

天津大学



知识图谱问答系统使用说明

编者：3023244322 蒋茜，3023244328 马佳一

资料提供：3023244327 邵玺冉，3023244338 张婉毓

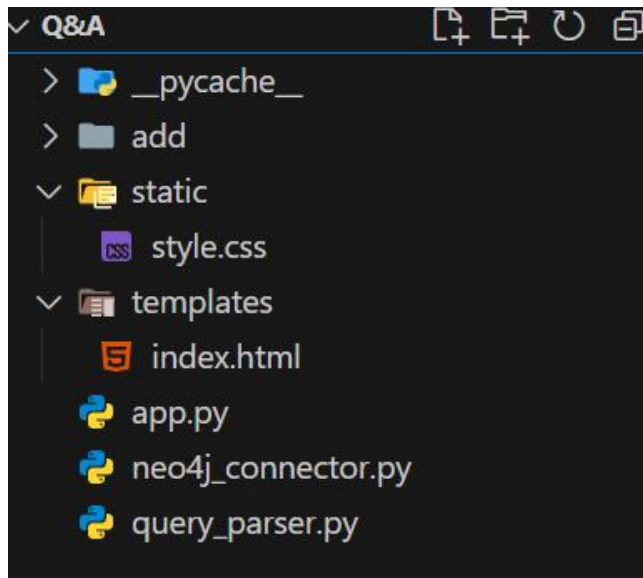
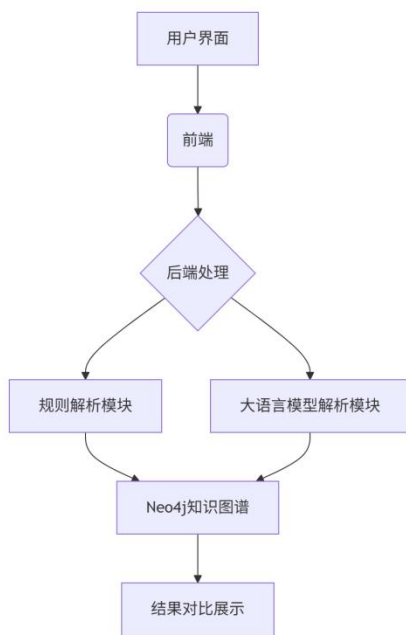
（姓名不分先后，按首字母排序）

2025 年 6 月 30 日

1. 系统概述

这是一个基于 neo4j 知识图谱的智能问答系统，结合了传统的规则解析和大语言模型 API 连接，能够对领域知识问题进行智能回答，并展示**两种不同方式**的不同结果，下面以认知科学相关知识库进行实例展示。

2. 系统组成

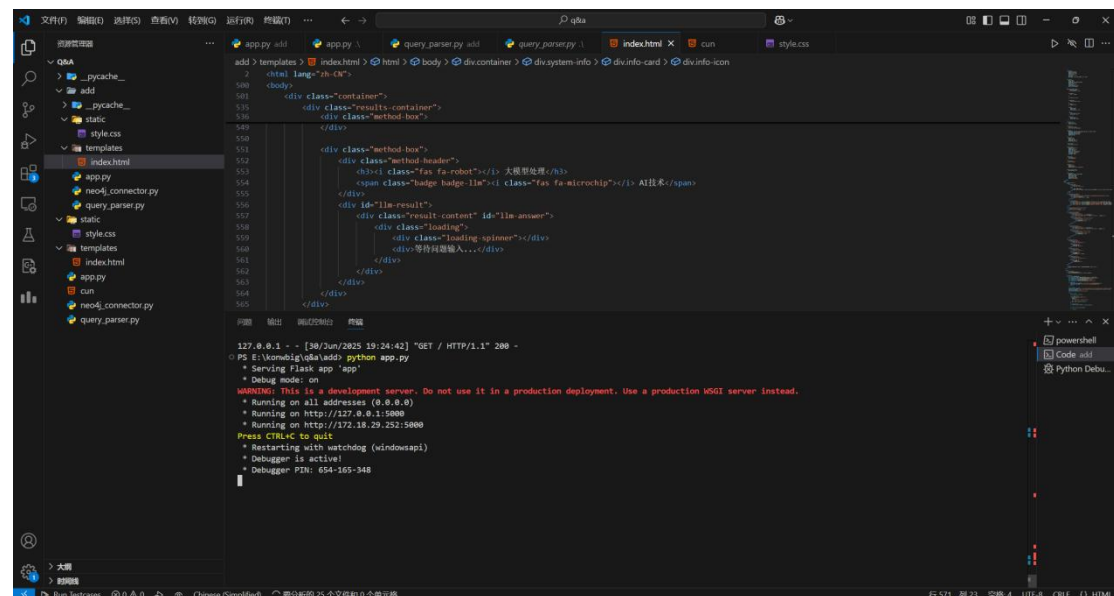


3. 使用前准备

运行环境

- Python 3.8+
- Flask 框架
- Neo4j 数据库(3.5+)
- 前端基础：HTML/CSS/JavaScript

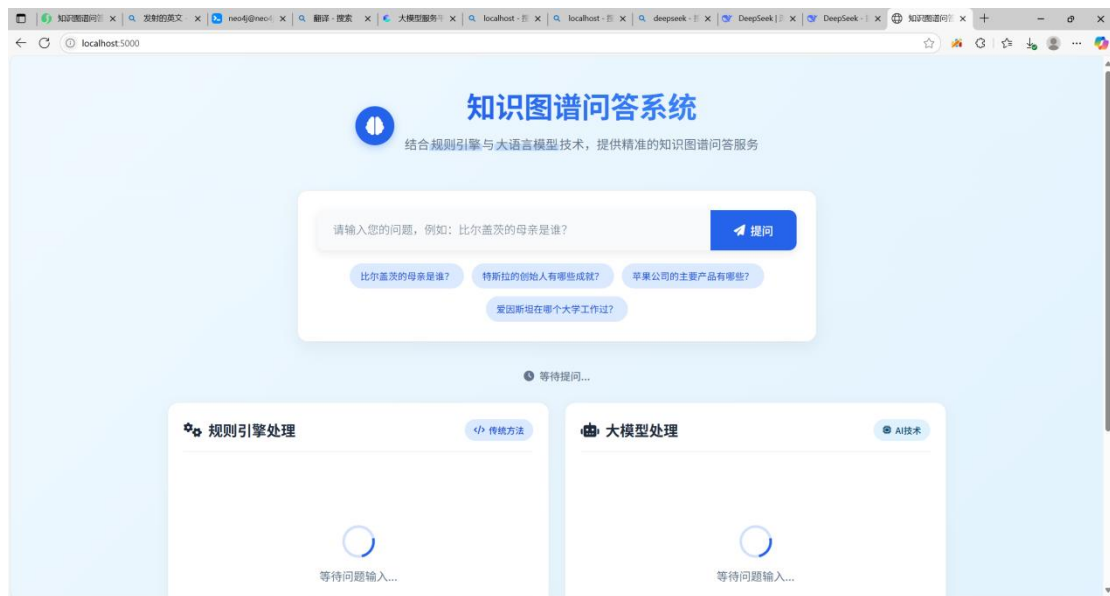
安装步骤



4. 使用流程

步骤 1：访问系统

在浏览器中访问：`http://localhost:5000`



步骤 2：输入问题

在输入框中输入自然语言问题，例如：

- "比尔盖茨的母亲是谁？"
- "认知科学相关的技术有什么？"

步骤 3：点击“提问”按钮

系统会同时启动两种方法进行处理：

- 左侧：原有规则处理方法
- 右侧：大语言模型处理方法

步骤 4：查看结果

结果包含：

- 生成的 Cypher 查询语句（可扩展查看完整语句）

- 查询执行结果
- 处理响应时间

5. 系统结果解读



5.1 结果区域介绍

Cypher 查询框：展示系统生成的查询语句

结果区域：显示查询返回的结果

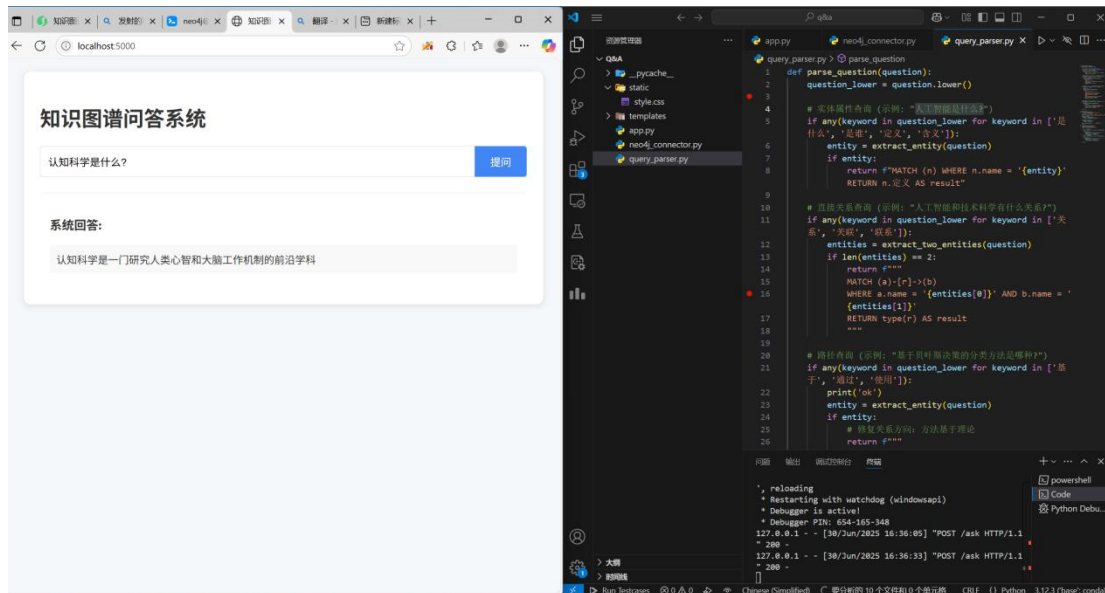
时间统计：显示整个处理耗时

5.2 不同结果类型解读

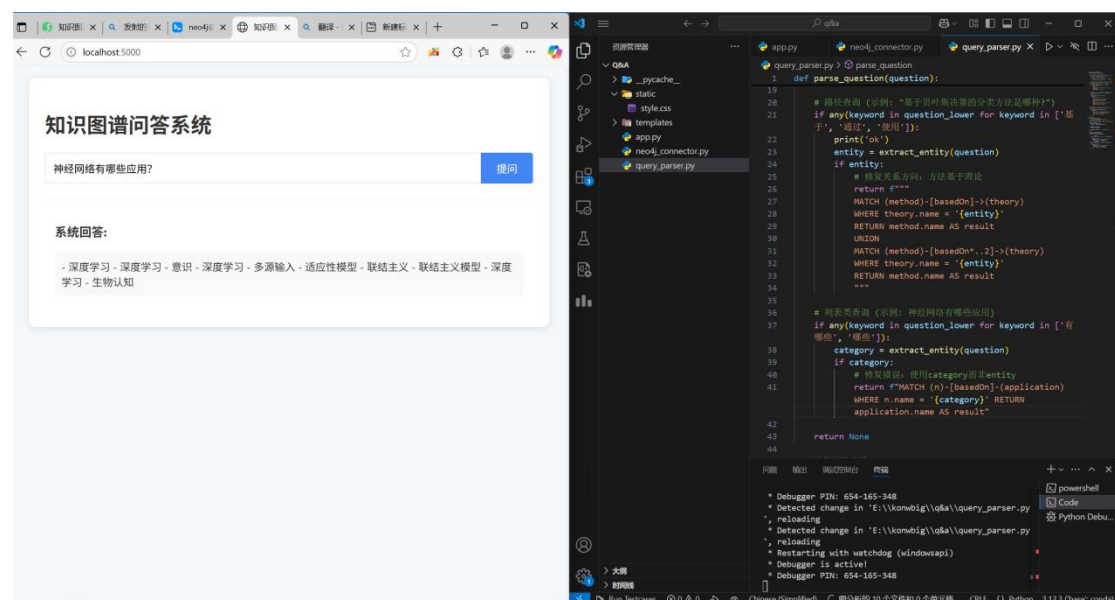
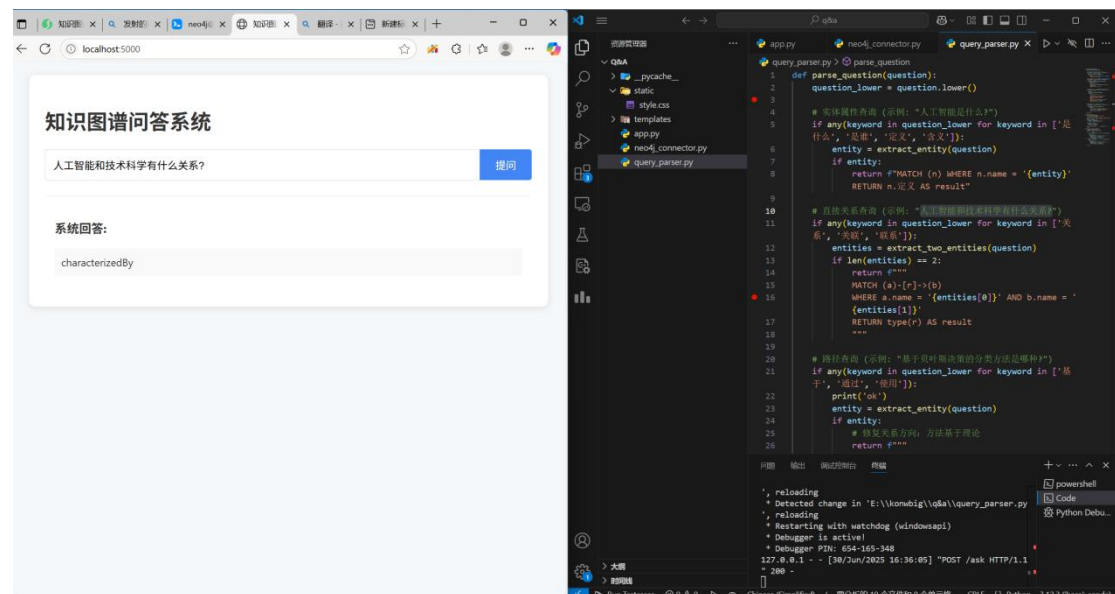
标准查询结果：

(下面展示的为 1.0 版本的标准查询)

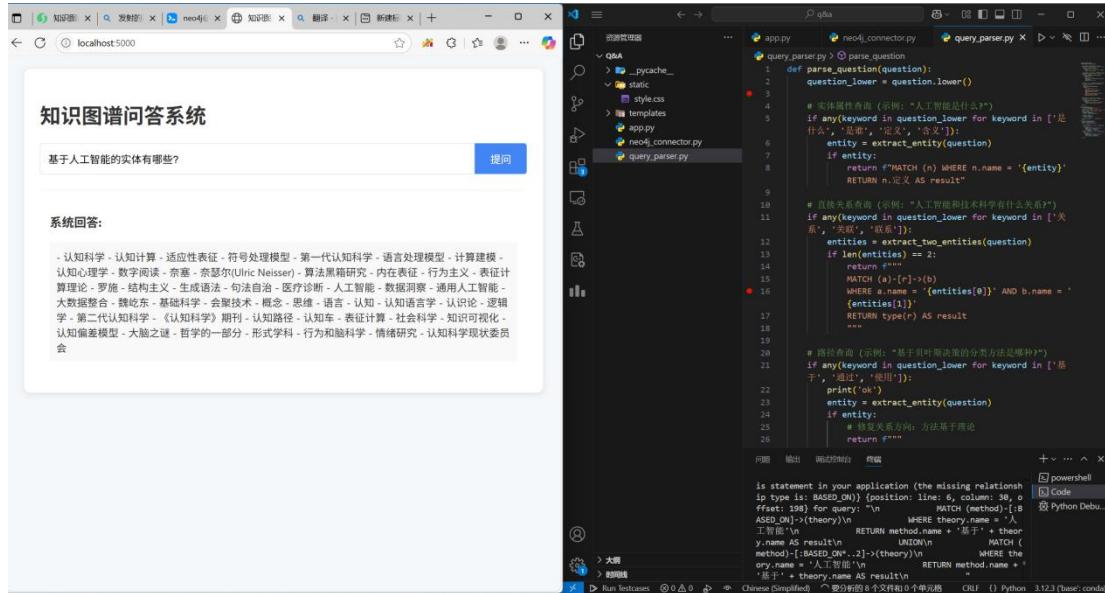
1. 实体属性查询



2. 直接关系查询



3. 简单路径查询



知识图谱问答系统

基于人工智能的实体有哪些?

提问

系统回答:

- 认知科学 - 认知计算 - 适应性表征 - 符号处理模型 - 第一代认知科学 - 语言处理模型 - 计算建模 - 认知心理学 - 数字阅读 - 奈塞 - 奈瑟尔(Ulric Neisser) - 算法黑箱研究 - 内在表征 - 行为主义 - 表征计算理论 - 罗德 - 结构主义 - 生成语法 - 句法自治 - 医疗诊断 - 人工智能 - 数据洞察 - 通用人工智能 - 大数据整合 - 魏屹东 - 基础科学 - 会聚技术 - 概念 - 思维 - 语言 - 认知 - 认知语言学 - 认识论 - 逻辑学 - 第二代认知科学 - 《认知科学》期刊 - 认知路径 - 认知车 - 表征计算 - 社会科学 - 知识可视化 - 认知偏差模型 - 大脑之谜 - 哲学的一部分 - 形式学科 - 行为和脑科学 - 情绪研究 - 认知科学现状委员会

```
def parse_question(question):
    question_lower = question.lower()

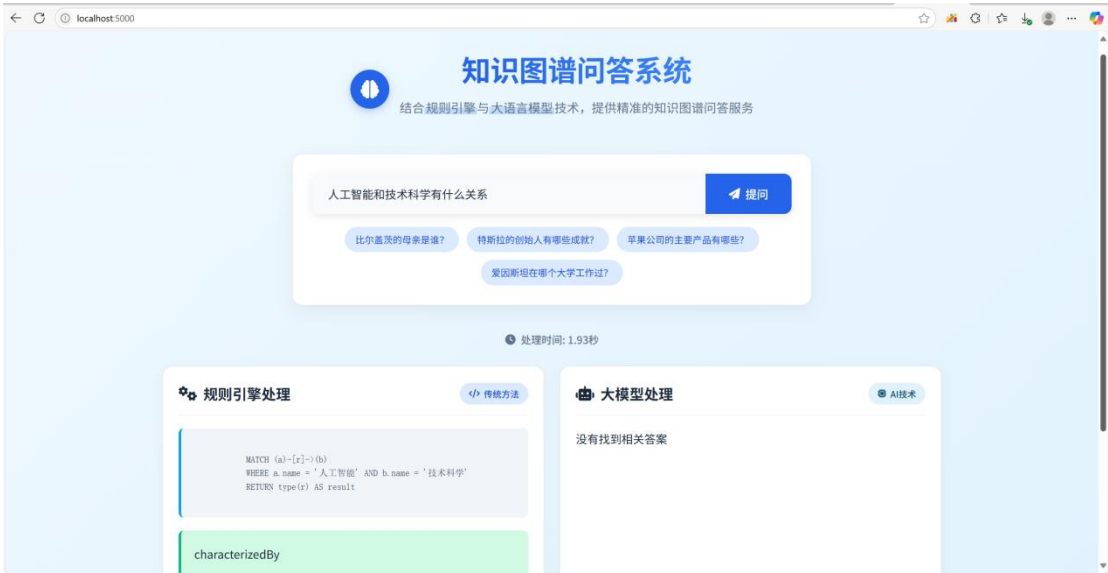
    # 实体属性查询 (示例: "人工智能是什么?")
    if any(keyword in question_lower for keyword in ['是什么', '定义', '含义']):
        entity = extract_entity(question)
        if entity:
            return f"MATCH (n) WHERE n.name = '{entity}'\nRETURN n.定义 AS result"

    # 直接关系查询 (示例: "人工智能和技术科学有什么关系?")
    if any(keyword in question_lower for keyword in ['关系', '关联', '联系']):
        entities = extract_two_entities(question)
        if len(entities) == 2:
            return f""
            MATCH (a)-[r]->(b)
            WHERE a.name = '{entities[0]}' AND b.name = '{entities[1]}'
            RETURN type(r) AS result
            ""

    # 路径查询 (示例: "基于贝叶斯决策的分类方法是哪种?")
    if any(keyword in question_lower for keyword in ['基于', '通过', '利用']):
        print('ok')
        entity = extract_entity(question)
        if entity:
            # 基于关系查询: 方法基于理论
            return f""
```

is statement in your application (the missing relationship type is: BASED_ON) (position: line: 6, column: 38, offset: 198) for query: "\n MATCH (method)-[:BASED_ON]->(theory)\n WHERE theory.name = '人工智能'\n RETURN method.name + '基于' + theory.name AS result\n UNION\n MATCH (method)-[:BASED_ON]->(theory)\n WHERE theory.name = '人工智能'\n RETURN method.name + '基于' + theory.name AS result\n"

这些已经全部移植到 2.0 版本:



但是如图可见, 大模型的 api 调用并不稳定, 很容易出现调用失败

5.3 结果对比分析



如图所示，大模型可以处理规则引擎所不能处理的一些问题
二者之间的对比：

问题类型	规则处理方法	大模型处理方法
基础查询	快速但有限	容易调用失败
复杂关系查询	可能失败	成功率更高
新颖问题	必然失败	尝试推理

6. 系统亮点

6.1 双引擎并行处理架构

```
@app.route('/ask', methods=['POST'])
def ask_question():
    try:
        data = request.get_json()
        question = data.get('question', '')

        if not question:
            return jsonify({'error': '问题不能为空'})

        # 创建两个线程并行处理
        start_time = time.time()
        original_thread = QueryThread(["original", question])
        llm_thread = QueryThread("llm", question)

        original_thread.start()
        llm_thread.start()

        # 等待线程完成
        original_thread.join()
        llm_thread.join()

        end_time = time.time()
        process_time = round((end_time - start_time) * 1000) # 毫秒

        # 构建结果对象
        response = {
            "processing_time": f"{process_time}ms",
            "original": format_query_result(original_thread, question),
            "llm": format_query_result(llm_thread, question)
        }

        return jsonify(response)
```

6.2 大模型交互技术

```

24 def generate_cypher_with_api(question):
25     """调用大模型API生成Cypher查询"""
26     try:
27         # 构建提示词
28         relationships_str = "\n".join(RELATIONSHIPS[:20]) # 只显示部分避免过长
29         prompt = f"""
30         你是一个Neo4j知识图谱查询生成器。用户的问题是:{question}
31
32         请根据问题生成合适的Cypher查询语句,并遵守以下规则:
33         1. 只能使用以下关系类型:
34         {relationships_str}
35         (完整列表包含{len(RELATIONSHIPS)}种关系)
36         2. 只返回Cypher语句,不要包含其他任何内容
37         3. 所有节点都有'name'属性用于匹配
38         4. 结果使用'result'作为返回值的别名
39
40         现在请为以下问题生成Cypher查询:
41         {question}
42         """
43
44         # 调用API
45         headers = {
46             "Authorization": f"Bearer {API_KEY}",
47             "Content-Type": "application/json"
48         }
49
50         payload = {
51             "model": "qwen-turbo",
52             "input": {
53                 "messages": [
54                     {"role": "system", "content": "你是一个专业的Cypher查询生成器"},
55                     {"role": "user", "content": prompt}
56                 ]
57             },
58             "parameters": {
59                 "temperature": 0.1,
60                 "top_p": 0.9
61             }
62         }
63
64         response = requests.post(API_URL, headers=headers, json=payload)
65         response.raise_for_status()
66
67         # 提取生成的Cypher语句
68         data = response.json()
69         cypher = data["output"]["text"].strip()

```

7. 项目总结

过程关键点:

知识图谱建模: 设计合适的图数据库模式

查询语言转换: 从自然语言到 Cypher 的转换

API 集成: 大语言模型服务的调用和结果处理

项目亮点:

- 创新的双处理引擎架构
 - 实时的处理时间对比
 - 清晰的查询过程可视化
 - 健壮的错误处理机制
-

8. 常见问题解答

Q: 系统返回"未生成 Cypher 查询"怎么办?

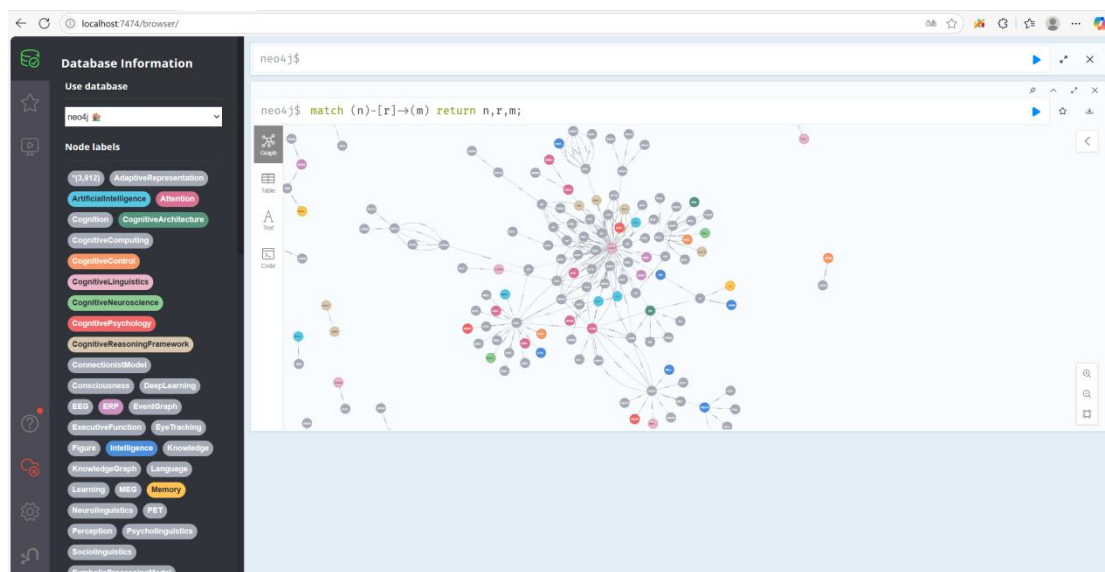
A: 尝试简化问题结构, 或使用更标准的专业术语提问

Q: 为什么大模型方法有时比规则方法更慢?

A: 大模型需要额外 API 调用时间, 但处理复杂问题有优势

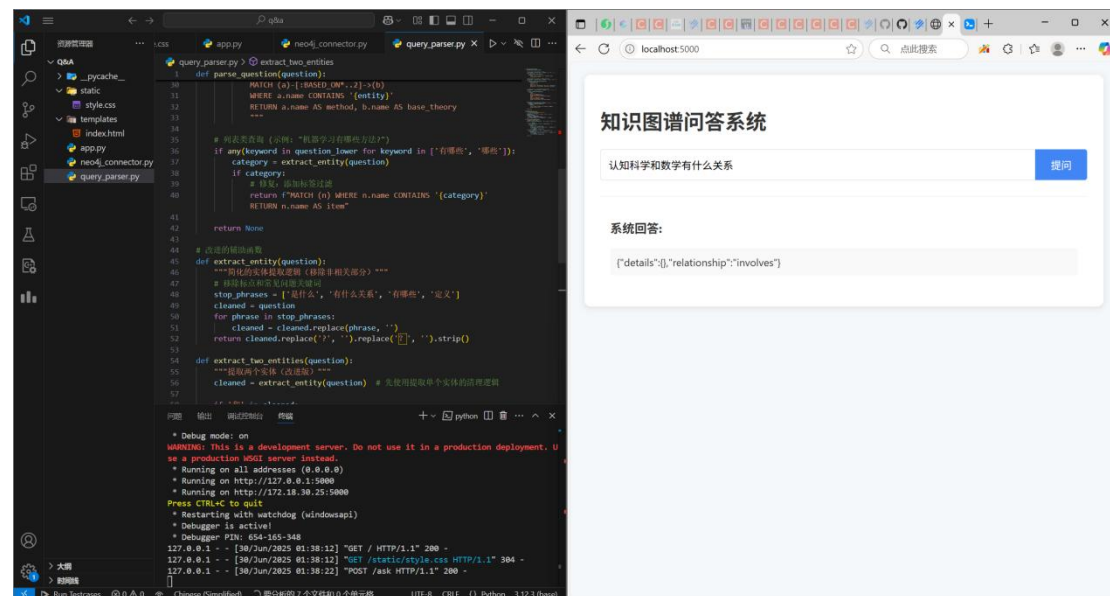
Q: 如何扩展系统的知识库?

A: 通过 Neo4j 添加新的节点和关系即可

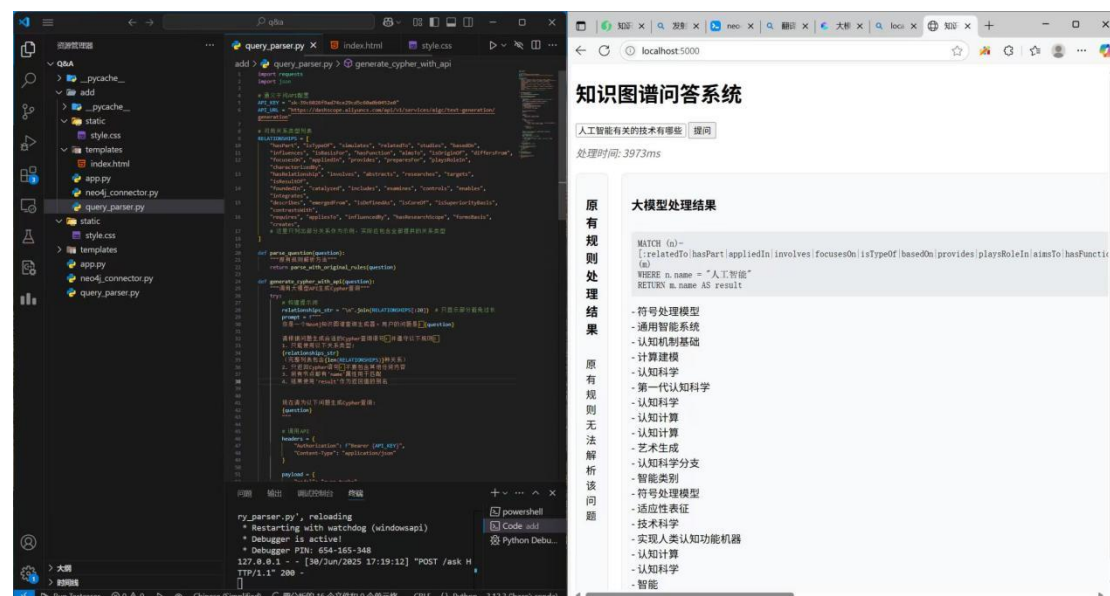


9. 结语

本知识图谱问答系统作为结课项目，完整实现了从前端到后端、从规则系统到人工智能的多种技术栈整合。我们经过了初步的 1.0 版本，只实现了简单的模版匹配：



再进行与大模型 api 的结合，实现了问答系统的创新性提升,2.0 版本诞生：



最后我们进行了页面升级，更好地实现了与用户的交互：



3.0 最终版就此出现

通过这个过程，我们对于知识工程的基本架构和流程有了更为清晰的掌握，并且动手实践了很多之前没有尝试过的技术栈，真的很累，但是也收获颇多。