AI Agents Interface for Multi-Model Architecture

Authors*: *Suryakanta Karan<m22aie207@iitj.ac.in> ,Satish Arjun Chilloji <m22aie242@iitj.ac.in>,Kumar Roushan<m22aie243@iitj.ac.in>
GitHub Repository: https://github.com/SuryakantaK-Dev/Major_Project_FMGAI

Abstract

This report outlines the design and implementation of a multi-agent system leveraging generative AI models for handling database queries, document retrieval, and web search tasks. The architecture uses trained models integrated with modular functionalities for SQL generation, document interaction, and web search. The interface simplifies user interaction by routing their inputs to specific large language models (LLMs) suited for the task. This report explains the architecture, components, and their use cases.

1. Introduction

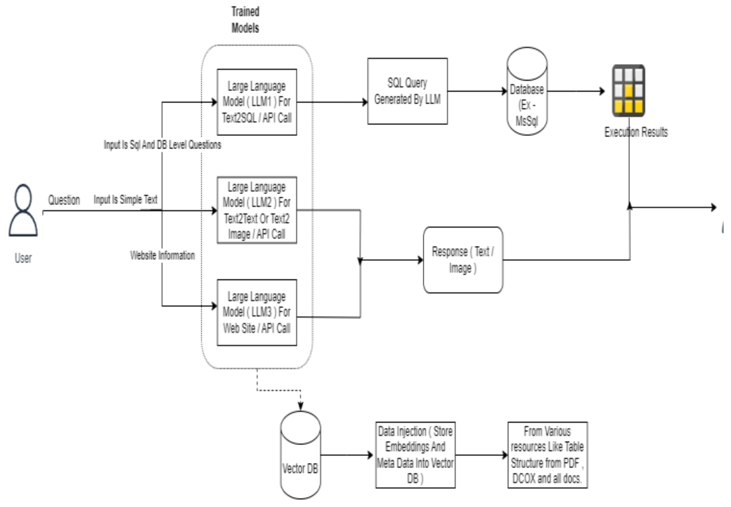
Modern AI systems aim to streamline complex workflows through automation and intelligent interfaces. In this project, the focus is on building a system with three specialized agents, each catering to a specific use case:

- **LLM1** for Text-to-SQL (Text2SQL)
- **LLM2** for document-based question-answering with Retrieval-Augmented Generation (RAG)
- **LLM3** for web search and general-purpose queries

The architecture integrates these agents with a backend database and vector storage for efficient query resolution, presenting a unified interface to users.

2. Architecture Design

The proposed architecture (refer to the figure provided) involves the following core components:


Trained Models
Large Language Model (LLM1) For Text2SQL / API Call
SQL Query Generated By LLM
Database (Ex - MsSql
Execution Results
Input Is Sql And DB Level Questions
Question
Input Is Simple Text
User
Large Language Model (LLM2) For Text2Text Or Text2 Image / API Call
Website Information
Response (Text / Image)
Large Language Model (LLM3) For Web Site / API Call
Vector DB
Data Injection (Store Embeddings And Meta Data Into Vector DB)
From Various resources Like Table Structure from PDF , DCOX and all docs.

1. **User Interface:**
    a. A user submits text queries, which are routed to the appropriate LLM based on their nature (e.g., SQL, document-based, or general web search).
2. **LLM Agents:**
    a. **LLM1 (Text2SQL)**: Converts natural language questions into SQL queries. Example: "What was the last open and close value for Tesla?"
    b. **LLM2 (RAG)**: Handles document-based queries by indexing documents into a vector database and using embeddings for context-aware responses.
    c. **LLM3 (Web Search)**: Retrieves information from web sources for queries involving current events or trends.
3. **Vector Database:**
    a. Stores embeddings and metadata extracted from documents (e.g., PDFs) using a FAISS vector store. Enables efficient similarity search and supports LLM2.
4. **Backend Integration:**
    a. A SQL database stores structured data.
    b. API endpoints manage interactions with the LLMs and vector database.
5. **Execution Workflow:**
    a. Text inputs are pre-processed and routed to the appropriate LLM.
    b. Results are retrieved from the database, vector store, or web and returned to the user.

3. Implementation Details

The implementation comprises two main modules:

1. **Multi-Agent1.py**:
    a. **LLM1 (Text2SQL)**:
        i. Utilizes ConversableAgent from FLAML AutoGen for generating SQL queries based on schema and user inputs.
        ii. Example schema provided for querying stock data.
    b. **LLM2 (RAG)**:

i. Generates context-aware responses from uploaded documents using FAISS and a generative model.

c. **LLM3 (Web Search)**:

i. A specialized agent for web search queries.

2. **App.py**:
   a. Built with Streamlit for an intuitive interface.
   b. Enables document uploads, indexing into the vector database, and querying.
   c. Implements Google Generative AI for embeddings and response generation.

4. Use Cases

1. **Business Intelligence:**
   a. Use LLM1 to extract insights from SQL databases, such as financial trends or product analytics.
2. **Document Analysis:**
   a. Use LLM2 for retrieving specific information from business reports, legal documents, or technical manuals.
3. **Market Research:**
   a. Use LLM3 for fetching real-time data and analyzing trends.

5. Challenges and Solutions

- **Model Configuration:** Configuring multiple LLMs for diverse tasks was streamlined using a common API configuration file.
- **Data Privacy:** Ensuring safe storage and retrieval of embeddings in FAISS using proper deserialization checks.
- **Scalability:** Modular design allows extending the system by integrating new agents for other tasks.

6. Conclusion and Future Work

The system demonstrates a flexible and scalable multi-agent framework for solving diverse queries efficiently. Future enhancements could include:

- Expanding LLM2 to support other file formats like Excel or Word.
- Improving web search capabilities in LLM3 by integrating advanced search APIs.
- Introducing a collaborative agent mechanism where agents interact to answer complex, multi-modal queries.

References

- GitHub Repository: Major Project FMGAI
- FAISS Documentation
- FLAML AutoGen API