Major Project Report: LMA-o (Learning-Material Agent - omni)

Subject Matter Expert AI Agent for Software Engineering

Raveesh Vyas Prakhar Singhal

November 18, 2025

Abstract

This report documents the design, implementation, and evaluation of **LMA-o**, a highly extensible Retrieval-Augmented Generation (RAG) system designed to serve as a Subject Matter Expert (SME) in the domain of Software Engineering. The system leverages a multi-provider LLM architecture, a modular RAG pipeline with hybrid search and reranking, and an agentic workflow engine powered by LangGraph. This document details the system architecture, design choices, implementation of core and bonus capabilities, and comprehensive evaluation results.

Contents

1 Project Overview

1.1 Objective

The objective of this major project was to design and implement a robust AI agent capable of acting as a **Subject Matter Expert (SME)**. The system is required to perform complex question-answering, generate expert content, and execute multi-step workflows using tool-calling capabilities.

1.2 Domain Selection: Software Engineering

The chosen domain for this SME is **Software Engineering**. The agent is designed to assist users with:

- Technical problem-solving and architectural advice.
- Generation of technical documentation (PDF, DOCX) and presentations.
- Educational support for students and teachers through adaptive explanations and learning materials.
- Code analysis and generation.

2 System Architecture

The system follows a microservices architecture comprising three main components: the Main API Server, the Chat/Agent Server, and the Modular RAG Pipeline.

2.1 High-Level Architecture

The architecture is designed to be modular and scalable. The interactions are as follows:

- **Frontend**: An interactive React-based Web UI communicates with the backend via REST and WebSockets.
- **Main API Server (FastAPI)**: Acts as the gateway, handling authentication, rate limiting, and request routing.
- **Agent Server (LangGraph)**: Manages stateful conversational workflows, planning, and tool orchestration.
- **RAG Pipeline**: Handles document ingestion, embedding, hybrid retrieval, and reranking.

3 Implementation Details

3.1 Tech Stack

- **Language**: Python 3.10+

- **API Framework**: FastAPI (Asynchronous support, validation)
- **Agent Framework**: LangGraph (Stateful orchestration)
- **Vector Database**: Elasticsearch (Dense vector + Sparse keyword search)
- **LLM Integration**: Support for OpenAI, Anthropic, Google Gemini, and Ollama (local)
- **Frontend**: React.js with TypeScript and Material-UI.

3.2 RAG Pipeline

The Retrieval-Augmented Generation pipeline is designed for high precision:

1. **Ingestion**: Supports PDF, DOCX, MD, and TXT formats. Documents are chunked using a content-aware strategy with overlap to preserve semantic continuity.
2. **Hybrid Search**: Combines dense vector embeddings (using `all-mpnet-base-v2` and `GraphCodeBERT`) with keyword-based BM25 search.
3. **Reranking**: Implements a reranking step using BGE Cross-Encoder to refine the top-k results before context assembly.

3.3 Agentic Capabilities

The Agent Server uses **LangGraph** to manage workflows. Key modules include:

- **Planning Module**: Decomposes complex queries into subtasks (ReAct framework).
- **Tool Orchestrator**: Dynamically selects and executes tools based on the plan.
- **Memory Management**: Maintains conversation history and user preferences.

4 Bonus Features Implemented

The following bonus requirements have been successfully implemented:

4.1 Reranking Mechanism

Requirement: Implement a reranking step with fallback to basic similarity search.
Implementation: The `KnowledgeRetrievalTool` integrates a `BGEReranker`. Initial retrieval fetches a broad set of candidates (e.g., top 50) using hybrid search, which are then re-scored for semantic relevance to select the final context window.

4.2 Automated Batch Ingestion

Requirement: Automated batch ingestion pipeline with error logging.
Implementation: The `DocumentProcessor` and `DataPipeline` components support batch processing of heterogeneous file types from the `data/ingested_documents` directory, with automatic format detection and error handling.

4.3 Self-Learning & Adaptation

Requirement: Incorporate human feedback to reorganize content or improve clarity.
Implementation: An `AdaptationEngine` and `FeedbackLearning` module have been implemented. The system captures user feedback (ratings, corrections) to adjust future planning strategies and prompt structures.

4.4 Guardrails (Input/Output)

Requirement: Input sanitization and output moderation.
Implementation: The API Server includes a `Security` middleware layer.

- **Input:** Regex patterns and injection detection sanitizers.
- **Output:** Hallucination prevention checks and content moderation filters before response delivery.

5 Evaluation Results

A comprehensive test suite was developed to validate the system.

5.1 Test Suite Summary

- **Total Test Files**: 21
- **Categories**: Unit, Integration, E2E, Performance, Security.
- **Coverage**: >85% overall system coverage.

5.2 Key Performance Metrics

- **Concurrency**: Validated for 100+ simultaneous users via $\texttt{test}_{l}oad_{t}esting.py$.
- `Latency: API response time averages <200ms; Search response <100ms.`
- `Retrieval Quality: Verified using test_rag_integration.py to ensure relevant chunks are retrieved for domain queries.`

6 User Manual: Setup and Usage

6.1 Installation

The system is containerized for easy deployment.

1. **Clone Repository**:

```
git clone <repo-url>
cd LMAo

```

2. **Configuration**: Copy the example environment file and configure API keys.

```
cp .env.example .env
# Edit .env: Add OPENAI_API_KEY, GOOGLE_API_KEY, etc.

```

3. **Docker Deployment**:

```
docker-compose up -d

```

6.2 API Usage Examples

6.2.1 Text Generation

```
# POST /api/v1/chat/message
{
  "message": "Explain the Factory Pattern in Python",
  "session_id": "user-session-123",
  "provider": "openai"
}
```

6.2.2 Document Generation (Tool Usage)

The agent automatically invokes tools based on natural language requests.

> **User Query:** "Create a study guide for Microservices and email it to student@example.com."
> **System Action:** 1. Retrieval: Fetches microservices content.
> 2. Generation: Uses `DocumentGenerationTool` to create PDF.
> 3. Delivery: Uses `EmailAutomationTool` to send the file.

7 Conclusion

The **LMA-o** project successfully delivers a comprehensive Software Engineering SME agent. By combining advanced RAG techniques, robust agentic workflows, and a microservices architecture, the system meets all core requirements and implements significant bonus features, providing a highly extensible platform for AI-assisted technical education and problem-solving.