Technical Assessment

Technical Assessment: Preparation Time 5 Working Days

In this assignment, you will design and implement multi-agent systems that coordinate complex workflows. It simulates orchestration challenges, requiring both technical expertise and architectural thinking.

Overview

You will build a **Content Intelligence Pipeline** using multi-agent orchestration with Retrieval Augmented Generation (RAG) to process and analyse content requests. This scenario requires multiple specialised agents working together to deliver comprehensive results.

The Scenario: Content Intelligence Pipeline

Your system should handle a request like: *"Create a comprehensive clinical evidence synthesis for the effectiveness of telemedicine interventions in managing Type 2 diabetes"*

The pipeline should orchestrate multiple agents to:

- **Research Agent**: Gather relevant information from multiple sources
- **Analysis Agent**: Process and synthesise the gathered data
- **Writing Agent**: Structure and compose the final report
- **Quality Agent**: Review and validate the output quality
- **Coordinator Agent**: Orchestrate the entire workflow

Sample Workflow

Note: This is a sample for your reference. Feel free to develop your own workflow, which doesn't have to be limited to healthcare.

Input: "*Create a comprehensive clinical evidence synthesis for the effectiveness of telemedicine interventions in managing Type 2 diabetes*"

Agent Flow:

1. Coordinator → Research Agent: "Gather clinical evidence on telemedicine effectiveness for Type 2 diabetes management from peer-reviewed sources"

2. Research Agent uses multiple specialised tools:
    - PubMed API for peer-reviewed medical literature
    - Cochrane Database for systematic reviews and meta-analyses
    - ClinicalTrials.gov for ongoing and completed clinical trials
    - Medical society guidelines and position statements
    - Returns structured data with study methodologies, patient populations, and outcome measures

3. Research Agent → Analysis Agent: "Here's clinical data from 47 studies, including 12 RCTs, 8 systematic reviews, and 27 observational studies"

4. Analysis Agent processes medical evidence:
    - Categorises studies by intervention type (remote monitoring, video consultations, mobile apps)
    - Analyses patient outcomes (HbA1c levels, medication adherence, quality of life measures)
    - Identifies study limitations and potential biases
    - Synthesises evidence strength using established clinical frameworks
    - Returns structured clinical insights with evidence quality ratings

5. Analysis Agent → Writing Agent: "Here's processed clinical evidence with outcome measures, effect sizes, and quality assessments"

6. Writing Agent creates clinical evidence synthesis:
    - Structures content following medical writing standards (PRISMA guidelines)
    - Incorporates statistical analyses and forest plots where appropriate
    - Maintains clinical terminology and evidence-based language
    - Creates executive summary with clinical recommendations
    - Returns draft synthesis with proper medical citations

7. Writing Agent → Quality Agent: "Review this clinical evidence synthesis for Type 2 diabetes telemedicine interventions"

8. Quality Agent validates clinical content:
    - Fact-checks medical claims against original study data
    - Ensures appropriate interpretation of statistical significance
    - Checks for clinical consistency and logical conclusions
    - Validates citation accuracy and formatting
    - Returns feedback on clinical accuracy and evidence interpretation

9. Quality Agent → Coordinator: "Clinical synthesis approved with minor statistical clarifications needed"

10. Coordinator → User: "Final clinical evidence synthesis delivered with comprehensive citations and evidence quality ratings"

Technical Requirements

Framework Flexibility

Choose **one** of the following approaches (we encourage you to pick the one you're most interested in exploring):

- **Semantic Kernel**: Microsoft's orchestration framework (can optionally integrate with Azure OpenAI Service)
- **LangGraph**: LangChain's graph-based agent framework
- **OpenAI Agent SDK**: OpenAI's native agent development kit

- **Strands**: Open-source multi-agent orchestration framework
- **Custom Implementation**: Using your preferred language/framework, or hybrid approach combining multiple frameworks
- **Cloud-Native**: Build on AWS Bedrock or Azure AI services as your foundation layer

Language Model Flexibility

You have complete flexibility in choosing your underlying language models:

- **Azure OpenAI Service**: GPT-4o, GPT-4o-mini, GPT-4.1, GPT-4.1 mini, or other Azure-hosted models
- **OpenAI API**: Direct integration with OpenAI's models
- **Google Vertex AI**: Gemini Pro, Gemini Flash, and other Google models
- **Anthropic API**: Claude models for specific agent capabilities
- **Mixed Approach**: Different models for different agents based on their specialised tasks

Agent Tool Examples

Your agents can integrate with various tools and services:

- **Research Agent**: Bing Search API, Google Search, Tavily API, web scraping tools, database queries, i.e. vector search
- **Analysis Agent**: Pandas for data processing, statistical analysis libraries, and visualisation tools
- **Writing Agent**: Document formatting libraries, template engines, citation management
- **Quality Agent**: Fact-checking APIs, grammar checking, response quality checks for completeness and relevancy

Core Deliverables

1. Multi-Agent System Implementation

- **Agent Design**: Implement at least three specialised agents with distinct roles
- **Orchestration Logic**: Create a coordinator that manages agent interactions
- **State Management**: Handle data flow between agents effectively
- **Error Handling**: Implement robust error handling and retry mechanisms
- **Traceability**: Add logging and observability for agent interactions

2. System Architecture

- **Agent Communication**: Design how agents share information and coordinate
- **Workflow Management**: Implement conditional logic and branching workflows
- **Resource Management**: Handle concurrent agent execution efficiently
- **Configuration**: Make the system configurable for different content types

3. Code Quality & Documentation

- **Repository Structure**: Organise code in a clear, maintainable structure
- **Documentation**: Comprehensive README.md with setup and usage instructions
- **Testing**: Include unit tests for key components
- **Examples**: Provide sample inputs and expected outputs

Detailed Instructions

Step 1: System Design (Day 1)

- Design your agent architecture and interaction patterns
- Create sequence diagrams showing agent workflows
- Define data models for inter-agent communication
- Plan your error handling and monitoring strategy

Step 2: Agent Implementation (Days 2-3)

- Implement each specialised agent with clear responsibilities
- Build the orchestration layer that coordinates agent interactions
- Create configuration management for different workflow types
- Implement proper logging and state tracking

Step 3: Integration & Testing (Day 4)

- Integrate all agents into a working pipeline
- Test with various content request scenarios
- Implement error recovery and fallback mechanisms
- Add performance monitoring and metrics collection

Step 4: Documentation & Refinement (Day 5)

- Write comprehensive documentation with architectural decisions
- Create clear usage examples and API documentation
- Add deployment instructions and configuration guides
- Prepare demo scenarios for the presentation

Technical Evaluation Criteria

We'll evaluate your solution based on four key areas that reflect the skills you'll use daily in multi-agent system development:

Architecture & Design

Understanding how you approach system design reveals your ability to create scalable, maintainable solutions. We'll assess how effectively you separate concerns between agents, establishing clear boundaries and responsibilities for each component.

Your orchestration patterns demonstrate how you think about complex workflows and coordination challenges. We're also keen to see how you designed for scalability and whether your system can handle increased complexity gracefully. Error resilience is essential because multi-agent systems face unique failure modes that single-agent systems do not encounter.

Implementation Quality

Your code quality reflects your professional development practices and attention to detail. We will assess how well you organised your codebase for maintainability and whether other developers can easily understand and extend your work.

Your use of the chosen framework demonstrates whether you can effectively utilise existing tools or if you tend to reinvent solutions. Performance considerations reveal your understanding of resource management in distributed systems. The flexibility of your configuration system indicates how thoroughly you've planned deployment across different environments.

Multi-Agent Coordination

This is where multi-agent systems become challenging and many implementations fail. We'll assess how well your workflow management handles complex scenarios, including conditional logic, parallel processing, and error recovery.

Your approach to state management indicates whether you grasp the complexities of maintaining consistency across distributed agents. Resource optimisation shows how effectively you can balance responsiveness with efficiency. Your implementation of monitoring and observability demonstrates whether you've considered the operational aspects of managing multi-agent systems in production.

Documentation & Presentation

Clear communication is essential when working with complex systems that involve multiple stakeholders. Your documentation should help others understand not just what you've built, but also why you made specific design choices.

We'll assess whether your README provides a clear path from installation to successful operation. Your architecture documentation should outline your thought process and help others learn from your approach. Code comments should offer insight into complex logic and decision points. Ultimately, your preparation for the demo demonstrates your ability to communicate technical concepts to diverse audiences effectively.

Repository Requirements

Required Files

```
├── README.md (Comprehensive documentation)
├── ARCHITECTURE.md (System design and agent interaction patterns)
├── requirements.txt or package.json (Dependencies)
├── src/ (Source code organised by agent/component)
├── tests/ (Unit tests for key components)
├── config/ (Configuration files and examples)
├── docs/ (Additional documentation and diagrams)
└──examples/ (Sample requests and outputs)
```

README.md Must Include

- **Project Overview**: What does the system do and why?
- **Architecture**: High-level system design with diagrams
- **Setup Instructions**: Step-by-step installation and configuration
- **Usage Examples**: How to run the system with sample inputs
- **Agent Descriptions**: What each agent does and how they interact
- **Framework Justification**: Why you chose your specific framework
- **Performance Considerations**: Scalability and optimization notes
- **Future Enhancements**: What would you add with more time?

Bonus Considerations

While not required, these additions would strengthen your submission:

- **API Interface**: REST or GraphQL API for system interaction
- **Web Interface**: Simple UI for submitting requests and viewing results
- **Docker Support**: Containerization for easy deployment
- **Cloud Integration**: Integration with cloud services (AWS, Azure, GCP)
- **Advanced Orchestration**: Conditional branching, parallel execution, or retry logic
- **Metrics Dashboard**: Visual monitoring of agent performance (Arize)

Presentation Preparation

After acceptance, you'll have 5 additional days to prepare a presentation covering:

- **System Architecture**: How agents work together
- **Live Demo**: Processing a content request end-to-end
- **Technical Deep Dive**: Interesting implementation challenges
- **Lessons Learned**: What you discovered about multi-agent orchestration

---

Submission: Kindly share your GitHub repository URL along with complete documentation and working code. Make sure all sensitive API keys are securely managed through environment variables or configuration files.