Lesson 14: Implementing Agentic Orchestrator-Workers Pattern in Python

Overview

This lesson focuses on implementing the Orchestrator-Workers pattern in Python, which is a flexible agentic workflow pattern that allows a central orchestrator agent to coordinate multiple specialized worker agents.

Key Concepts

1. Orchestrator-Workers Pattern

- **Orchestrator Agent**: Central agent that analyzes user prompts, determines required tasks, and coordinates worker agents
- **Worker Agents**: Specialized agents that handle specific types of tasks
- **Dynamic Task Assignment**: The orchestrator dynamically assigns tasks to appropriate workers based on task type

2. Pattern Components

- **Task Analysis**: Orchestrator analyzes user input and breaks it down into specific tasks
- **Worker Selection**: Orchestrator selects appropriate worker agents based on task requirements
- **Result Synthesis**: Orchestrator combines results from multiple workers into a final response

Exercise: Lab Results Interpretation Assistant

Challenge

Implemented the `get_worker` method in the Orchestrator class to dispatch tasks to appropriate specialist agents:

- **HematologyAgent**: Handles blood count and hematology-related tasks
- **RenalFunctionAgent**: Handles kidney and renal function tasks
- **LiverFunctionAgent**: Handles liver function tasks

Implementation

```python
def get_worker(self, task_type: str) -> WorkerAgent:
    type_lower = task_type.lower()
    if "hematology" in type_lower or "blood count" in type_lower:
        return HematologyAgent(task_type)
    elif "renal" in type_lower or "kidney" in type_lower:
        return RenalFunctionAgent(task_type)
    elif "liver" in type_lower:
        return LiverFunctionAgent(task_type)
    else:
        raise ValueError(f"No worker agent configured for task type: {task_type}")
```

Workflow Process

1. **Plan Creation**: Orchestrator creates a dynamic plan based on user input
2. **Task Parsing**: XML response is parsed to extract individual tasks
3. **Worker Dispatch**: Each task is sent to the appropriate specialist worker
4. **Result Collection**: Results from all workers are collected and synthesized

Key Learning Points

- The orchestrator pattern provides flexibility to handle diverse use cases
- Dynamic task assignment allows for scalable and maintainable agent systems
- Proper error handling ensures robust operation when task types don’t match available workers
- XML parsing enables structured communication between orchestrator and workers

Status

✅ **COMPLETED**: Successfully implemented the get_worker method and ran the exercise successfully.