## High-Level Design (HLD): Code Reviewer with Llama 3 LLM

### 1. Introduction

1.1 Purpose

This High-Level Design (HLD) document outlines the architectural design and key components of "Code Reviewer," a web application designed to automate and enhance code review using the Llama 3 large language model (LLM). The HLD serves as a blueprint for development, guiding the implementation and ensuring alignment with project goals.

1.2 Scope

The scope of Code Reviewer includes:

* Code Upload and Analysis: Providing an interface for users to upload code files (initially supporting Python, Java, JavaScript) and trigger LLM-based analysis.
* Natural Language Interaction: Enabling users to input questions or context in natural language to guide the code review process.
* LLM-Powered Review Generation: Utilizing the Llama 3 model to analyze code and generate comprehensive review feedback and suggestions.
* Review Presentation: Displaying the review output in a clear, concise, and user-friendly manner through a web interface.
* Direct Integration with Version Control Systems (VCS): While integration is planned, the initial version will focus on file uploads.

1.3 Out of Scope

* Support for All Programming Languages: The initial focus will be on Python, Java, and JavaScript, with plans to expand language support iteratively.
* Advanced Code Visualizations: The initial version will prioritize text-based feedback.

### 2. System Architecture

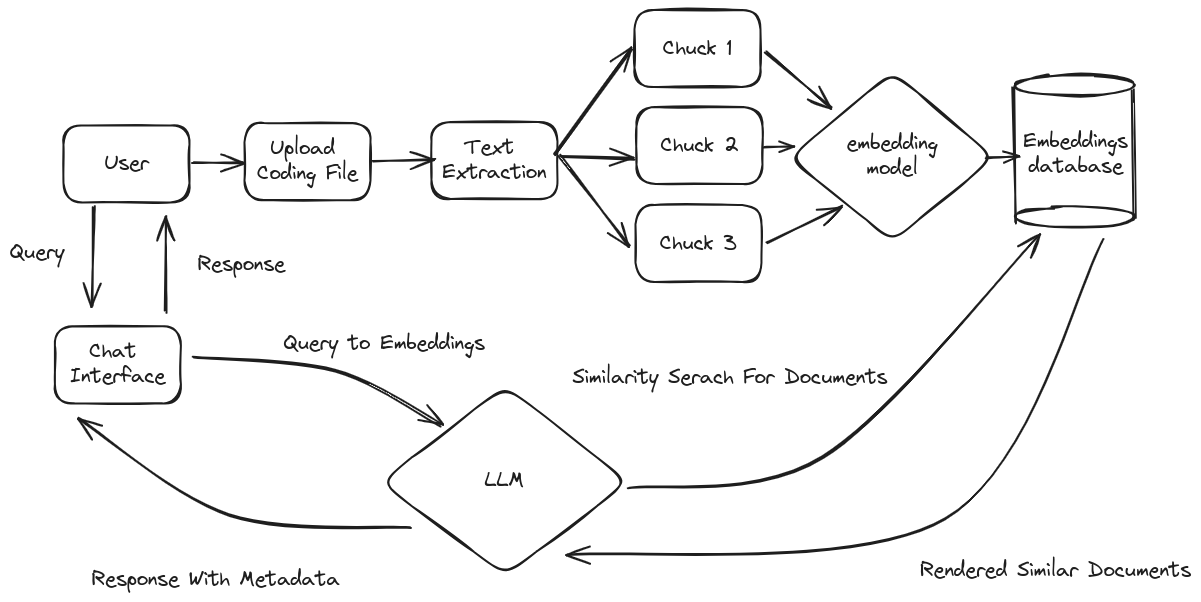
2.1 Overview

Code Reviewer will be implemented as a web application, leveraging a combination of cloud-based services and open-source libraries. The system architecture is designed for scalability, maintainability, and extensibility.

2.2 Components

|  |  |  |
| --- | --- | --- |
| **Component** | **Description** | **Technology** |
| Client (Web UI) | User interface for code upload, input, and review display. | Streamlit |
| Backend (API) | Handles client requests, code processing, and LLM interactions. | Python (e.g., Flask or FastAPI) |
| LLM (Code Analysis) | Analyzes code and generates review feedback using a large language model. | Llama 3 (70B) (Accessed via Groq API) |
| Embeddings | Converts code and queries into numerical representations for LLM understanding. | AI21Embeddings |
| Vector Store | Efficiently stores and retrieves code chunks based on semantic similarity. | FAISS |

2.3 Architecture Diagram



### 3. Functional Design

3.1 Code Upload and Preprocessing

1. User Interaction: The web UI (Streamlit) provides a file upload component for selecting code files (.py, .java, .js).
2. File Handling: The backend API receives the uploaded file and performs basic validation (file type, size).
3. (Optional) Code Preprocessing: The backend may perform syntax highlighting or basic static analysis to prepare the code for LLM analysis.

3.2 Natural Language Input

1. User Interaction: The web UI provides a text input area for users to ask questions or provide context.
2. Query Processing: The backend receives the user's query and prepares it for LLM interaction.

3.3 LLM-Powered Review Generation

1. Embedding Generation: The backend uses AI21Embeddings to create numerical representations of both the code and the user's query.
2. Context Retrieval (Vector Store): If needed, the FAISS vector store is queried using the code embeddings to retrieve relevant code chunks for context.
3. LLM Query: LangChain constructs a prompt incorporating the user's query, relevant code, and instructions for the LLM. The prompt is sent to the Llama 3 model via the Groq API.
4. Response Generation: Llama 3 generates a text response containing the code review feedback and suggestions.

3.4 Review Output

1. Response Parsing: The backend processes the LLM's response, potentially structuring it further for presentation.
2. Review Display: The web UI renders the review output using Streamlit's components. This includes:
   * Summary of findings
   * Detailed explanations
   * Relevant code snippets (potentially with highlighting)

### 4. Non-Functional Requirements

* Performance: The system should provide responses within a reasonable time frame to ensure a smooth user experience.
* Scalability: The architecture should be designed to handle increasing code volume and user traffic.
* Security: Appropriate measures (e.g., secure file handling, API authentication) should be implemented to protect code and user data.
* Maintainability: The codebase should follow best practices for modularity and documentation to facilitate future development and updates.

### 5. Deployment

(Refer to the deployment options outlined in the sample HLD document. For the Code Reviewer project, a cloud-based deployment using a platform like AWS, Google Cloud, or Azure might be suitable for scalability and ease of management.)

Note: Consider adding sections for "Technology Stack" and "Future Enhancements" to further detail the chosen technologies and planned features.