LUMOS : Lowcode Usercentric Multi-agent Orchestration System

Team 14

March 2025

1 Description of the Use Case

The primary users and stakeholders affected by this problem include data scientists, machine learning engineers, business analysts, and domain experts, startup founders who want to leverage LLM-based multi-agent systems without extensive programming expertise. Currently, building and deploying such systems require significant technical knowledge, including integrating multiple agents, defining interactions, and managing deployment infrastructure. This creates a barrier for non-technical users and slows down the adoption of LLM-powered automation in various industries. Our project addresses this challenge by providing a low-code platform that allows users to visually design, configure, and deploy multi-agentic LLM systems. By enabling seamless conversion of agentic system specifications into Python code or deployable endpoints, we eliminate the complexity of manual coding and integration. This significantly lowers the entry barrier for LLM adoption, accelerates development cycles, and democratizes access to advanced AI capabilities, making it easier for businesses and individuals to implement intelligent automation solutions.

2 Key Functionalities

2.1 Core Features

- **Visual Design**: Canvas for creating AI agent systems with templates and edit options.
- **Multi-Agent Orchestration**: Workflow tools with conditional logic and monitoring
- **LUMOS Definition Language (LDL)**: JSON specification for agent systems
- **Deployment**: One-click API deployment and code generation
- **Testing**: Debugging and analytics tools with summary of decisions.

2.2 Technical Implementation

Web interface with interactive canvas, CLI tools, and comprehensive API access. Built on React.js frontend with Python/FastAPI backend, supporting LLM integration using API Keys provided by users. Employs three-tier microservices architecture: Frontend (UI, Canvas), Middleware (API Gateway, LDL Translator), and Backend (Executor, LLM Adapters).

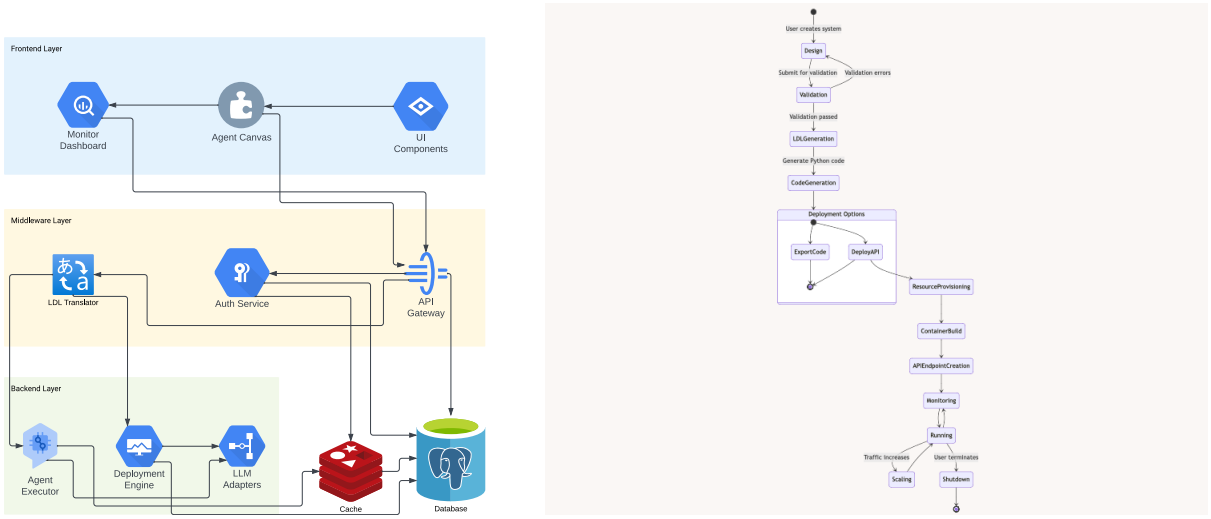
2.3 Design Patterns

- **Adapter Pattern**: Provides uniform interface to various LLM providers (OpenAI, Anthropic, open-source models) with different APIs and requirements. Allows seamless switching between models without changing application code.
- **Observer Pattern**: Implements a publish-subscribe mechanism where components register interest in specific events (agent activation, state changes, errors). Enables real-time monitoring dashboard and decoupled system components.
- **Strategy Pattern**: Permits runtime selection of different orchestration algorithms for agent execution. Teams can choose sequential, parallel, or conditional execution strategies based on specific needs.

- **Composite Pattern**: Treats individual agents and agent groups uniformly through common interfaces. Enables building hierarchical agent systems where sub-systems can function as independent units.

2.4 Architectural Tactics

LUMOS incorporates several architectural tactics to enhance scalability, reliability, security, and maintainability. Layered architecture ensures separation of concerns, where the frontend, middleware, and backend operate independently, enabling modular updates and easier debugging. Interoperability is achieved by abstracting agent interactions through a standardized translation layer, ensuring seamless integration of diverse components. Performance optimization is handled through caching mechanisms that reduce redundant computations and database queries. Security and access control are enforced via an authentication mechanism at the API gateway, ensuring only authorized access. Scalability is supported by dynamically allocating resources for agent execution and deployment, preventing bottlenecks. Fault tolerance is improved through real-time monitoring and logging, allowing quick detection and mitigation of failures.



(a) Overall Architecture

(b) Deployment Pipeline

Figure 1: System Overview

3 Expected Time to Build a Prototype

- **Week 1:** Research & planning – problem analysis, user needs, tech evaluation, and architecture design.
- **Week 2:** UI/UX design, database schema, API contracts, security model, and deployment strategy.
- **Week 3:** Backend & frontend development – core services (Auth, API, execution engine), UI framework, and state management.
- **Week 4:** Integration, testing, performance tuning, documentation, and final deployment.
- **Total Time: 4 weeks**, with an MVP possible in **3 weeks**.

4 Domain

The project falls under the **Software Engineering and AI Automation** domain, impacting businesses, researchers, and citizen developers who require AI automation without deep ML expertise. By providing a no-code/low-code solution, we enhance accessibility, reduce development time, and foster innovation in AI-driven applications.