LUMOS : Lowcode Usercentric Multi-agent Orchestration System

Team 14

March 20, 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 Functional Requirements

- **Visual Design Interface**: Interactive canvas for creating AI agent systems with intuitive editing options.
- **Multi-Agent Orchestration**: Workflow tools with conditional logic, parallel execution paths, and real-time monitoring.
- **LUMOS Definition Language (LDL)**: JSON-based specification language for modeling agentic systems.
- **Deployment Capabilities**: One-click API deployment and automatic code generation.
- **Template Library**: Collection of pre-built agent templates and workflow patterns for common use cases.

2.2 Non-Functional Requirements

- **Performance**: The system should aim to maintain user engagement and process agent execution requests within reasonable timeframes (5s) based on workflow complexity.
- **Reliability**: The system should strive for high robustness, with recovery mechanisms for component failures and data persistence to safeguard user-designed workflows against interruptions.
- **Usability**: The interface should be intuitive enough for users with limited technical background to create agent workflows after brief orientation, keeping System Usability Scale (SUS) score above 75.
- **Maintainability**: The codebase should follow consistent organization with sufficient documentation, with technical debt below 5%.
- **Interoperability**: LUMOS should support integration with atleast 3 major LLM providers, work effectively with common cloud platforms, and support transferable workflow formats compatible with version control.

2.3 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).

2.4 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 subsystems can function as independent units.

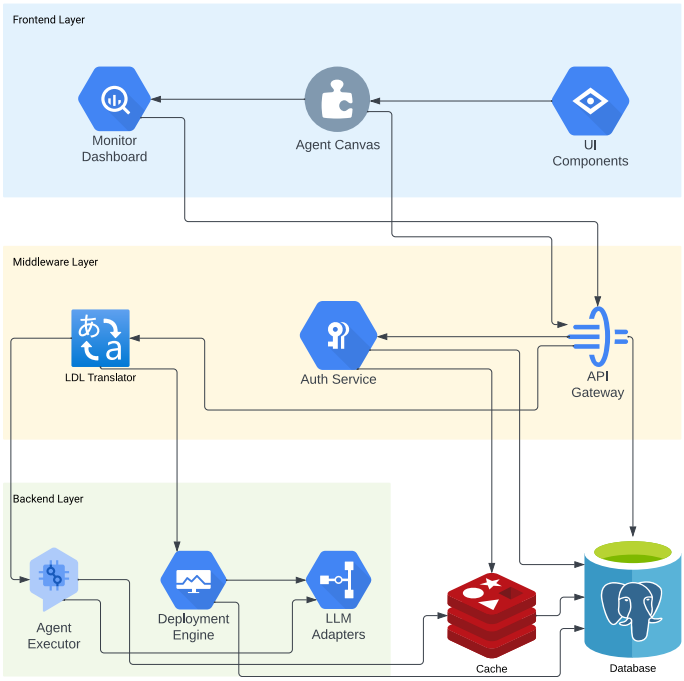

Figure 1: Overall Architecture

2.5 Architectural Tactics

- **Layered Architecture**
  - Keeps frontend, middleware, and backend working independently
  - Makes updates easier since we can change one part without breaking others
  - Helps with debugging by isolating problems to specific layers
- **Interoperability**
  - Uses a translation layer so different agents can talk to each other
  - Helps connect diverse components without compatibility issues and makes it easier to add new types of agents in the future
- **Performance Optimization**
  - Uses caching to avoid repeating the same calculations
  - Reduces unnecessary database lookups
  - Keeps the system responsive even with complex workflows
- **Fault Tolerance**
  - Watches for problems in real-time
  - Keeps track of what's happening with detailed logs
  - Helps quickly find and fix issues before users notice

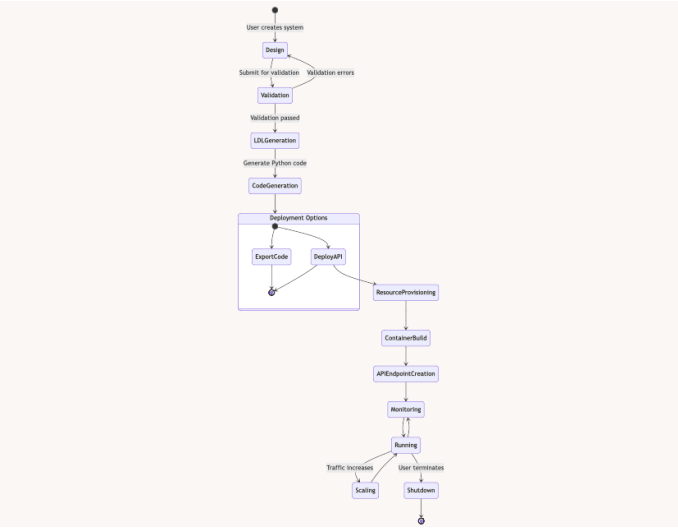

Figure 2: Deployment Pipeline

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.
[1]

[1] Some of our teammates had a discussion with Prof. Karthik regarding the scope of the project and its potential as a research work. Consequently, he suggested keeping an open scope in the initial stage of the project and narrowing it down as we progress with our implementation.