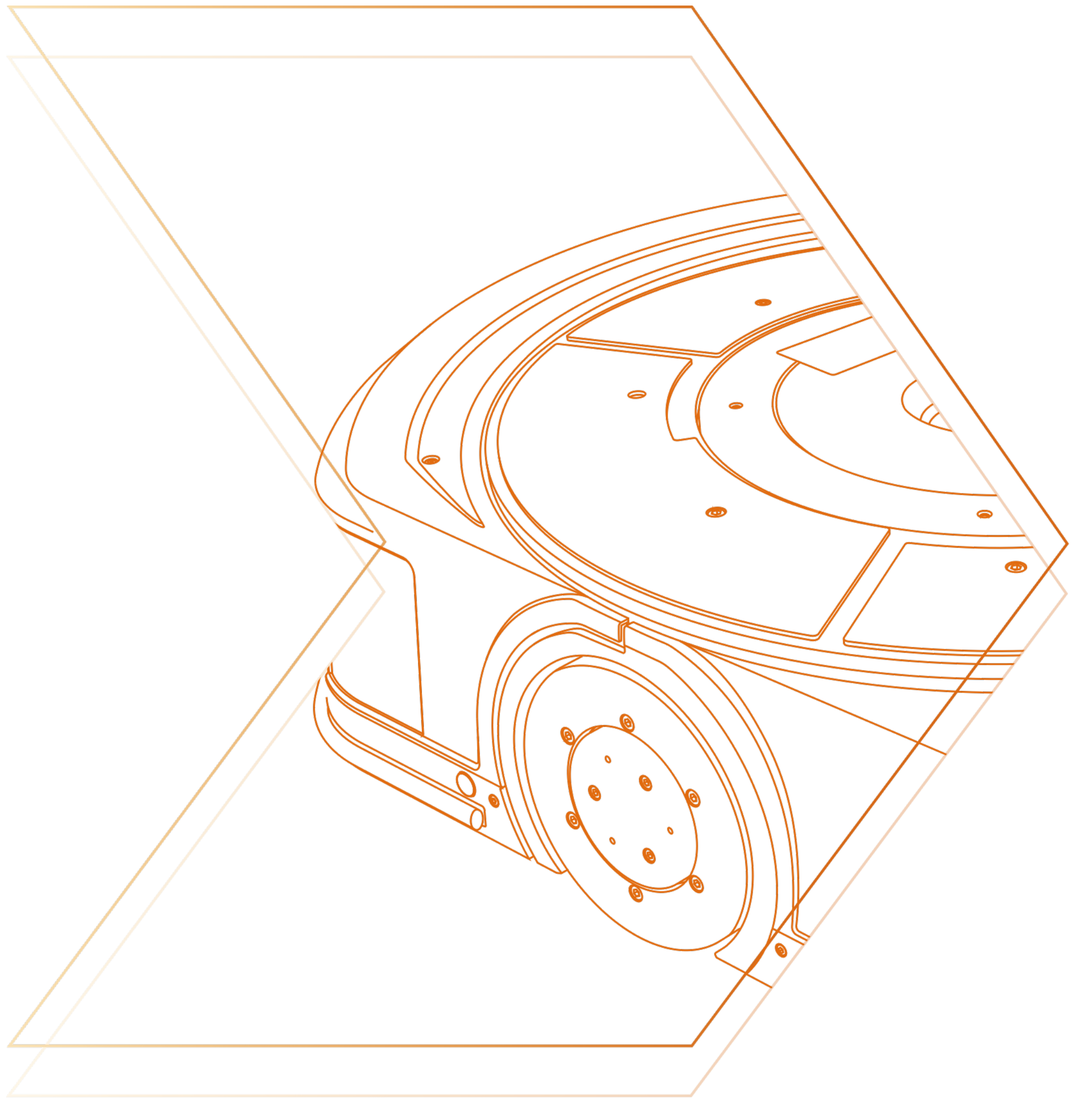
Quicktron 快仓

新人双周报—3~4周

Agentic Workflow 在代码知识库方面的 Demo构建与应用



汇报人：刘晓枫

汇报时间：2025/6/30

目录

1 现实问题

Quicktron

新成员上手难

新成员入职后，面对复杂的业务场景、各类非体系化的文档、版本极多的代码文件，往往需要花费很长时间才能熟悉业务、上手开发。

知识孤岛化

代码中核心复杂逻辑可能由前人掌握，人员流动带来巨大风险。

文档与代码脱节

当前文档与代码较为杂乱，难以明晰其关系。且代码文件内部注释存在缺失、杂乱现象，可能会产生误导。

Code Review低效

当需要code review时，需要花费大量时间理解代码逻辑，制约研发效率。

目录

2 工作流构建 | 总体了解

Quicktron

什么是Agentic Workflow?

一个流程： 自主规划、执行的自动化工作流；

多个“专家”Agent的协作： 一个团队，包含多个角色不同的AI，共同完成复杂的总目标。

它的输入输出会是什么?

输入： 类似大模型对话，一条模糊指令；

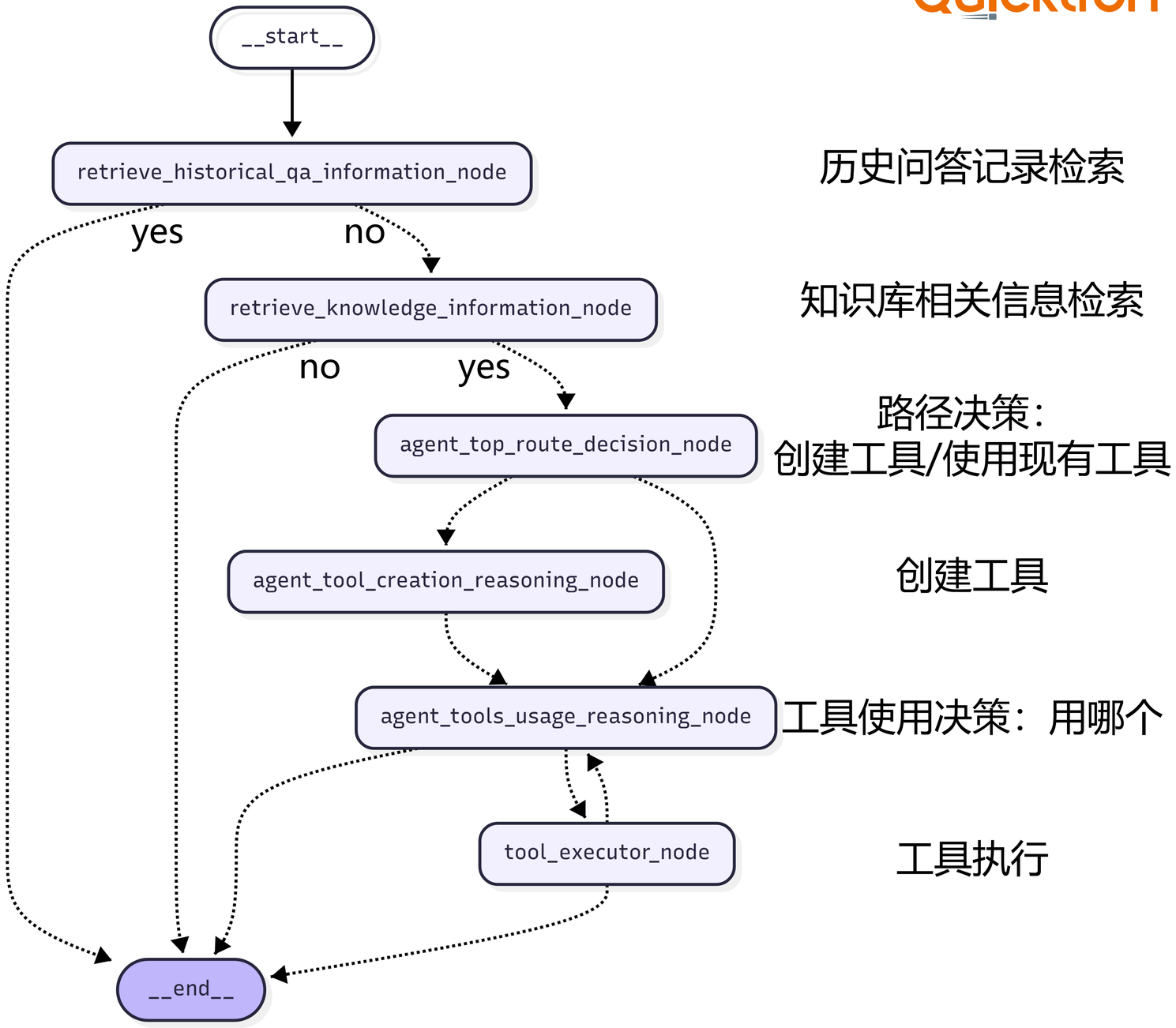
输出： 一个具体的、可交付的最终成果。

那Agentic Workflow与直接和大模型对话的区别是什么?

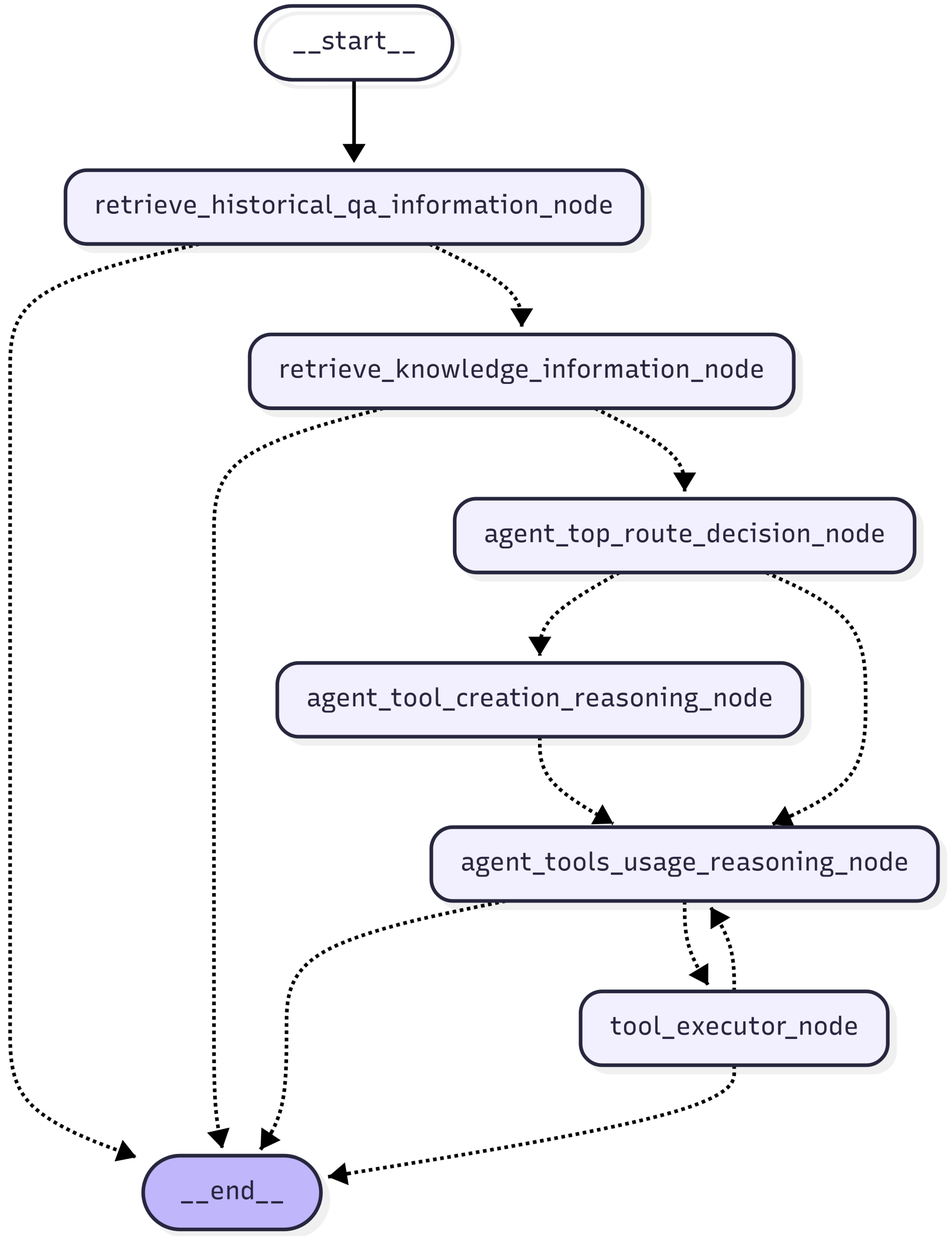
	直接对话大模型	Agentic Workflow
工作模式	被动反应式：你问我答	**主动执行式：**设定目标自动执行
角色定位	单一领域的专家，无法完成复杂任务	**多个领域**专家组成的项目**团队**
处理能力	擅长处理单一、明确的任务 例如：“帮我解释这段50行代码的功能”	擅长处理**复杂、多步骤**的**系统性**工程 例如：“分析整个代码库，找出所有复杂函数并添加注释”
自主性	低 (需要人类持续引导和追问)	**高** (能够自主分解任务、调用工具、创建工具)

2 工作流构建 | 架构与模块设计

- **数据库**
  - **历史问答：** 问题重复可直接回复；
  - **Chroma向量知识库：** 目前包含鸿鹄调整巷道、任务分配鸿鹄各版本代码+1个算法说明PPT+1个算法说明word；无相关知识则结束工作流。
- **工具（数据获取、网络搜索、代码执行、人机交互、版本控制等）**
  - **随工作流开发工具（3个）：** 代码理解（LLM）、增加注释（LLM）、问答总结（LLM）；
  - **工作流中agent新建的通过测试的其他**



2 工作流构建 | 构建流程



明确工作流的**作用与定位**（代码开发、数据分析、等）

确定大致的工作流**流程**

梳理并设计所需工具、数据库、节点、边、大模型

数据库、知识库构建

LangGraph代码开发

状态：定义agentstate的各参数，如input、上次回复等；

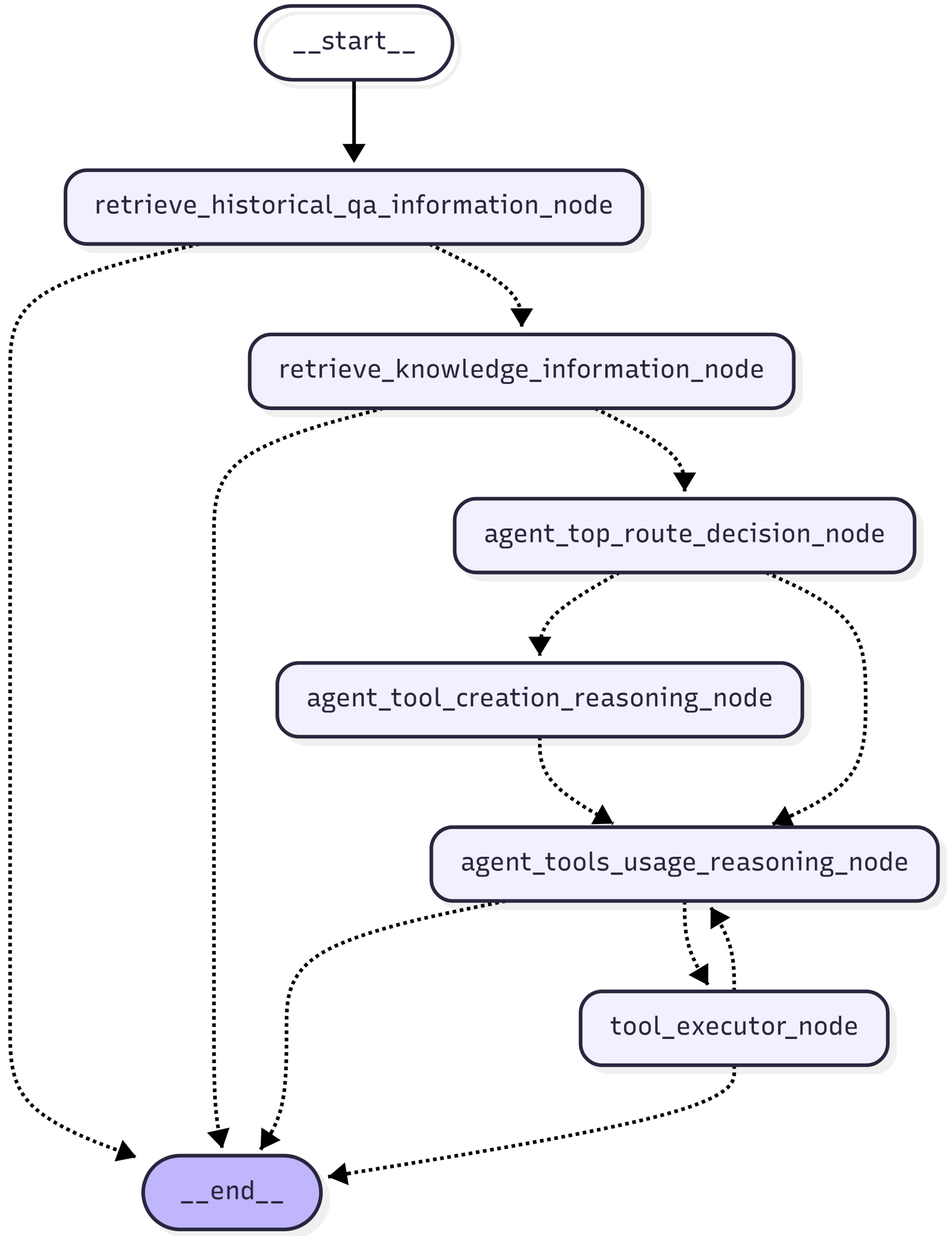
节点：可以是一段代码、可以是一个agent，返回更新的state

边：根据state，确定跳到哪个节点，返回节点名

工具：可以让大模型来写工具，返回自己构造的结构化文本

Agent/工具中用到的**提示词**：给个框架，大模型优化

2 工作流构建 | 避坑



➢ 已经总结到语雀了

- 分享记录
  - AI Lab 相关平台
  - 新人双周报
    - 社招
    - 校招
    - 实习
      - 闫以墨
      - 冯国昊
      - 刘晓枫
        - **避坑**
        - 20250616 - 含视...
    - AILAB主线会议

避坑

1. langchain/langgraph中的newAPI调用相关

1.1 chat model调用方式（例）：

Python | ⓡ运行代码 | 复制代码

```python
from langchain_openai import ChatOpenAI
llm = ChatOpenAI(
    api_key = os.getenv("NEW_API_KEY"),
    base_url = os.getenv("NEW_API_URL"),
    model = "deepseek/deepseek-r1-0528",
    temperature = 0
)
```

1.2 embedding model调用方式（例）：

Python | ⓡ运行代码 | 复制代码

```python
from langchain_ollama import OllamaEmbeddings
embeddings = OllamaEmbeddings(
    base_url = "http://ailab.flashhold.com:11434",
    model = "qwen3-8b-embedding:f16"
)
```

尤其注意url地址：①非13001端口；②末尾不带/v1；③末尾不带/v1/embeddings

参考：http://www.enmalvi.com/2024/06/05/langchain-rag-2/#Ollama_zaiLangChain_zhong_de_shi_yong

qwen3-8b-embedding:f16：这应该是个本地模型，newAPI上显示的已用额度应该不会产生实际消耗。

构建向量知识库（如chroma）时，把API的额度调高一点，参考：WorkBinAgvAdjustServiceImpl.java，共124个版本，约100~120W token。

2. 向量数据库相关

2.1 不同embedding模型可能具有不同的维度，数据库不兼容。

例：modelA创建了一个数据库，modelB无法在其基础上新增内容，需要统一同一个数据库采用的model维度。

2.2 尽量不要整个文件作为一条数据导入，个人感觉还是split一下比较好；否则可能出现超过模型支持的上下文窗口长度从而被截断，正则匹配失败的问题。

2 工作流构建 | 核心特点

Quicktron

基石：结构化的对话艺术

Agentic Workflow底层运行原理：文字游戏；

提示词是操作指令，输出是结构化的文本，从中匹配得出关键信息

进阶：提示词的自我进化

提示词直接影响了大模型回复的质量；

用大模型撬动大模型的能力，先用大模型优化提示词，再用于工作流中使大模型精确回复

架构：多领域的专家团队

工作流可以包含多个专精不同的大模型；

如LLM1代码能力强，让它去加注释；

LLM2说话通俗易懂，让它去总结……

优势：可观测性与调试

可以在每一个node、每一个edge打印信息；

清晰直观地了解到工作流的执行路径，打印关键信息，方便后续调试修改

目录

3 应用成效 | 采用Deepseek官方API——250623

Query: 结合代码说明：如何筛选可调整巷道的料箱车？Prompt增加返回在哪个代码段

```
Query: 结合代码说明：如何筛选可调整巷道的料箱车？
Reply: 根据代码逻辑，筛选可调整巷道的料箱车主要包括以下步骤：

1. **过滤未绑定巷道的料箱车**：
   - 通过 `agv -> StringUtils.isBlank(agv.getBindingRoadWayCode())` 筛选未绑定到任何巷道的料箱车，以处理冷启动或被推出巷道的情况。

2. **排除已调整到其他巷道的车**：
   - 使用 `!busyMoveAgvs.containsKey(agv.getWorkBinAgvCode())` 确保料箱车未被标记为正在调整到其他巷道。

3. **检查料箱车的可调整性**：
   - 通过 `WorkBinAgv::canAdjust` 方法确认料箱车是否可以被调整。

4. **记录可调整的料箱车**：
   - 使用 `toAdjustAgvCodes.add(agv.getWorkBinAgvCode())` 记录可调整的料箱车编号，并在日志中输出。

5. **筛选候选巷道**：
   - 根据巷道的任务需求（如“料箱AGV缺失度”）、故障状态、任务类型等条件，筛选出需要调整料箱车的巷道候选集合。

6. **距离和优先级计算**：
   - 计算料箱车到目标巷道的距离，并结合巷道的任务优先级（如强制优先任务数）生成成本矩阵，使用匈牙利算法进行最优匹配。

7. **处理特殊情况**：
   - 如果料箱车所在巷道与目标巷道的缺失度差小于阈值，取消匹配；如果巷道内有多台车，优先选择离目标巷道入口点最近的车。

通过这些步骤，系统能够高效地筛选出适合调整的料箱车，并确保调整后的分配满足任务需求和系统性能优化。
```

工作流：

只需提问，执行完后给出回复，不超过5分钟（取决于各大模型回复时间）

传统方式：

问题 → 查找相关文档是否有相关内容 → 不全面，找相关代码 → 结合文档内容，理解代码 → 总结

3 应用成效 | 采用Deepseek官方API——250623

Quicktron

Query: 对料箱车调整巷道中的处理KM分配结果部分代码增加注释

```
    private void putResult(String toAdjustAgvCode, String toAdjustRoadWayCode, Map<String, String> result, Map<String, String>
        Pair<Boolean, String> ret = new Pair<>( left: false,  right: "");
        // 有车要调去所选车所在巷道，则把那台车调去所选巷道
        if (result.containsValue(originalAgvRoadWayMap.get(toAdjustAgvCode))) {
            result.entrySet().stream() Stream<Entry<…>>
                    .filter( Entry<String, String> en -> en.getValue().equals(originalAgvRoadWayMap.get(toAdjustAgvCode)))
                    .findFirst() Optional<Entry<…>>
                    .ifPresent( Entry<String, String> en -> {
                        ret.setKey(true);
                        en.setValue(toAdjustRoadWayCode);
                    });
        }
        // 所选巷道有车要调出去，所选车要调进所选巷道，则辆车互换，该车调去所选巷道的车要去的巷道
        result.entrySet().stream() Stream<Entry<…>>
                .filter( Entry<String, String> en -> toAdjustRoadWayCode.equals(originalAgvRoadWayMap.get(en.getKey())))
                .findFirst() Optional<Entry<…>>
                .ifPresent( Entry<String, String> en -> {
                    ret.setKey(true);
                    ret.setValue(en.getKey());
                    result.put(toAdjustAgvCode, en.getValue());
                    result.put(en.getKey(), toAdjustRoadWayCode);
                });

        if (!ret.getLeft()) {
            result.put(toAdjustAgvCode, toAdjustRoadWayCode);
            agvRoadWayMap.put(toAdjustAgvCode, toAdjustRoadWayCode);
        } else {
            agvRoadWayMap.putAll(result);
            if (StringUtils.isNotBlank(ret.getRight())) {
                result.remove(ret.getRight());
            }
        }
    }
}
```

```
/**
 * 处理特殊情况的AGV巷道分配结果
 *
 * @param toAdjustAgvCode 需要调整的AGV编号
 * @param toAdjustRoadWayCode 目标巷道编号
 * @param result KM算法分配结果
 * @param originalAgvRoadWayMap 原始AGV巷道映射
 * @param agvRoadWayMap 最终AGV巷道映射
 *
 * 处理逻辑：
 * 1. 如果有其他AGV要调到当前AGV的原巷道，则让该AGV直接调到目标巷道
 * 2. 如果目标巷道有AGV要调出，则与当前AGV交换目标巷道
 * 3. 否则直接分配当前AGV到目标巷道
 */
private void putResult(String toAdjustAgvCode, String toAdjustRoadWayCode, Map<String, String> result, Map<String, String>
    Pair<Boolean, String> ret = new Pair<>(false, "");

    // 情况1：有其他AGV要调到当前AGV的原巷道
    if (result.containsValue(originalAgvRoadWayMap.get(toAdjustAgvCode))) {
        result.entrySet().stream()
                .filter(en -> en.getValue().equals(originalAgvRoadWayMap.get(toAdjustAgvCode)))
                .findFirst()
                .ifPresent(en -> {
                    ret.setKey(true);
                    en.setValue(toAdjustRoadWayCode);  // 让该AGV直接调到目标巷道
                });
    }

    // 情况2：目标巷道有AGV要调出
    result.entrySet().stream()
            .filter(en -> toAdjustRoadWayCode.equals(originalAgvRoadWayMap.get(en.getKey())))
            .findFirst()
            .ifPresent(en -> {
                ret.setKey(true);
                ret.setValue(en.getKey());
                // 交换两个AGV的目标巷道
                result.put(toAdjustAgvCode, en.getValue());
                result.put(en.getKey(), toAdjustRoadWayCode);
            });

    // 根据处理结果更新最终映射
    if (!ret.getLeft()) {
        // 无特殊情况：直接分配
        result.put(toAdjustAgvCode, toAdjustRoadWayCode);
        agvRoadWayMap.put(toAdjustAgvCode, toAdjustRoadWayCode);
    } else {
        // 有特殊情况：更新所有结果
        agvRoadWayMap.putAll(result);
        if (StringUtils.isNotBlank(ret.getRight())) {
            result.remove(ret.getRight());
        }
```

传统方式： 问题 → 查找相关文档是否有相关内容 → 不全面，找相关代码 → 结合文档内容，理解代码 → 增加注释

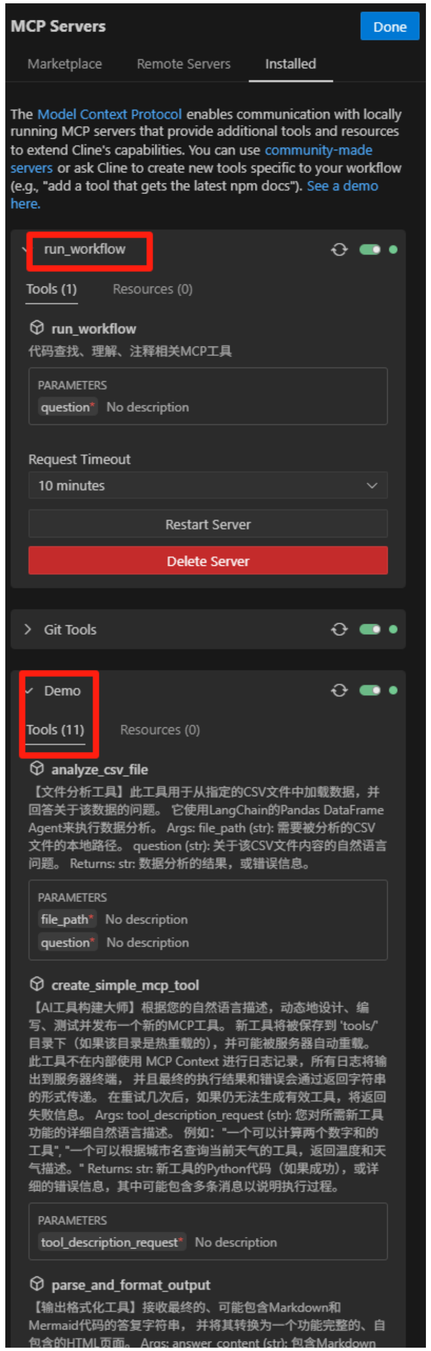
后续根据调试过程中出现的各种bug、问题，对workflow进行了优化，提高了其在多场景下的稳定性。

D:\OneDrive\qc\ny_python\langg

3 应用成效 | MCP封装与（TODO）Dify部署

MCP封装：已跑通，也能使用国昊的MCP工具

Dify部署：之后需要把代码放服务器上



Quicktron

目录

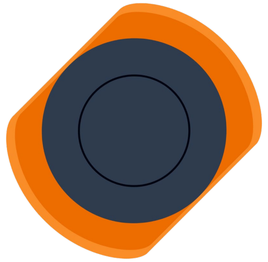
4 未来展望 | 工作流优化

	本工具TODO	完成状态
短期	1.打包部署Dify；不盲目优化，根据使用结果优化工作流。	√(?)
	2.实现历史版本检索前，先用只包含最新版代码的知识库（模型、条数、长度见readme）	√
	3.各类BUG，增强代码鲁棒性	√
中期	1.选用多种不同领域专精大模型	—
	2.增加注释后的代码保存（his_comm？）→ **MCP**直接merge?	—
	3.代码依据文件名和版本号查找（现在可能找到古早版本、其他代码文件）→ **MCP**	—
	4.知识库检索tool → 结合上条**MCP**查找，改langgraph图结构，删除知识库检索	—
	5.历史问答检索幻觉貌似比较严重	√
	6.准确率覆盖率指标与测试	—
长期	1.代码版本对比与问题定位工具→ 结合**MCP**，workflow新增tool	—
	2.知识库拓展（所有算法？部分算法？）→ **MCP**	—
	3.创建工具后，若测试执行报错，则重新修正。	—
其他	1.工具调用的toolcall丢失：现正则匹配，可①改提示词②括号平衡解析替代正则匹配	—

4 未来展望 | 赋能团队业务

Quicktron

新人双周报—3~4周



谢谢！

汇报人：刘晓枫

汇报时间：2025/6/30