

基于MCP的多链代码生成智能体Agent开发课题总结

课题概述

本课题设计并实现了一个基于 MCP(Model Context Protocol)的多链代码生成智能体系统，结合 AutoGen 框架中的 GraphFlow 结构化执行和 MagenticOne 双账本风格调度，实现了从需求分析到代码生成、测试、重构、质量扫描的自动化开发工作流。

GitHub 地址

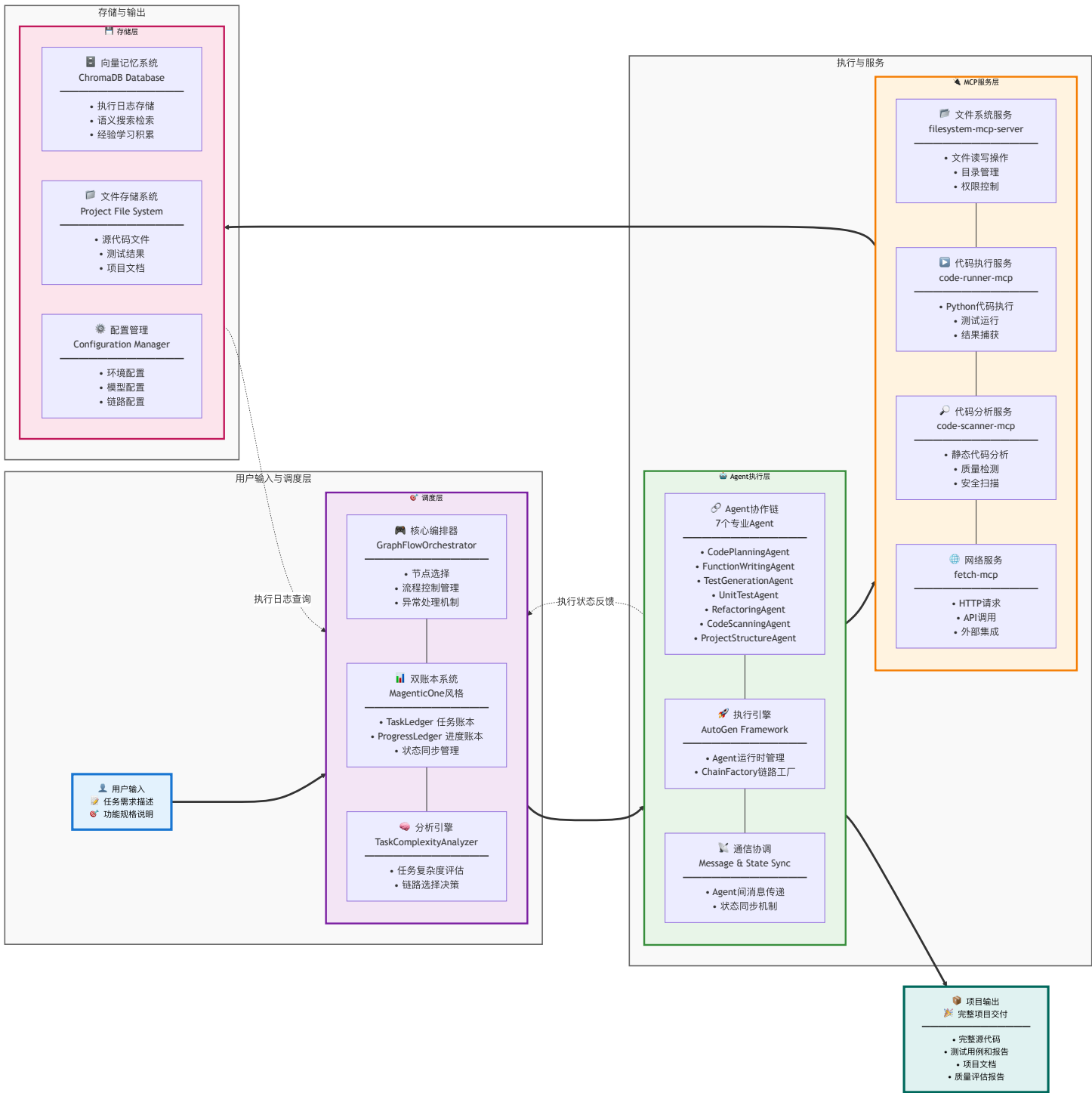
<https://github.com/Jabez2/-MCP-Agent>

核心工作

- **多链路支持**：支持不同复杂度的执行链路，可根据任务复杂度选择最优路径
- **MCP服务集成**：集成多个专业MCP服务，实现Agent工具能力扩展
- **向量化记忆**：基于ChromaDB向量数据库记忆存储，支持 Agent 上下文共享和执行记录存储
- **双账本状态管理**：TaskLedger + ProgressLedger 实现任务层和执行层的状态记录

系统架构

整体架构图

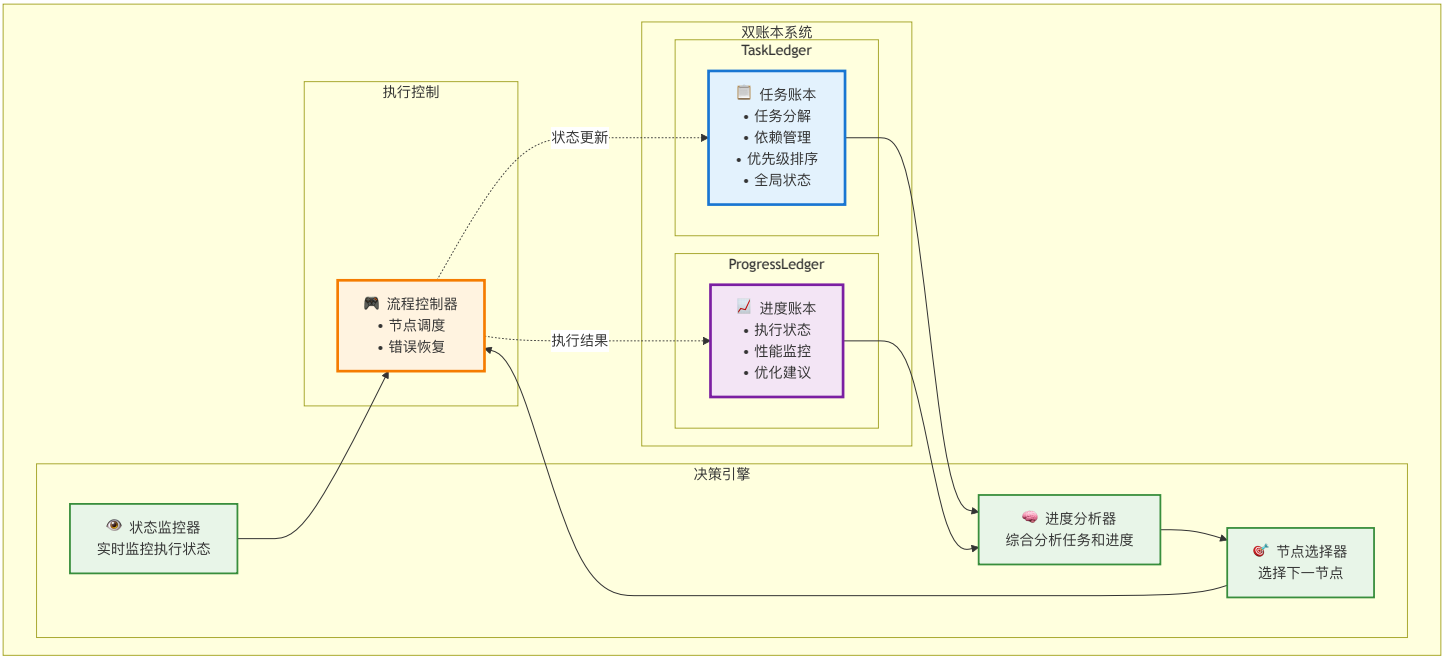


架构特点：

- **调度层：** MagenticOne双账本机制 + 决策引擎
- **Agent执行层：** 7个专业Agent协作 + AutoGen执行引擎
- **MCP服务层：** 标准化工具服务 + 模块化扩展

- 存储层：向量化记忆 + 文件系统 + 配置管理

MagenticOne双账本机制

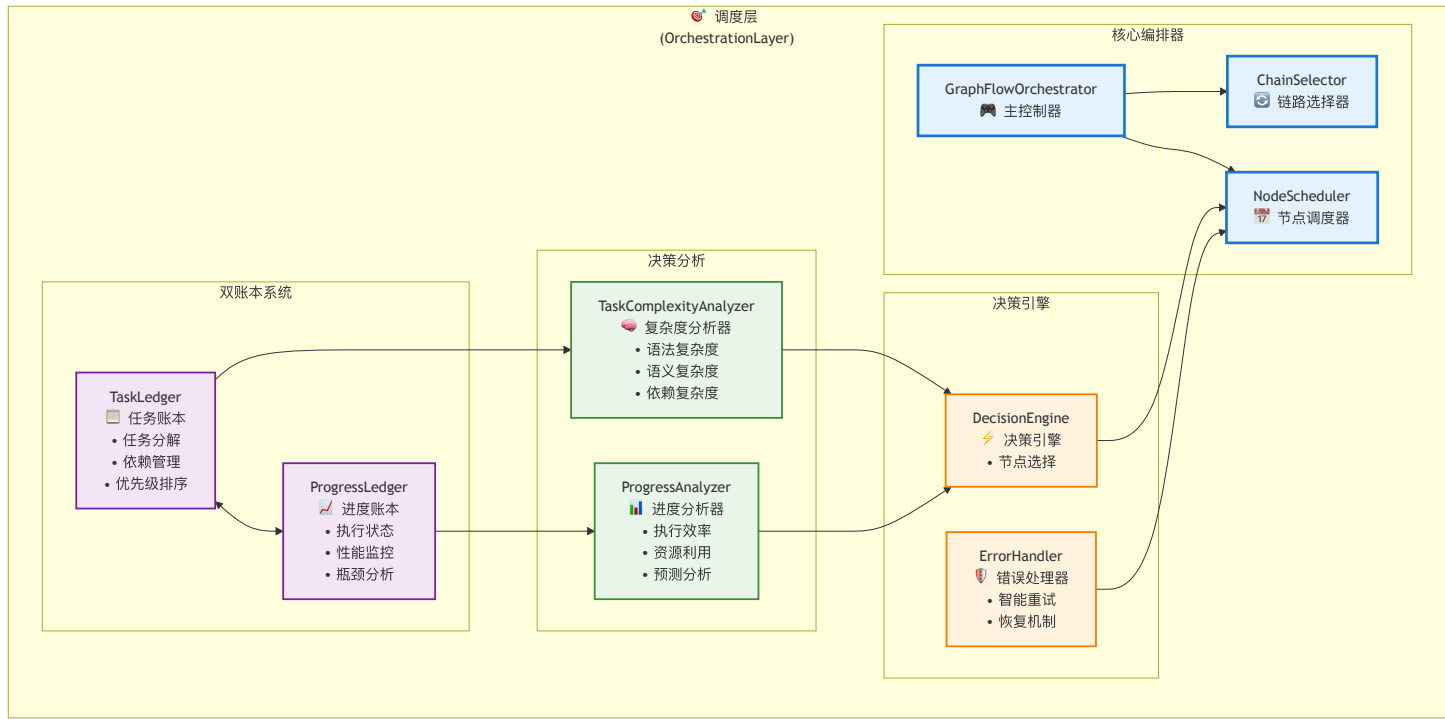


双账本机制核心特性

- 1. **TaskLedger (任务账本):**
 - **任务分解**: 将复杂任务分解为可执行的子任务
 - **依赖管理**: 维护任务间的依赖关系和执行顺序
 - **全局状态**: 实时跟踪整体任务执行状态
- 2. **ProgressLedger (进度账本):**
 - **执行监控**: 实时监控每个Agent的执行状态
 - **性能分析**: 收集和分析执行性能指标
 - **瓶颈识别**: 识别执行过程中的瓶颈点
 - **优化建议**: 基于历史数据提供执行优化建议
- 3. **决策机制:**
 - **进度分析**: 综合分析任务和进度信息
 - **节点选择**: 基于当前状态选择下一个执行节点
 - **错误恢复**: 包含多种错误检测和错误恢复策略

分层架构详细设计

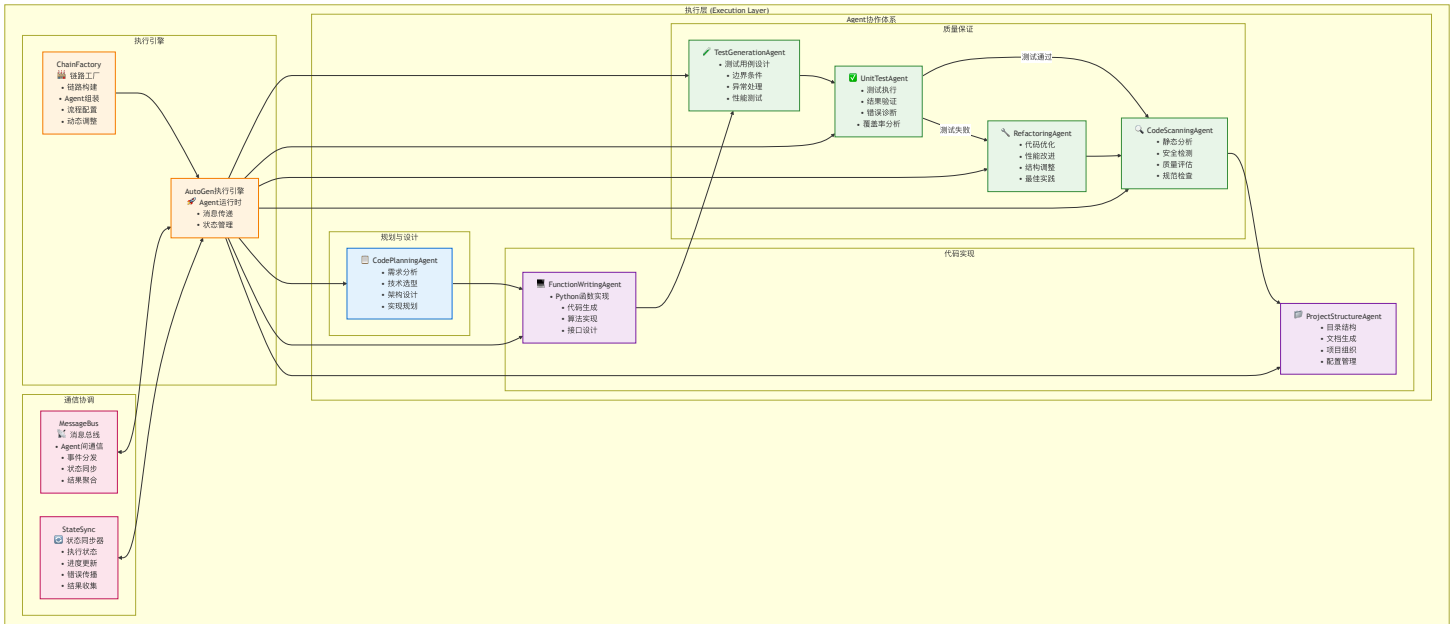
调度层 (Orchestration Layer)



调度层核心功能：

- **GraphFlowOrchestrator**：主控制器，协调整个系统的执行流程
- **双账本系统**：TaskLedger管理任务层面，ProgressLedger管理执行层面
- **智能分析**：复杂度分析和进度分析，为Agent 执行提供更准确的指令
- **决策引擎**：基于分析结果进行决策和错误处理

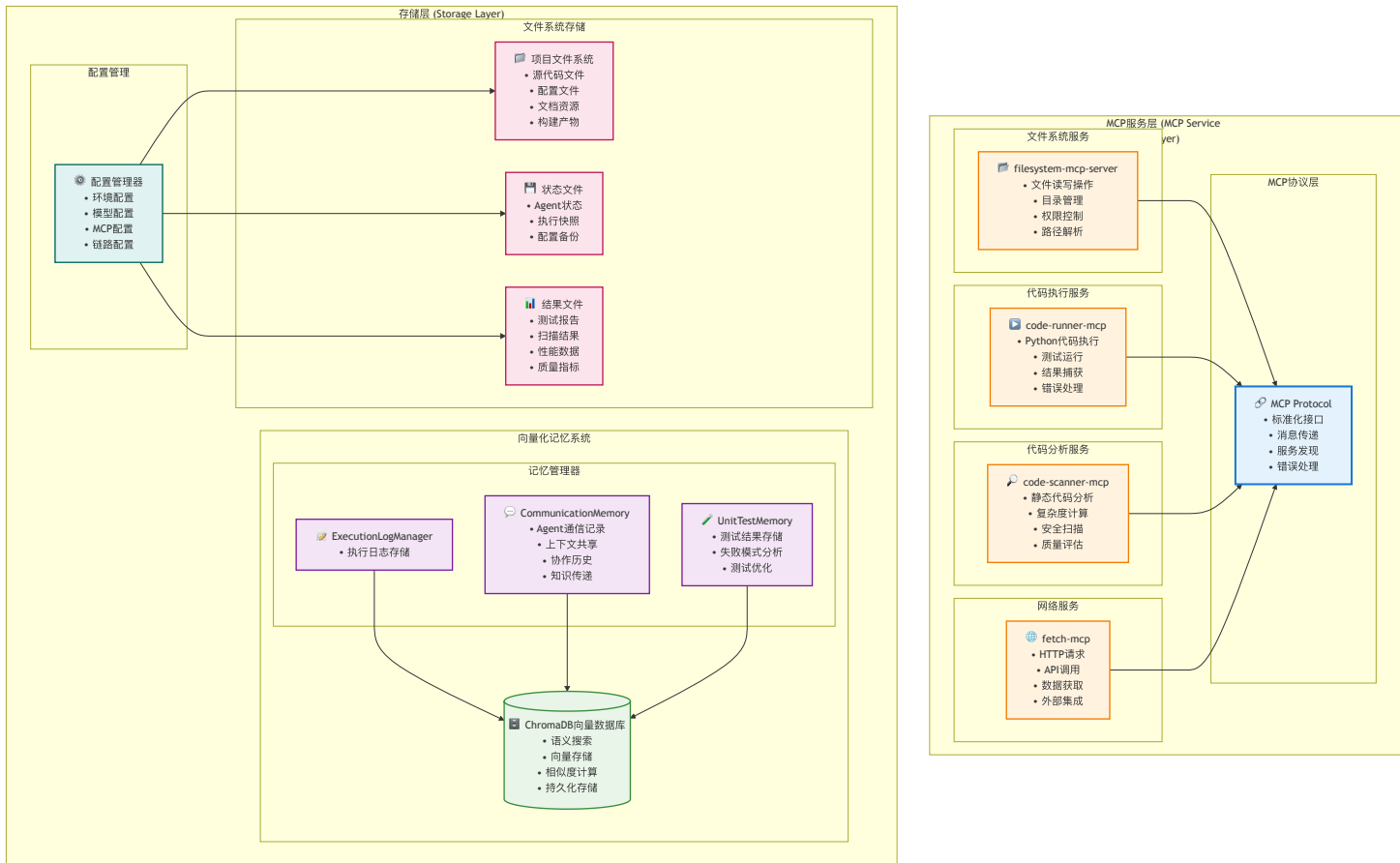
执行层 (Execution Layer)



执行层核心功能：

- **Agent协作体系**：7个专业Agent按职责分工协作
- **执行引擎**：基于AutoGen的Agent运行时和链路工厂
- **通信协调**：消息总线和状态同步器确保Agent间协调

MCP服务层和存储层



MCP服务层核心功能：

- **标准化接口：**基于MCP协议的统一服务接口
- **模块化设计：**可插拔的服务架构，支持动态扩展
- **安全隔离：**每个服务在独立环境中运行
- **错误处理：**完善的错误检测和恢复机制

存储层核心功能：

- **向量化记忆：**基于ChromaDB的语义搜索和知识存储
- **多维度记忆管理：**执行记忆、通信记忆、测试记忆
- **文件系统管理：**项目文件、状态文件、结果文件的分类存储
- **配置管理：**统一的配置管理和环境控制

主要工作内容

1. 系统架构设计

多Agent协作系统

- 基于AutoGen框架：结合GraphFlow结构化执行和MagneticOne双账本风格调度
- 7个专业Agent：每个Agent具有明确的职责分工和专业工具集成
- 节点选择：基于进度账本(ProgressLedger)分析的调度
- 异步执行流程：支持Agent间的协调和状态同步

MCP服务集成

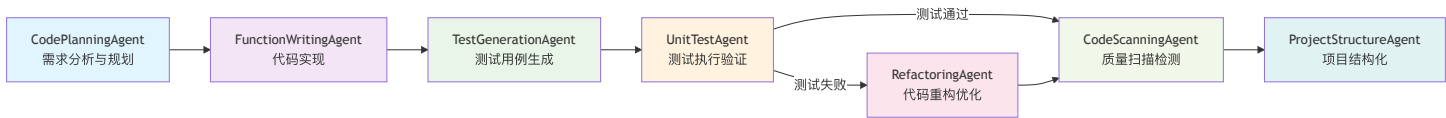
- filesystem-mcp-server：提供完整的文件系统操作能力
- code-runner-mcp：支持代码执行和测试运行
- code-scanner-mcp：静态代码分析和质量检测
- fetch-mcp：外部API调用和数据获取

三层架构

- 调度层：GraphFlowOrchestrator核心编排器
- 执行层：基于AutoGen的Agent执行引擎
- 存储层：基于ChromaDB的向量化内存管理

2. 多Agent系统开发

Agent协作体系



专业工具集成

每个Agent配置专门的系统消息和MCP工具集成：

- **CodePlanningAgent**：需求分析、技术选型、实现规划。配置 FileSystem MCP。
- **FunctionWritingAgent**：Python函数实现、代码生成。配置 FileSystem MCP。
- **TestGenerationAgent**：测试用例设计、边界条件覆盖。配置 FileSystem MCP。
- **UnitTestAgent**：测试执行、结果验证、错误诊断。配置 Code-runner MCP。

- **RefactoringAgent**: 代码优化、性能改进、结构调整。配置 FileSystem MCP。
- **CodeScanningAgent**: 静态分析、安全检测、质量评估。配置静态扫描MCP。
- **ProjectStructureAgent**: 目录结构、文档生成、项目组织。配置 FileSystem MCP。

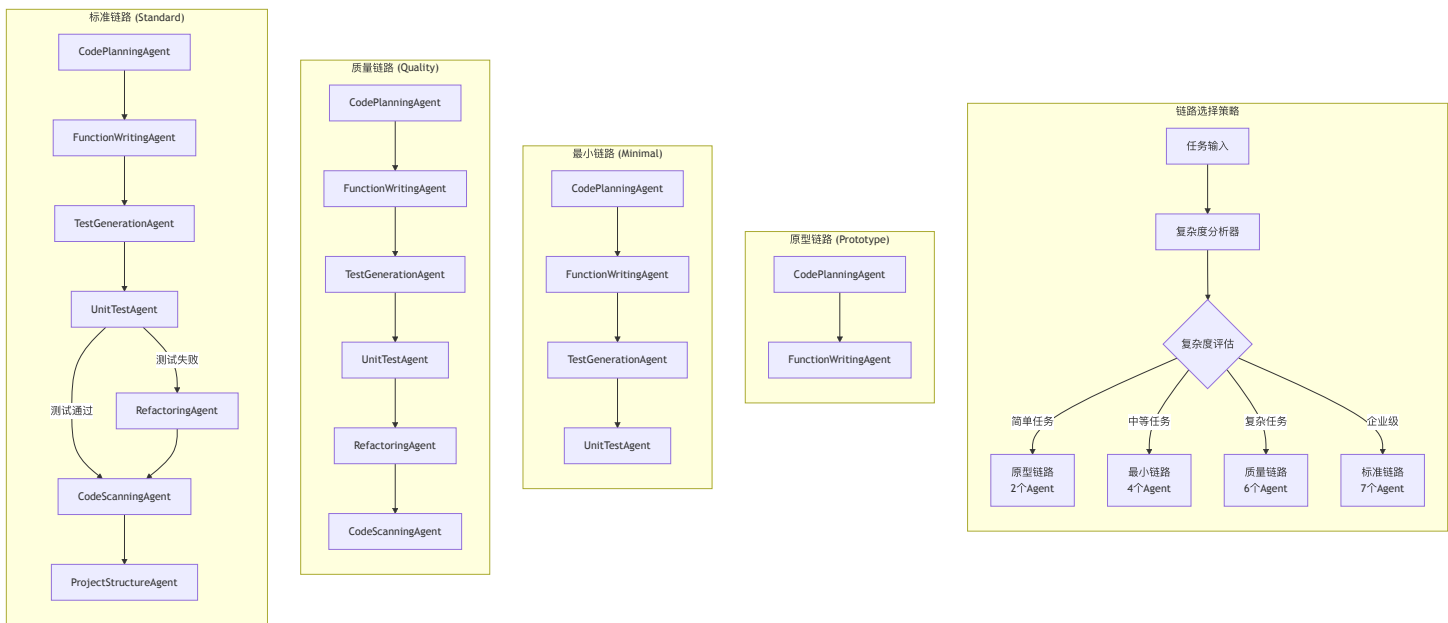
任务复杂度分析

任务评估方法：

- **静态代码分析**：函数复杂度、依赖关系、代码规模
- **LLM辅助评估**：语义理解、技术难度、实现复杂度

3. 多种链路

四种执行链路

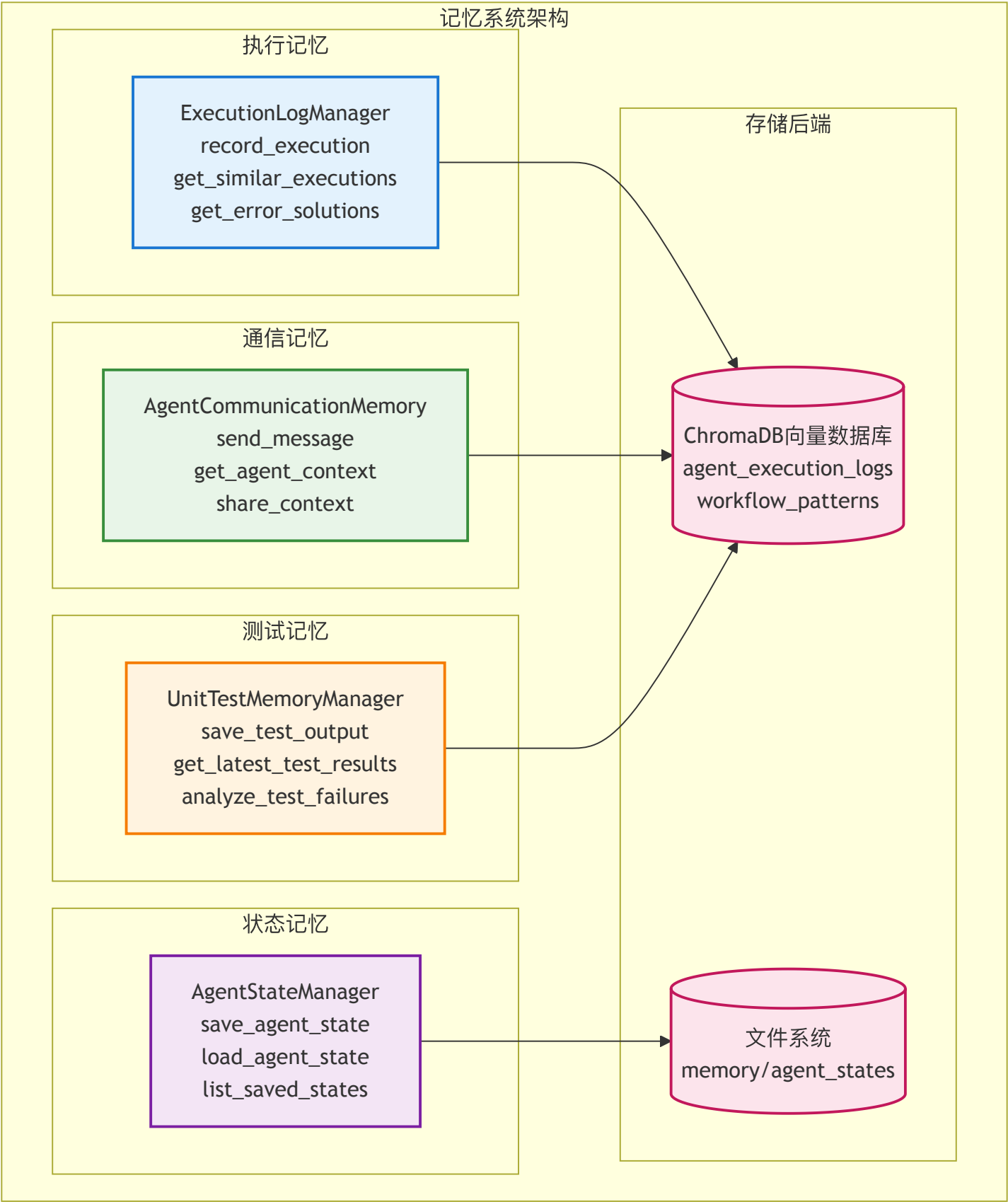


4. 向量化数据库

ChromaDB

- **采用模型**：使用paraphrase-multilingual-MiniLM-L12-v2本地部署模型
- **语义搜索能力**：支持中英文混合查询

记忆管理

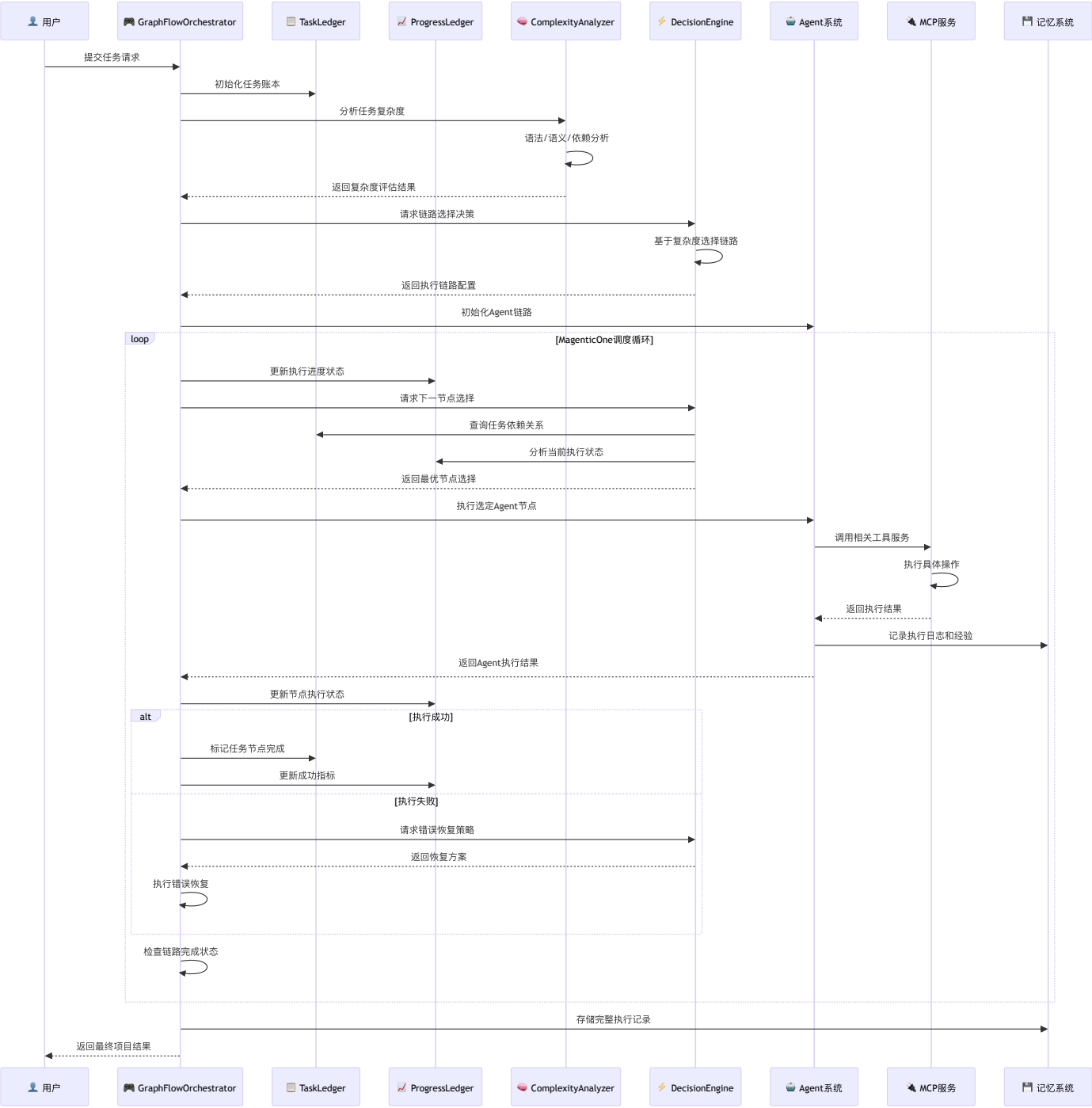


技术栈

- 开发语言：Python 3.13
- **AI框架**：AutoGen AgentChat 0.4.0+
- 协议标准：MCP (Model Context Protocol) 1.0+
- 向量数据库：ChromaDB with Sentence Transformers
- **AI模型**：支持OpenAI兼容API，默认使用Qwen3-Coder

附录

A. 系统执行时序图



B. MCP服务详细配置

1. 文件系统MCP服务配置

```
{
filesystem_mcp_server = StdioServerParams(
    command="node",
    args=[
        "/Users/jabez/Nutstore Files/multiAgent/mcp_services/filesystem-mcp-server/",
        "/Users"
    ],
    env={
        "FS_BASE_DIRECTORY": "/Users"
    }
)
```

2. 代码扫描MCP服务配置

```
{
  "name": "code-scanner-mcp",
  "version": "0.1.0",
  "scan_types": [
    {
      "name": "complexity",
      "description": "复杂度分析: 圈复杂度、认知复杂度、Halstead复杂度"
    },
    {
      "name": "style",
      "description": "代码风格检查: PEP8合规性、命名规范、导入排序"
    },
    {
      "name": "security",
      "description": "安全扫描: 安全漏洞检测、依赖安全检查"
    },
    {
      "name": "documentation",
      "description": "文档质量: 文档字符串检查、类型注解检查"
    }
  ],
  "supported_formats": ["json", "markdown", "html"],
  "supported_languages": [".py"]
}
```

3.代码执行 MCP 服务配置

```
{
  code_runner_mcp_server = StdioServerParams(
    command="npx",
    args=[
      "-y",
      "mcp-server-code-runner@latest"
    ]
  )
}
```

C. 向量数据库配置详情

ChromaDB配置参数

```
# 执行日志向量存储配置
execution_memory_config = {
    "collection_name": "agent_execution_logs",
    "embedding_model": "paraphrase-multilingual-MiniLM-L12-v2",
    "distance_metric": "cosine",
    "vector_dimension": 384,
    "k": 50, # 返回最相关的50个结果
    "score_threshold": 0.0, # 不过滤任何结果
    "persistence_path": "./memory/execution_logs/"
}

# 工作流模式向量存储配置
workflow_memory_config = {
    "collection_name": "workflow_patterns",
    "embedding_model": "paraphrase-multilingual-MiniLM-L12-v2",
    "distance_metric": "cosine",
    "vector_dimension": 384,
    "k": 3, # 返回最相关的3个工作流模式
    "score_threshold": 0.0,
    "persistence_path": "./memory/workflow_patterns/"
}
```

嵌入模型选择

模型名称	维度	大小	语言支持
all-MiniLM-L6-v2	384	22MB	英文为主
paraphrase-multilingual-MiniLM-L12-v2	384	118MB	多语言
all-mpnet-base-v2	768	420MB	英文

存在的问题

- 链路内部节点执行主要采用顺序执行，缺乏并行执行能力
- 链路配置不够灵活，无法根据运行时动态调整
- 有向量数据库，但Agent 目前不具备 RAG 能力

- Agent 扩展比较复杂，需要修改多处代码
- ...