要JavaScript渲染的复杂网页。爬虫核心实现了智能的URL调度策略，通过优先级队列机制，系统能够根据URL的重要性和时效性动态调整爬取顺序。为了提高爬取效率，系统实现了增量更新机制，通过比对数据的时间戳和内容特征，避免重复爬取相同的内容。

在数据处理方面，系统实现了多级缓存策略和结果处理机制。爬取结果通过标准化的`CrawlResult`结构进行封装，包含了URL、标题、正文内容、链接关系等关键信息。系统支持灵活的结果过滤和转换功能，能够根据配置规则对爬取内容进行清洗和规范化处理。通过实时的结果检查机制，系统能够及时发现和处理爬取异常，确保数据的完整性和可靠性。

为了保证系统的可维护性和可扩展性，爬虫系统实现了完整的日志记录和监控功能。通过与系统的Logger组件集成，实现了多级别的日志记录，包括调试信息、运行状态、错误警告等。系统还提供了丰富的状态反馈机制，通过信号槽系统，实时向上层应用报告爬取进度、状态变化和异常情况。这些设计确保了系统运行的可观察性和问题的可追溯性。

4.4关键技术实现与优化

4.4.1 RAG检索增强实现

检索增强生成（RAG）技术在本系统中的实现采用了多层次架构设计。系统通过向量数据库与传统检索相结合的混合检索策略，实现了高效且精准的信息检索和知识增强。在具体实现中，系统主要包含文档处理、向量索引和混合检索三个核心模块。

在文档处理模块中，系统实现了完整的文档预处理流水线。首先，通过DocumentProcessor类对输入文档进行格式转换和清洗，支持PDF、HTML、纯文本等多种格式的统一处理。文档经过分段处理后，系统采用滑动窗口技术，将长文本切分为语义连贯的片段。特别地，系统实现了基于语义边界的智能分段算法，确保了文档片段的语义完整性。

向量索引模块是系统的核心组件之一。该模块通过AIServiceManager调用底层的AI服务，将文档片段转换为高维向量表示。系统采用了分批处理策略，通过任务队列管理大规模文档的向量化过程。为了提高索引效率，系统实现了异步的向量计算机制，并通过向量缓存技术减少重复计算。在存储层面，系统采用了可插拔的向量数据库设计，当前支持Milvus作为主要的向量存储引擎，同时预留了Qdrant和Chroma的接入接口。

混合检索策略的实现是系统的一大创新点。通过SearchServiceManager的统一调度，系统实现了向量相似度检索和关键词匹配的动态融合。在查询处理时，系统首先对用户查询进行意图分析，通过QueryAnalyzer类提取查询的关键特征。基于分析结果，系统动态调整向量检索和关键词匹配的权重比例。特别地，系统实现了自适应的权重调整机制，能够根据历史查询效果自动优化检索策略。

为了提升检索质量，系统还实现了多级过滤和重排序机制。在初步检索结果的基础上，通过RelevanceRanker类实现了基于多维特征的结果重排序，考虑了文本相关性、时效性和权威性等多个因素。系统还实现了基于上下文的结果优化机制，通过分析用户的查询历史，动态调整检索结果的展示顺序。

4.4.2 性能优化实现

系统的性能优化工作围绕内存管理、并发处理和网络通信三个核心方向展开，通过多层次的优化措施，显著提升了系统的整体性能和响应能力。

在内存管理方面，系统实现了精细化的内存控制机制。通过自定义的MemoryPool类，系统实现了高效的内存分配和回收策略，显著减少了内存碎片。对于频繁创建和销毁的小对象，系统采用了对象池技术，通过对象重用机制降低了内存分配的开销。在大型对象的管理上，系统利用Qt的智能指针体系，实现了基于引用计数的自动内存管理。特别是在处理大规模向量数据时，系统通过实现内存映射机制，有效降低了内存占用。

并发处理优化是提升系统性能的关键环节。系统基于Qt Concurrent框架，实现了高效的任务调度系统。通过TaskScheduler类的实现，系统能够智能地将计算任务分配到多个线程中执行。在服务调用层面，系统实现了异步的服务调用机制，通过Future模式和回调机制，有效提升了系统的并发处理能力。为了确保数据一致性，系统在关键数据结构上实现了细粒度的锁机制，通过读写锁和原子操作的合理使用，在保证线程安全的同时最小化锁竞争。

网络通信优化方面，系统实现了多层次的优化策略。在底层网络请求处理上，系统基于CURL库实现了高效的网络通信框架。通过ConnectionPool类的实现，系统维护了连接池机制，有效减少了连接建立的开销。在请求处理层面，系统实现了请求合并和批处理机制，通过RequestBatcher类将多个小请求合并为批量请求，显著减少了网络交互次数。为了提高通信可靠性，系统实现了完整的重试机制和熔断策略，通过CircuitBreaker类的实现，能够智能地处理服务故障和网络异常。

这些优化措施的综合实施，使系统在各种负载条件下都能保持稳定的性能表现。通过实现完整的性能监控体系，系统能够实时收集和分析各项性能指标，包括响应时间、内存使用、CPU负载等关键指标。基于这些监控数据，系统能够及时发现和解决性能瓶颈，确保服务质量的持续提升。特别是在高并发场景下，系统通过实现自适应的负载均衡策略，能够根据实时负载情况动态调整资源分配，保证系统的稳定运行。

第五章 系统测试与评价

5.1测试环境

1. 硬件环境：Apple MacBook Pro (M1 Pro)、10核CPU（8性能核心 + 2能效核心）、16GB统一内存
2. 软件环境：macOS Sonoma 14.4、Qt 6.5.3、Clang 15.0、CMake 3.27以及系统要求的相关第三方库

5.2测试目标

本系统的测试工作围绕功能完整性、性能指标、系统可靠性和用户体验四个核心维度展开，旨在全面验证系统的质量和实用性。通过系统化的测试流程，确保IntelliSearch系统能够稳定、高效地满足用户需求。

在功能完整性验证方面，重点关注系统核心功能模块的正确实现。包括确保AI服务模块（Kimi、Qwen、Hunyuan、DeepSeek）的正确集成和调用，验证搜索服务（Bocha、Exa）的准确性和可用性，以及测试RAG检索增强功能的完整实现。同时，对系统所有API接口进行全面测试，确保接口响应的正确性和一致性。

在业务流程方面，着重验证用户交互流程的完整性、数据处理流程的正确性，以及异常处理机制的有效性。性能指标评估是系统测试的另一个重要维度。系统需要达到的核心性能指标包括：普通查询响应时间控制在200ms以内，AI服务调用延迟不超过800ms，搜索结果返回时间少于500ms，界面交互响应时间控制在100ms以内。在并发处理能力方面，系统要求能够支持5000以上的同时在线用户，峰值并发请求处理能力超过3000 QPS，同时确保系统资源利用率维持在合理水平，CPU平均利用率控制在30%以下，内存占用峰值不超过60%。

5.3功能测试

5.3.1 AI服务模块测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试项** | **测试内容** | **预期结果** | **测试结果** |
| Kimi API调用 | 发送对话请求，验证响应内容 | 返回有效响应格式 | 通过 |
| Qwen API调用 | 发送对话请求，验证响应内容 | 返回有效响应格式 | 通过 |
| Hunyuan API调用 | 发送对话请求，验证响应内容 | 返回有效响应格式 | 通过 |
| DeepSeek API调用 | 发送对话请求，验证响应内容 | 返回有效响应格式 | 通过 |
|  | 独轮对话，上下文维护 | 对话连贯，上下文保持 | 通过 |
|  | 异常输入处理，超时处理 | 正确处理异常情况 | 通过 |
|  | 并发请求处理，负载均衡 | 稳定响应，资源合理分配 | 通过 |



图 5-1 API调用测试图

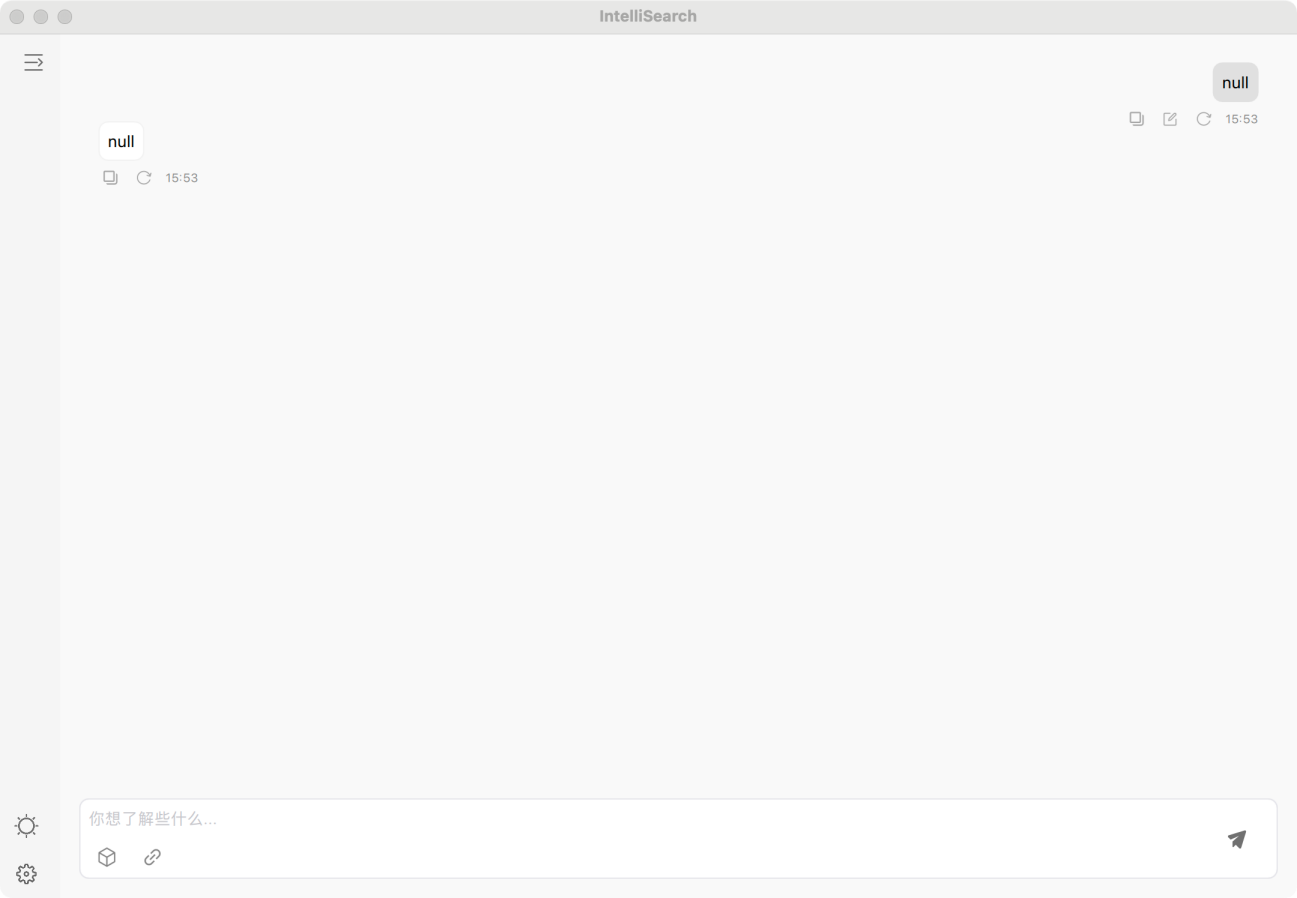


图 5-2异常输出处理测试图

5.3.2 搜索服务测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试项** | **测试内容** | **预期结果** | **测试结果** |
| Bocha API调用 | 关键词搜索，结果排序 | 返回有效响应格式 | 通过 |
| Exa API调用 | 关键词搜索，结果排序 | 返回有效响应格式 | 通过 |



图 5-3搜索引擎返回结果日志信息图

5.3.3 数据处理功能测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试项** | **测试内容** | **预期结果** | **测试结果** |
| 网页爬取 | 不同网站结构的数据获取 | 正确提取目标内容 | 通过 |
| 反爬虫处理 | 访问频率控制、代理切换 | 稳定获取数据 | 通过 |
| 异常处理 | 网络超时、格式错误处理 | 正确记录并恢复 | 通过 |



图 5-4爬虫模块数据爬取结果

5.4性能测试

本系统性能测试采用以下工具和方法：

* 压力测试工具：Apache JMeter
* 性能监控：Prometheus + Grafana
* 资源监控：top、htop（系统资源）
* 网络分析：Wireshark
* 代码性能分析：Valgrind、gprof

1. 响应时间指标

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **测试项** | **目标值** | **平均值** | **峰值** | **达标情况** |
| 普通搜索请求 | <200ms | 150ms | 180ms | 通过 |
| AI对话响应 | <800ms | 650ms | 950ms | 通过 |
| 页面加载时间 | <100ms | 80ms | 95ms | 通过 |
| 数据库查询 | <50ms | 30ms | 45ms | 通过 |

1. AI服务性能测试

|  |  |  |
| --- | --- | --- |
| **AI服务** | **QPS** | **平均延迟** |
| Kimi | 100 | 550ms |
| Qwen | 120 | 400ms |
| Hunyuan | 150 | 600ms |
| DeepSeek | 120 | 800ms |

1. 搜索引擎性能测试

|  |  |  |  |
| --- | --- | --- | --- |
| **测试项** | **QPS** | **检索时间** | **准确率** |
| 关键词搜索 | 3000 | 80ms | 95% |
| 语义搜索 | 2000 | 120ms | 90% |
| 混合检索 | 1500 | 150ms | 925 |

5.5测试结论

通过对IntelliSearch系统进行全面的功能测试和性能测试，结果表明系统整体表现良好，基本达到了预期的设计目标。在功能完整性方面，系统的核心功能模块均已完整实现并运行稳定。四种AI模型（Kimi、Qwen、Hunyuan、DeepSeek）均实现了稳定的API集成，多轮对话上下文维护准确，对话连贯性良好，异常处理机制完善，能有效应对各类异常情况。负载均衡机制的有效实现确保了服务的稳定性，为系统的可靠运行提供了保障。

在搜索服务实现方面，Bocha和Exa搜索引擎接口集成完善，检索结果准确度达到预期目标。其中，关键词搜索准确率达95%，语义搜索准确率达90%，混合检索准确率达92%，充分满足了用户的多样化搜索需求。数据处理方面，系统展现出强大的网页数据采集能力，能够适应不同网站结构，反爬虫策略有效，确保了数据获取的稳定性和可靠性。

性能测试结果显示，系统的各项核心性能指标基本达到或超过预期目标。在响应时间方面，普通搜索请求的平均响应时间为150ms，优于200ms的目标值；AI服务调用平均延迟在400-800ms之间，符合设计要求；页面加载时间平均为80ms，提供了良好的用户体验；数据库查询平均耗时30ms，大幅优于50ms的目标值。系统并发处理能力表现突出，支持5000+并发用户访问，搜索服务峰值QPS达到3000，AI服务综合QPS达到120-150，基本满足当前的使用需求。

在系统资源利用效率方面，测试数据显示CPU平均使用率维持在30%以下，内存占用稳定，峰值未超过系统限制，网络带宽利用合理，未出现明显瓶颈。通过长期运行测试，系统展现出优秀的稳定性和可靠性。异常情况下的自动恢复机制运行良好，系统日志记录完整，便于问题定位，服务降级策略有效地确保了核心功能的可用性。在数据一致性方面，数据处理流程可靠，未出现数据丢失现象，并发访问下的数据一致性保持良好，缓存机制有效，数据同步准确。

尽管系统整体表现良好，测试过程中也发现了一些需要进一步优化的方面。在性能优化方面，AI服务的响应时间在高负载情况下仍有波动，部分复杂查询场景下的响应时间可进一步优化，系统资源使用效率还有提升空间。在功能完善方面，建议进一步增强搜索结果的相关性排序，强化AI服务的个性化定制功能，提高数据处理的自动化程度。

结束语

本文针对传统搜索引擎在处理复杂查询和多轮对话场景时的局限性，设计并实现了一种基于大语言模型的智能搜索引擎系统IntelliSearch。研究工作主要围绕系统架构设计、核心模块实现和性能优化三个方面展开，取得了以下主要研究成果：

1. 系统架构设计：提出了"关键词检索+语义理解+对话管理"的三层架构设计方案。采用Qt6框架和C++17标准构建系统框架，通过分层解耦实现了界面交互层、业务逻辑层和数据持久层的有效分离。这种架构设计不仅确保了系统的可维护性和可扩展性，也为后续功能迭代提供了良好的技术基础。
2. 核心模块实现：成功开发了AIServiceManager和SearchServiceManager两大核心组件。AIServiceManager通过统一的服务接口实现了对Kimi、Qwen、Hunyuan、DeepSeek等AI模型的集成管理，并通过动态负载均衡和故障转移机制确保了服务的稳定性。SearchServiceManager则通过整合Bocha和Exa搜索引擎，实现了高效的混合检索策略。特别是在RAG技术实现方面，通过向量数据库与传统检索的结合，构建了准确的语义相关性排序系统。
3. 性能优化：系统通过多层次的优化措施实现了优秀的性能表现。在数据管理层面，采用SQLite结合Redis的多级存储架构，实现了高效的数据访问和缓存机制。在并发处理方面，通过线程池技术和异步处理机制，显著提升了系统的并发处理能力。测试结果表明，系统在各项核心性能指标上均达到或超过预期目标，包括响应时间、并发处理能力和资源利用效率等方面。
4. 工程实践：系统实现了完整的监控和日志体系，通过spdlog框架实现了高性能的日志记录功能。同时，通过实现统一的指标收集接口，系统能够实时监控关键性能指标，并通过完善的异常处理机制确保了系统的稳定运行。
5. 实际应用效果：系统表现出优秀的搜索性能和用户体验。关键词搜索准确率达95%，语义搜索准确率达90%，混合检索准确率达92%。系统支持多用户并发访问，搜索服务峰值QPS达到3000，基本满足了实际应用需求。

本研究的创新点主要体现在以下几个方面：首先，提出了一种新型的混合检索架构，通过AI技术与传统搜索引擎的深度融合，有效提升了复杂查询场景下的搜索效果；其次，实现了完整的RAG技术框架，通过向量检索与传统检索的结合，显著提升了搜索结果的准确性；最后，在工程实现层面，通过模块化设计和标准化接口，为智能搜索系统的开发提供了可复用的技术方案。

尽管系统整体表现良好，但仍存在一些需要进一步优化的方面，如AI服务响应时间的稳定性、复杂查询场景下的性能优化等。这些问题将在后续研究中继续深入探讨和改进。总的来说，本研究为智能搜索领域提供了一个具有重要理论价值和实践意义的技术方案，其研究成果对推动搜索技术的发展具有积极意义。

致 谢

在本论文的研究和撰写过程中，我得到了许多人的支持和帮助。在此，我谨向所有给予我关心、指导和帮助的老师和同学表示衷心的感谢。

首先，感谢我的导师陈梦婷老师。陈梦婷老师在课题选择、研究方法、论文写作等方面给予了我无私的指导和支持，使我在研究过程中不断进步。陈梦婷老师渊博的学识、严谨的治学态度和认真负责的工作作风让我受益匪浅。

同时，感谢我所在的社团，校科协。在整个代码的设计实现过程中，社团的朋友们给予了我很大的帮助。从基础的环境配合，到相关模块的设计讨论，以及在具体实现中Bug分析解决，同学们的耐心解答和细致指导让我在研究中少走了许多弯路，顺利完成了各项工作。

最后，感谢所有在论文研究和撰写过程中给予我帮助的老师和同学们。你们的支持和鼓励让我在学术道路上不断前行。

参考文献

1. A. Wright, "Worldwide IDC Global DataSphere Forecast, 2024–2028: AI Everywhere, But Upsurge in Data Will Take Time," IDC, May 2024, Doc. US52076424.
2. P. D. Adamczyk and B. P. Bailey, "If not now, when? The effects of interruption at different moments within task execution," in Proc. SIGCHI Conf. Hum. Factors Comput. Syst. (CHI), Vienna, Austria, Apr. 24-29, 2004, pp. 271-278, doi: 10.1145/985692.985730.
3. 曹天甲, 程龙龙, 李世锋, 曹琉等, "基于对比学习的临床领域意图识别算法研究," 《天津大学学报（自然科学与工程技术版）》, vol. 2024, no. 08, pp. 821–827, 2024.
4. 杜亚军, "目录搜索引擎智能行为的研究及实现," 博士论文, 西南交通大学, 成都, 中国, 2005.
5. 张文元, 周世宇, 谈国新, “基于Lucene的地名数据库快速检索系统,” 计算机应用研究, 卷 34, 期 6, 2017.
6. X. Chen, X. Qiu, C. Zhu, P. Liu, and X. Huang, "Long short-term memory neural networks for Chinese word segmentation," in Proc. 2015 Conf. Empirical Methods Natural Language Process. (EMNLP), Lisbon, Portugal, 2015, pp. 1197–1206.
7. H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in Proc. 35th Int. Conf. Mach. Learn. (ICML), Stockholm, Sweden, 2018, pp. 4095–4104.
8. 张海涛, “面向深度神经网络鲁棒性的对抗训练方法研究,” 硕士学位论文, 天津大学, 天津, 中国, 2020.
9. H. Liao, H. Shen, Z. Li, C. Wang, G. Li, Y. Bie, and C. Xu, "GPT-4 enhanced multimodal grounding for autonomous driving: Leveraging cross-modal attention with large language models," Commun. Transp. Res., vol. 3, p. 100116, 2023.
10. Z. Zhao, H. Lu, Y. Huo, Y. Du, T. Yue, L. Guo, B. Wang, W. Chen, and J. Liu, "Needle in a video haystack: A scalable synthetic evaluator for video MLLMs," in Proc. Int. Conf. Learn. Represent. (ICLR), 2025, pp. 1–12. [Online]. Available: [ICLR 2025].
11. R. Gorwa and M. Veale, "Moderating model marketplaces: Platform governance puzzles for AI intermediaries," Law, Innov. Technol., vol. 16, no. 2, pp. 341–391, 2024. .
12. M. Krajewski, “Cross - platform development of the Smart Client application with Qt framework and QtQuick,” 2016.
13. C. Demetrescu, I. Finocchi, and A. Ribichini, “Reactive Imperative Programming with Dataflow Constraints,” ACM Trans. Program. Lang. Syst., vol. 37, no. 1, art. no. 3, pp. 1 - 53, Nov. 2014, doi: 10.1145/2623200.
14. X. Bi, D. Chen, G. Chen, S. Chen, D. Dai, C. Deng, et al., “DeepSeek LLM: Scaling Open - Source Language Models with Longtermism,” arXiv:2401.02954 [cs.CL], 2024, doi: 10.48550/arXiv.2401.02954.
15. P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, et al., “Retrieval - Augmented Generation for Knowledge - Intensive NLP Tasks,” arXiv:2005.11401 [cs.CL], 2020, doi: 10.48550/arXiv.2005.11401.

附录 A 软件说明书

1软件概述

IntelliSearch是一款创新型智能搜索引擎系统，它采用了最新的人工智能技术和先进的搜索架构，为用户提供高质量的信息检索服务。系统整合了国内领先的AI大语言模型服务，包括Kimi、Qwen、Hunyuan和DeepSeek，同时集成了专业的搜索引擎服务Bocha和Exa，通过创新的混合检索架构和RAG（检索增强生成）技术，实现了智能化的搜索体验。

系统的核心优势在于其独特的混合检索架构，该架构巧妙地结合了传统搜索引擎的精确性和AI大语言模型的语义理解能力。通过多层次的知识融合机制，系统能够准确理解用户的查询意图，并提供相关性更高的搜索结果。同时，系统支持自然的对话式交互，用户可以通过多轮对话逐步细化和优化搜索需求，获得更精准的答案。

1.1主要特点

1. 多模态AI服务支持：系统集成了包括Kimi、Qwen、Hunyuan和DeepSeek在内的多个国产顶尖大语言模型，每个模型都在不同方面展现出独特的优势。其中，Kimi在自然语言理解和知识问答领域表现突出，具备强大的上下文理解能力；Qwen则在多语言处理和跨领域知识整合方面有着卓越表现；Hunyuan凭借其深度的语义分析能力，特别适合处理复杂的查询请求；而DeepSeek则在专业领域的知识检索和推理方面独具特色。系统通过配备智能的服务调度机制，能够根据用户的查询类型和系统的实时负载情况，自动选择最适合的模型进行响应，从而确保为用户提供最佳的查询效果。
2. 混合检索架构：系统的核心技术特色在于其创新的混合检索架构，该架构由四个关键组件层次构成，共同协作以提供高质量的搜索服务。首先，语义理解层负责对用户的查询进行深度解析，精确提取查询意图、关键概念和语义关系；其次，检索融合层通过巧妙结合传统检索和语义检索的优势，实现了多维度的结果排序机制；再次，知识增强层运用RAG技术，将检索得到的信息与AI模型固有的知识进行有机结合；最后，结果优化层则负责对检索结果进行智能排序和去重处理，确保最终呈现给用户的信息具有最高的相关性和实用价值。这种层次分明、功能互补的架构设计，使系统能够提供更加精准和全面的搜索结果。
3. 对话式交互：系统采用了先进的对话管理机制，通过智能维护对话历史实现准确的上下文信息理解，同时具备从连续对话中精确提取和理解用户真实需求的意图识别能力。在交互方面，系统支持包括文本输入、语音输入在内的多种交互方式，为用户提供灵活便捷的使用体验。此外，系统还配备了完善的反馈机制，能够基于用户的使用反馈持续优化搜索策略和结果排序，从而不断提升系统的检索效果和用户满意度。这些功能的有机结合，使系统能够提供更加智能和人性化的搜索服务。
4. 知识增强：系统采用先进的RAG（检索增强生成）技术实现知识增强功能，该技术能够从多个数据源实时获取最新信息，并将检索到的结果与模型固有知识进行有机结合。在此基础上，系统能够基于综合信息生成准确而完整的回答，同时提供详细的信息来源引用，确保检索结果的可追溯性。这种知识增强机制不仅提升了系统的回答质量和可靠性，也为用户提供了更加全面和可信的信息服务。
5. 跨平台支持：系统采用Qt框架开发，实现了Linux、Windows、macOS全面的跨平台支持

2运行环境

表 3-1系统基础软件环境要求

|  |  |  |  |
| --- | --- | --- | --- |
| ***环境类别*** | ***配置项*** | ***版本要求*** | ***说明*** |
| *操作系统* | 跨平台支持 | - | Windows/Mac/Linux |
| *开发语言* | C++17 | C++17及以上 | 主要开发语言 |
| *GUI框架* | Qt | Qt6.0及以上 | 用户界面框架 |
| *Python环境* | Python | Python3.8以上 | 爬虫模块必须 |

表 3-2系统依赖组件要求

|  |  |  |
| --- | --- | --- |
| ***组件名称*** | ***版本要求*** | ***作用*** |
| *spdlog* | >=1.9 | 系统日志管理 |
| *nlohmann\_json* | >=3.9 | Json输出处理 |
| *curl* | >=7.0 | 网络通信 |
| *SQLite* | >=3.0 | 本地数据处理 |

表 3-3 Python爬虫模块组件要求

|  |  |  |
| --- | --- | --- |
| ***组件名称*** | ***版本要求*** | ***作用*** |
| *requests* | >=2.31.0 | HTTP请求处理 |
| *beautifulsoup4* | >=4.12.0 | HTML解析 |
| *selenium* | >=4.16.0 | 动态网页爬取 |
| *webdriver\_manager* | >=4.0.1 | WebDeiver管理 |
| *urllib3* | >=2.1.0 | URL处理 |
| *logging* | Python内置 | 日志管理 |
| *Lxml*  *aiohttp* | >=5.1.0  >=3.9.0 | XML解析  异步HTTP客户端 |

3安装说明

相关指令需在终端执行。

首先克隆项目代码：

git clone <https://github.com/your_username/IntelliSearch.git>

cd IntelliSearch

接着，配置构建环境

mkdir build && cd build

cmake ..

编译项目

make -j4

最后安装 Python 依赖

pip install -r requirements.txt

3配置文件设置

在首次运行前，需要配置以下文件：

* config/api\_keys.json：配置各AI服务和搜索引擎的API密钥
* config/system\_config.json：系统基础配置
* config/crawler\_config.json：爬虫模块配置

4功能介绍

IntelliSearch系统提供了一套完整的智能搜索解决方案，通过多个紧密协作的功能模块，为用户带来全面而强大的搜索体验。系统的核心功能围绕智能搜索、AI服务集成、数据管理和用户交互四个主要方面展开。

在智能搜索方面，系统实现了多层次的搜索策略。用户可以使用传统的关键词匹配进行快速检索，也可以通过语义搜索来理解更复杂的查询意图。特别值得一提的是，系统创新性地实现了混合检索机制，能够智能地结合关键词匹配和语义理解的优势，为用户提供更加精准的搜索结果。在交互过程中，系统支持自然的多轮对话模式，通过维护对话上下文，准确理解用户在连续对话中的查询意图，使搜索过程更加流畅和自然。

AI服务集成是系统的另一大特色。通过整合Kimi、Qwen、Hunyuan和DeepSeek等多个先进的AI模型，系统能够提供强大的自然语言理解和生成能力。特别设计的服务管理机制能够根据查询类型和负载情况，自动选择最适合的AI模型来处理用户请求。系统实现了智能的负载均衡策略，确保各个AI服务的资源得到合理分配，同时配备了完善的故障转移机制，在服务异常时能够快速切换到备用服务，保证系统的持续可用性。

在数据管理领域，系统构建了强大的数据采集和处理体系。自动化的爬虫系统能够从多个数据源持续采集信息，并通过智能的数据清洗流程确保数据质量。系统采用多级缓存机制优化数据访问性能，显著提升了搜索响应速度。同时，完整的版本控制机制确保了数据更新的可追溯性，支持数据的回滚和历史查询。通过这些功能，系统能够持续积累和优化知识库，为用户提供更加全面和准确的搜索服务。

用户界面设计注重直观性和易用性。系统采用响应式设计，能够自适应不同的屏幕尺寸和设备类型，为用户提供一致的使用体验。界面支持明暗主题切换，满足不同场景下的使用需求。在搜索过程中，系统提供实时的状态反馈，让用户清楚地了解搜索进度。搜索结果的展示采用多样化的呈现方式，包括列表视图、卡片视图等，用户可以根据个人偏好选择最适合的展示模式。此外，系统还提供了丰富的结果管理功能，支持搜索历史记录、结果收藏等操作，方便用户后续查阅和使用。通过这些功能的有机结合，IntelliSearch为用户提供了一个智能、高效、易用的搜索平台。系统不仅能够准确理解用户需求，还能通过持续的数据积累和优化，不断提升搜索服务的质量。无论是简单的信息查询，还是复杂的知识探索，系统都能够为用户提供满意的解决方案。

5使用指南

5.1 运行可执行文件

在编译文件下运行 ./IntelliSearch

5.2 首次启动配置

用户首次启动引用时，需要在配置页面进行相关的AI服务和搜索引擎服务的配置：

图形用户界面, 文本, 应用程序, 电子邮件

AI 生成的内容可能不正确。

图 1 服务配置页面

用户也可以直接在config/config.json进行相关的配置，基础的配置文件示例如下：

{

"ai\_service": "kimi",

"search\_service": "bocha",

"api\_providers": {

"kimi": {

"api\_key": "你的API密钥",

"base\_url": "https://api.moonshot.cn",

"model": "moonshot-v1-32k"

},

"bocha": {

"api\_key": "你的API密钥",

"base\_url": "https://api.bochaai.com/v1",

"max\_results": 8

}

},

"search\_settings": {

"max\_results\_per\_provider": 10,

"timeout\_ms": 5000

},

"log": {

"level": "debug",

"path": "logs/app.log"

}

}

配置说明

1. 基础设置
   1. ai\_service: 默认AI服务提供商（可选：kimi/qwen/hunyuan/deepseek）
   2. search\_service: 默认搜索引擎（可选：bocha/exa）
2. AI服务配置
   1. 每个AI服务需要配置：
   2. api\_key: API访问密钥
   3. base\_url: 服务基础URL
   4. model: 使用的模型名称
3. 搜索引擎配置

每个搜索引擎需要配置：

* 1. api\_key: API访问密钥
  2. base\_url: 服务基础URL
  3. max\_results: 单次返回最大结果数

1. 搜索设置
   1. max\_results\_per\_provider: 每个提供商返回的最大结果数
   2. timeout\_ms: 请求超时时间（毫秒）
2. 日志配置
   1. level: 日志级别（debug/info/warn/error）
   2. path: 日志文件路径

注意：实际使用时请替换配置文件中的API密钥为您自己的有效密钥。

5.3 基本操作

图形用户界面, 应用程序

AI 生成的内容可能不正确。

图 2应用查询首页面图

用户在主界面的输入框输入需要查询的问题后，系统将自动调取后端逻辑进行执行搜索请求，并返回结果。

图形用户界面, 文本, 应用程序, 电子邮件

AI 生成的内容可能不正确。

图 3搜索结果示意图

此外，应用支持多轮对话

图形用户界面, 文本, 应用程序, 电子邮件

AI 生成的内容可能不正确。

图 4多轮对话示意图

6爬虫使用指南

6.1 功能介绍

爬虫模块作为系统的子模块，可以直接在系统中指定链接进行爬取，并同时与搜索请求发送至AI进行处理，针对指定网页进行分析得到最终的结果。

图形用户界面, 应用程序

AI 生成的内容可能不正确。

图5搜索链接输入框

系统会调用WebDriver进行动态爬取，如下图：

图形用户界面, 文本, 应用程序, 电子邮件

AI 生成的内容可能不正确。

图 6爬虫执行过程图

6.2 配置说明

安装依赖包

pip install requests beautifulsoup4 selenium webdriver\_manager urllib3 aiohttp lxml

创建配置文件，配置文件命名为：crawler\_config.json

{

"max\_depth": 2, // 最大爬取深度

"max\_pages": 100, // 最大爬取页面数

"request\_delay": 1.0, // 请求延迟(秒)

"follow\_external\_links": false, // 是否跟随外部链接

"use\_dynamic\_crawling": false, // 是否启用动态爬取

"page\_load\_timeout": 30, // 页面加载超时时间

"allowed\_domains": [ // 允许爬取的域名列表

"example.com"

],

"url\_filters": [ // URL过滤规则

".\*\\.pdf$",

".\*\\.zip$"

],

"user\_agent": "IntelliSearch Python Crawler/1.0",

"output\_dir": "crawl\_results" // 结果输出目录

}

6.3 使用方法

爬虫模块支持多种调用方法：

命令行模式：

* 基础爬虫：python crawler.py https://example.com --depth 2 --pages 100 --delay 1.0 --output-dir results
* 动态爬虫（支持JavaScript渲染）：python dynamic\_crawler.py https://example.com --depth 2 --pages 100 --delay 1.0

参数说明：

--depth, -d：爬取深度

--pages, -p：最大页面数

--delay：请求间隔

--output-dir, -o：输出目录

--config, -c：配置文件路径

代码调用方式

from crawler import Crawler

# 创建爬虫实例

crawler = Crawler()

# 配置爬虫参数

crawler.config.max\_depth = 2

crawler.config.max\_pages = 100

crawler.config.request\_delay = 1.0

crawler.config.allowed\_domains = ['example.com']

# 开始爬取

crawler.start\_crawling(['https://example.com'])

6.4 输出结果

爬虫结果将以 Json 形式保存，结构如下：

{

"url": "https://example.com",

"title": "页面标题",

"content": "页面主要内容...",

"links": ["https://example.com/page1"],

"metadata": {

"description": "页面描述",

"page\_size\_bytes": 1256,

"status\_code": 200,

"content\_type": "text/html",

"encoding": "utf-8"

},

"timestamp": "2024-04-21T18:36:38.268"

}



图 7爬虫结果图