上海电力大学

# 《Python 程序设计与全栈开发》大作业


题　　目：基于本地大模型的 GitHub 智能推荐助手

学　　号：20231607

姓　　名：刘栩年

院　　系：计算机科学与技术学院

专业年级：计算机科学与技术专业 23 级

2025 年 12 月 29 日

# 目录

# 第一章、引言

在软件开发过程中，开发者经常需要寻找合适的开源库或框架来解决特定问题。传统的 GitHub 搜索功能虽然强大，但往往需要精确的关键词，且难以直接获取项目的横向对比和适用性分析。而通用的大语言模型（LLM）虽然具备自然语言理解能力，但由于训练数据的滞后性，往往无法提供最新的项目信息，甚至会产生编造不存在的项目链接（“机器幻觉”）的问题。

本项目“CodeWhisper AI”旨在结合大语言模型的语义理解能力与 GitHub 实时数据的准确性，构建一个基于检索增强生成（RAG）技术的智能推荐助手。系统通过本地部署的 Qwen2.5 模型理解用户意图，实时调用 GitHub API 获取真实数据，并经过清洗、翻译和重组，最终以极简的现代化 UI 为用户提供准确、可信且带有中文注解的开源项目推荐。

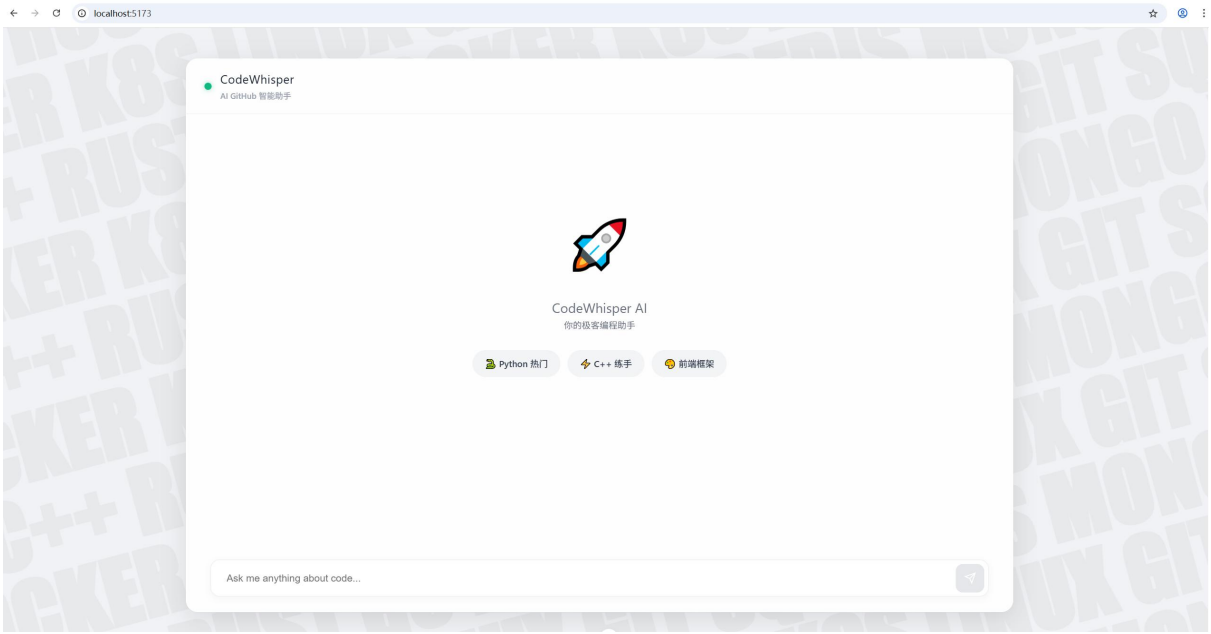



图 1-1：CodeWhisper AI 系统运行主界面截图

# 第二章、需求分析

## 2.1 项目背景与目标

随着开源生态的爆发，GitHub 上的项目数量呈指数级增长。初学者和资深开发者都面临着“选择困难症”。本项目旨在实现一个能够通过自然语言对话，例如“推荐一个 Python 爬虫库”，自动检索、筛选并总结 GitHub 项目的智能体，且所有模型计算在本地完成，保障隐私并降低成本。

## 2.2 系统需求

智能对话能力：能够进行日常闲聊，并能精准识别用户的“搜索”意图。

实时检索能力：能够根据自然语言提取关键词，从 GitHub 获取实时数据（Star 数、描述、链接）。

抗幻觉机制：严禁 AI 编造不存在的 URL，必须返回真实可点击的项目链接。

响应速度：后端需采用异步处理，确保在高并发请求下界面不卡顿。

## 2.3 用户需求

用户希望在一个界面美观、交互流畅的网页中，通过简单的中文描述获取高质量的开源项目推荐。推荐结果应包含项目名称、Star 数、简介以及中文翻译和直达链接。

## 2.4 技术需求

后端：Python (FastAPI) + Asyncio

前端：Vue 3 + Vite + CSS3 动画

模型服务：Ollama (运行 Qwen2.5:7b)

数据源：GitHub REST API

## 2.5 功能细化

意图分类：区分闲聊与搜索请求。

关键词提取与清洗：从口语化输入中提取技术关键词。

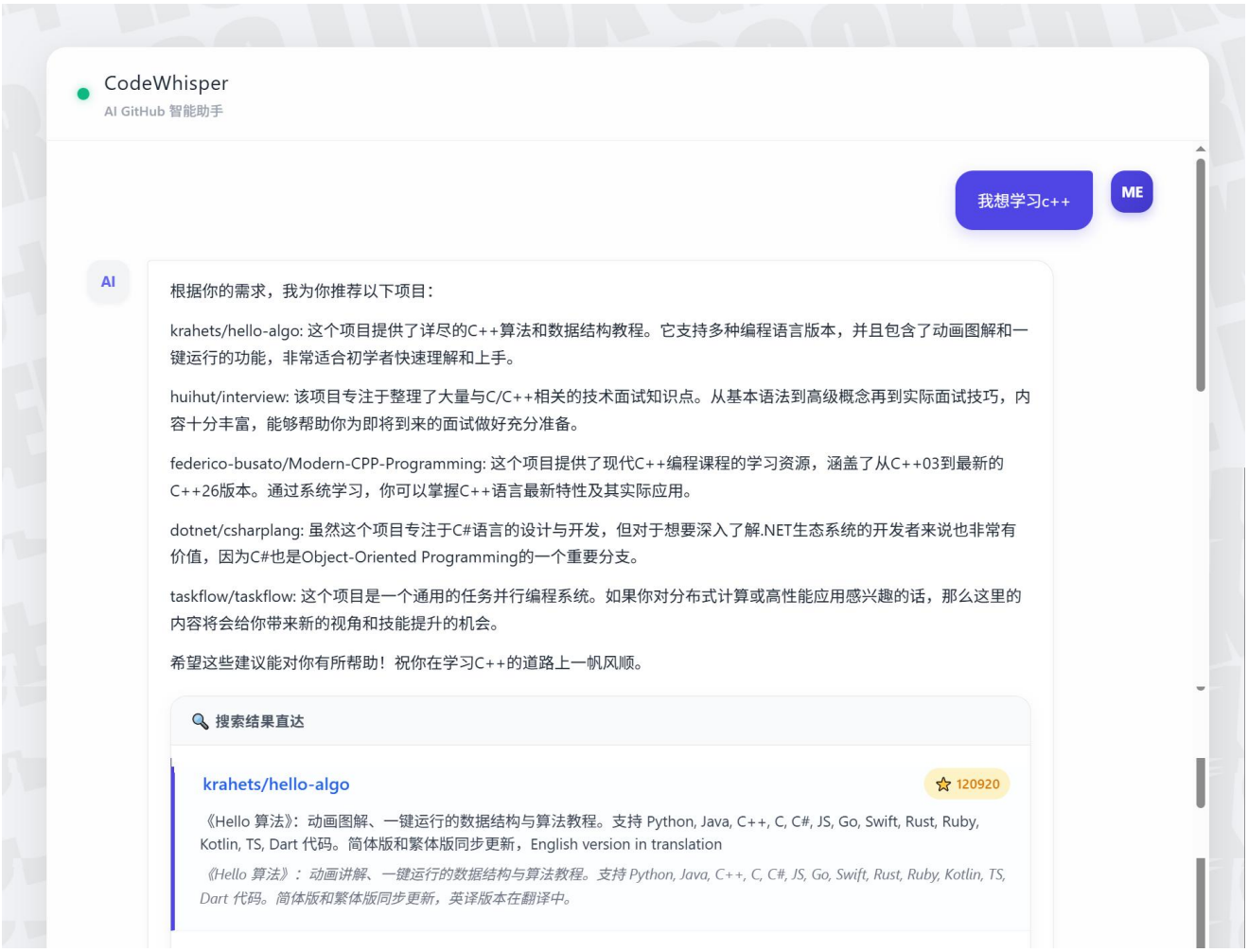
中英双语对照：利用 AI 将英文项目描述自动翻译为中文。

打字机特效：前端实现流式输出视觉效果。

```
2025-12-29 23:24:40,484 - INFO - >>> 进入 RAG 搜索模式
2025-12-29 23:24:49,024 - INFO - 正在请求GitHub API, Query: Python 开源项目 推荐
2025-12-29 23:24:51,282 - INFO - 搜索成功，获取到 6 个项目
2025-12-29 23:26:15,501 - INFO - 正在翻译项目描述...
INFO:     127.0.0.1:51684 - "POST /chat HTTP/1.1" 200 OK
```

## 2.6 预期效果

系统界面具备极客风格，交互如同与真人对话。当用户询问技术资源时，系统能迅速给出精准的项目列表卡片，点击即可跳转 GitHub 仓库。



# 第三章、系统设计

## 3.1 功能模块划分

系统主要分为三层架构：

表现层（Frontend）：负责用户交互、Markdown/HTML 渲染、打字机特效及动态背景展示。

业务逻辑层（Backend）：API 服务模块：处理 HTTP 请求，管理上下文历史。Agent 核心模块：包含意图识别、关键词提取、RAG 流程控制。翻译模块：对 GitHub 返回的英文元数据进行批量翻译。

数据/模型层：Ollama 服务：提供 LLM 推理能力。GitHub API：提供外部实时知识库。

## 3.2 系统数据库设计及实现

考虑到本项目的核心在于实时检索与生成，且注重轻量化部署，本项目采用了内存级会话管理而非传统的某些关系型数据库。

会话上下文：使用 Python 列表在内存中临时存储最近 4 轮对话，用于让 AI 理解上下文。

数据模型：定义了 ChatRequest（输入消息、历史）和 ChatResponse（纯文本回复、HTML 搜索卡片）的数据结构，确保前后端数据交互的规范性。

## 3.3 系统技术架构

采用前后端分离架构：

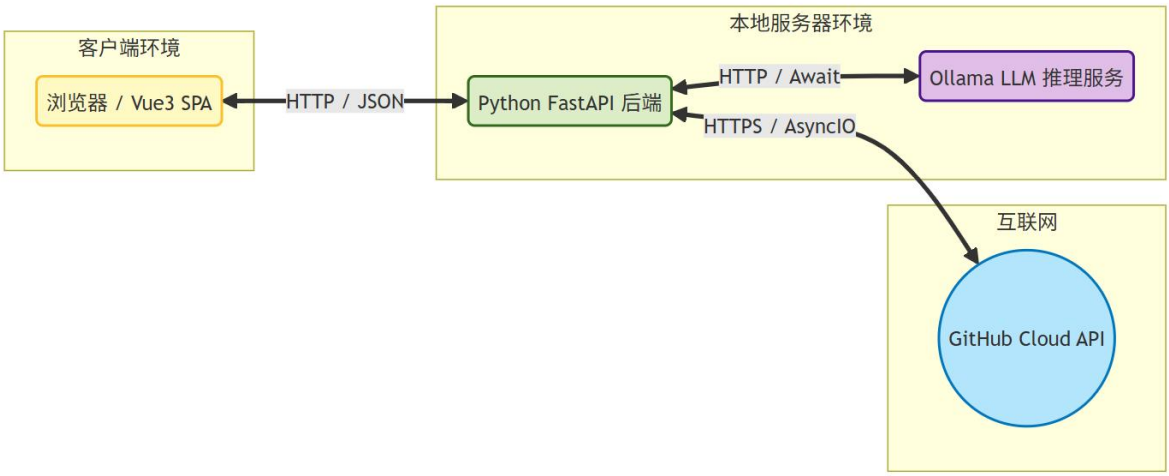
前端：Vue 3 Composition API，使用 Axios 进行网络通信，Marked.js 处理文本渲染。

后端：FastAPI 框架，利用 aiohttp 实现全链路异步操作，大幅提升了 I

/O 密集型任务，例如同时请求 LLM 和 GitHub 的效率。

RAG 链路：User Input -> Keyword Extraction (LLM) -> GitHub Search API -> Context Injection -> Answer Generation (LLM) -> Translation -> Response。

**图 3-1：系统技术架构拓扑图**




说明：展示了各组件之间的网络连接协议与数据流向。

# 第四章、系统核心模块设计与实现

## 4.1 RAG 检索增强生成模块

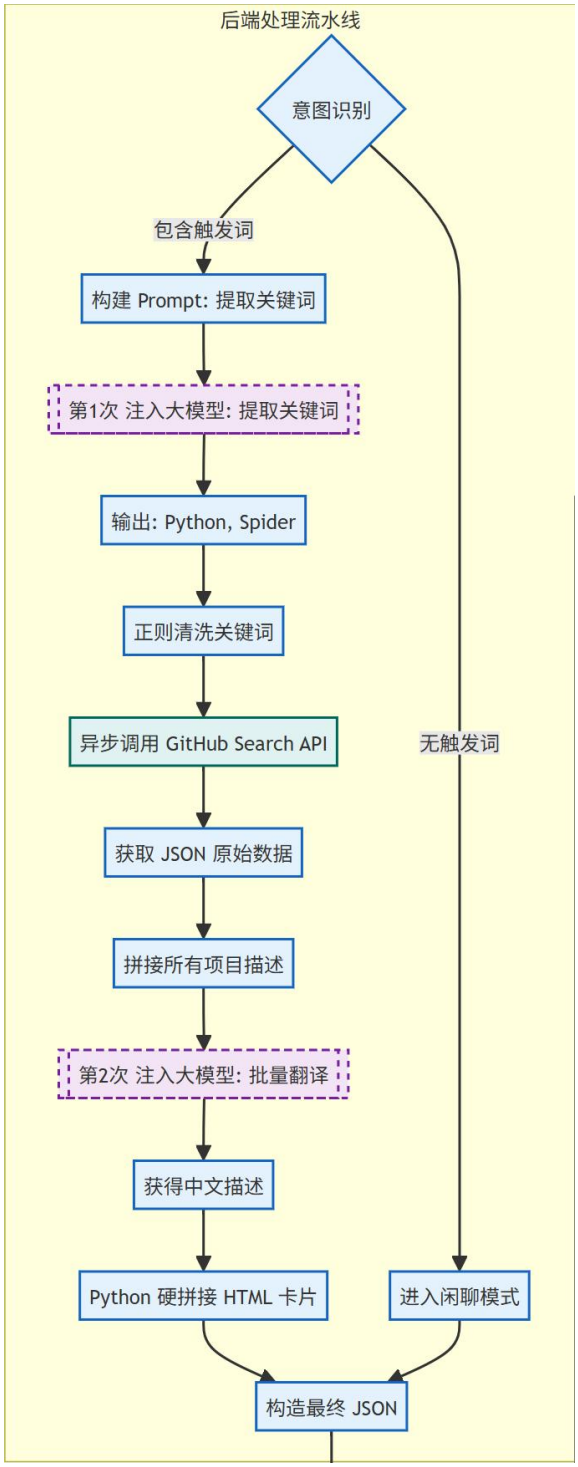
这是本系统的核心。为了解决大模型容易编造链接的问题，我设计了“数据与生成分离”的策略：

关键词提取：使用专门的 Prompt 让模型提取技术栈关键词如 "Python, Spider"，并通过正则表达式 clean_model_keywords 进行清洗，去除无关字符。

双路返回机制：search_github_repos 函数不仅返回给 AI 阅读的文本摘要，还返回包含真实 URL 的原始数据列表 raw_items。

硬拼接策略：为了彻底根治前端显示 undefined 或 [object Object] 的问题，系统不再依赖 AI 生成 Markdown 链接，而是由 Python 后端直接生成标准的 HTML 卡片，并强制拼接到回复末尾。这保证了所有链接 100% 真实有效。

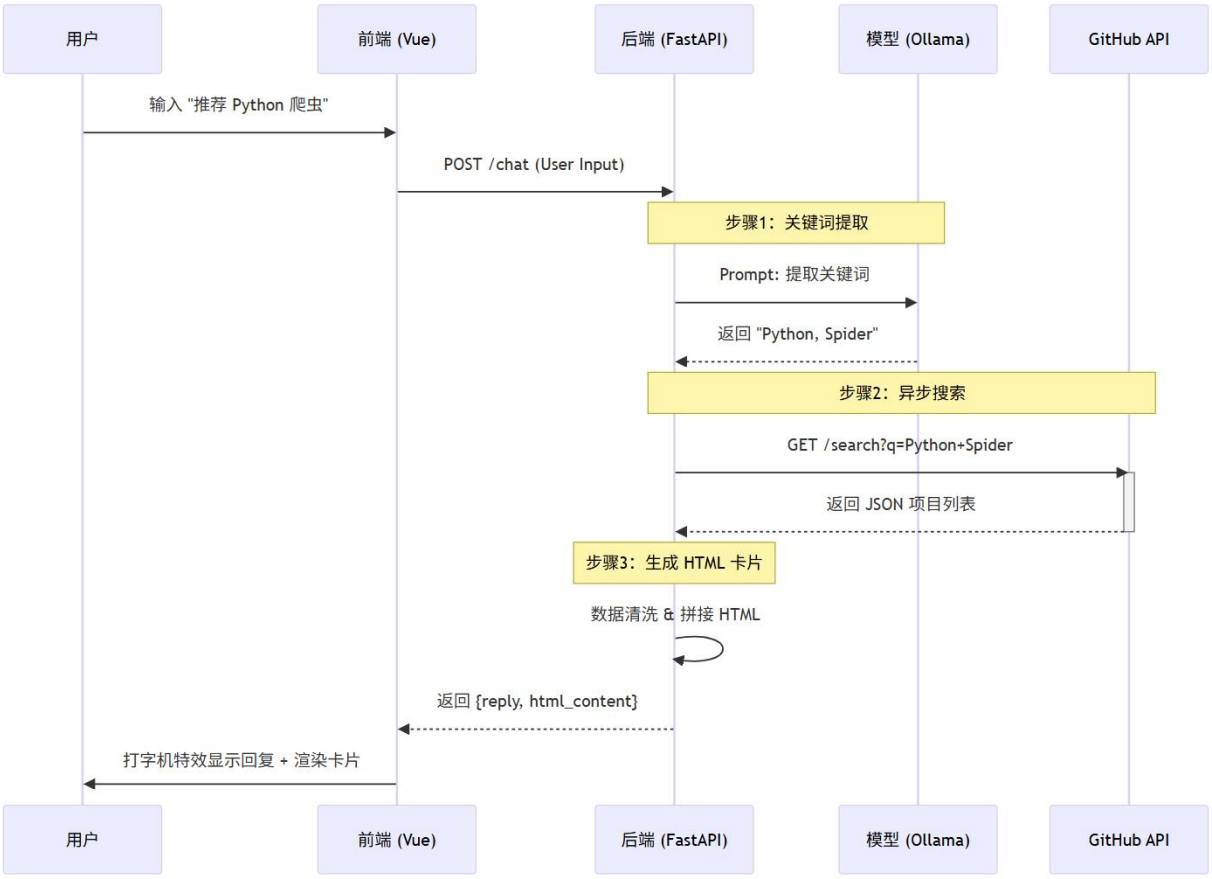
当用户输入“推荐 python 爬虫”，后端运行流程如右图：



## 4.2 异步批量翻译模块

GitHub API 返回的描述通常是英文。为了提升用户体验，系统在获取到搜索结果后，会将所有项目的描述打包，通过一次 LLM 调用进行批量翻译，然后再拆分回各个项目条目中。这既保证了速度，又实现了中英双语对照展示。

**图 4-1：异步搜索流程时序图**




说明：展示了核心业务的时序交互，体现了异步调用的过程。

## 4.3 动态视觉前端

动态背景：利用 CSS3 transform 和 @keyframes 实现了倾斜滚动的“代码墙”效果。通过 JS 动态生成 30 行技术词汇（如 PYTHON, DOCKER），并让它们交错滚动，营造出极强的科技感。

流式打字机特效：实现了 typeWriter 逻辑。为了防止打字效果破坏 HTML

结构，系统将“纯文本回复”与“HTML 搜索卡片”分离传输。前端先逐字打印文本，待打印完成后，再以淡入动画（Fade-in）的形式展示底部的搜索结果卡片。

毛玻璃 UI：聊天窗口采用 backdrop-filter: blur 实现了类似 iOS 的毛玻璃效果，配合阴影和圆角设计，界面简洁现代。

# 第五章、系统总结

## 5.1 已实现功能

全栈开发：完成了从 Vue3 前端到 FastAPI 后端的完整链路打通。

本地 RAG 智能体：成功集成了 Ollama Qwen2.5 与 GitHub API，实现了基于真实数据的问答。

工程化容错：通过后端生成 HTML 的方式，完美解决了 Markdown 解析器在处理复杂链接时的兼容性问题，解决了开发过程中遇到的 undefined 链接 Bug。

高颜值交互：实现了动态背景、打字机输出、中英对照卡片等高级交互功能。

## 5.2 系统值得改进

多轮对话深度：目前仅保留了最近 4 轮对话，未来可以引入 Redis 或向量数据库来实现长期的记忆存储。

流式传输：目前使用的是模拟流式，即前端收到完整包后逐字显示，未来可以升级为 Server-Sent Events 或 WebSocket，实现后端生成一个字、前端显示一个字的极致低延迟体验。

更多数据源：目前仅支持 GitHub，未来可以接入 StackOverflow 或 HuggingFace，打造全能的技术搜索助手。

## 5.3 心得体会

通过本次大作业，我深刻理解了 RAG（检索增强生成）技术在解决大模型“幻觉”问题上的巨大价值。

在开发过程中，我遇到了前端 Markdown 解析器将加粗链接识别为对象的棘手 Bug。通过排查，我学会了不应过分依赖 AI 的格式化输出，而应通过代码逻辑，例如用 Python 拼接 HTML 来兜底关键数据。

此外，在 UI 设计上，我学会了如何利用 CSS 动画和 JS 逻辑构建具有视觉冲击力的界面，这让我明白全栈开发不仅是功能的实现，更是用户体验的打磨。