Arogo AI – LLM Engineer Intern Assignment

Note: All queries and details are included in the 'FAQs and Details' of this document.

Overview

The objective of this assignment is to design and develop a multi-functional AI assistant leveraging a large language model (LLM). The assistant should be capable of processing various data formats, executing multiple NLP tasks, integrating document retrieval mechanisms, and managing user interactions. The implementation must follow robust software engineering principles, performance optimization strategies, and ethical considerations.
Candidates may use any LLM platform (open-source or commercial) and should present the final demo via either a Streamlit application or an interactive Jupyter notebook (.ipynb format).

Task Details

1. LLM Integration and Abstraction

- LLM Agnostic Design: Implement an abstraction layer to allow easy switching between different LLMs (e.g., OpenAI, Hugging Face, or self-hosted models).
- Configuration Management: Use a configuration file or environment variables for defining API endpoints, authentication keys (if applicable), and model parameters.

2. Data Ingestion and Processing

- Supported Formats: Enable ingestion of CSV, JSON, and plain text files.
- Preprocessing Pipeline: Implement text preprocessing steps such as tokenization, normalization, and filtering to prepare input data for NLP tasks.

- Error Handling: Ensure robust handling of invalid or corrupted data files, providing meaningful logs for debugging.

3. Core NLP Functionalities

The AI assistant should support the following functionalities using the integrated LLM:

- Text Summarization: Generate concise summaries from lengthy text inputs.
- Sentiment Analysis: Classify text sentiment (positive, negative, neutral).
- Named Entity Recognition (NER): Extract entities such as names, locations, and dates.
- Question Answering: Answer questions based on a provided context.
- Code Generation & Assistance: Generate or review Python code snippets based on problem statements.

4. Retrieval-Augmented Generation (RAG)

- Document Ingestion & Indexing: Develop a vector-based document retrieval system for indexing a collection of documents.
- Semantic Search: Implement a mechanism to retrieve the most relevant documents based on user queries.
- Dynamic Prompting: Enhance LLM responses by integrating relevant retrieved documents into the prompts.

5. Conversational Interface and Session Management

- Multi-Turn Dialogue: Maintain context over multiple user interactions.
- Persona Switching: Allow the assistant to adapt different response styles (e.g., technical, professional, casual).
- User Interface: Provide an interactive experience via Streamlit or Jupyter Notebook.

6. Performance Optimization and System Robustness

- Caching Mechanism: Implement caching to minimize redundant LLM calls.
- Prompt Engineering: Optimize prompts to enhance response quality and reduce latency.
- Logging & Monitoring: Maintain logs for API calls, errors, and system performance metrics.

7. Testing, Evaluation, and Documentation

- Unit & Integration Testing: Develop unit tests for core functionalities and integration tests for end-to-end validation.
- Evaluation Framework: Design a method to assess output quality against expected responses.
- Comprehensive Documentation:
    - A README file with setup instructions and project architecture details.
    - Inline code comments explaining complex logic.
    - A brief technical report (2-3 pages) covering:
        - Design decisions and trade-offs.
        - Optimization strategies.
        - Ethical considerations and safeguards.
        - Future improvements.
- Ethical Considerations:
    - Implement content moderation to prevent harmful or biased outputs.
    - Ensure user data privacy and transparency in processing.

Submission Guidelines

A structured Git repository containing:

- **Source Code:** Well-organized, modular, and documented.
- **Video Walkthrough:** A detailed walkthrough video for the entire assignment.
- **README:** Setup instructions, project architecture, and configuration details.
- **Requirements File:** (requirements.txt or equivalent) listing dependencies.
- **Demonstration:** A fully functional Streamlit app or Jupyter Notebook showcasing the assistant.
- **Technical Report:** A document summarizing:
    - Design choices and technical decisions.
    - Optimization and debugging strategies.
    - Ethical considerations and data handling policies.
    - Potential enhancements for future iterations.

Evaluation Criteria

Category	Description
Technical Quality	Efficient and modular LLM integration with well-defined abstractions.
Functionality	Successful implementation of NLP tasks and RAG integration.
Coding Practices	Clean, maintainable, and well-documented code.
Performance	Effective caching, optimized prompts, and efficient retrieval mechanisms.
User Experience	Smooth, interactive, and responsive UI.
Ethical Considerations	Implementation of safeguards against biased/harmful outputs and secure data handling.