AI-ENHANCED SOFTWARE DEVELOPMENT LIFECYCLE

Project Documentation

1. Introduction

Project Title: AI-Enhanced Software Development Lifecycle: An Educational AI Assistant

Team Member: Mithula.K.M

Team Member: Deepika.K

Team Member: Abinaya.V

Team Member: Akshaya karthika.J

2. Project Overview

Purpose:

The purpose of this project is to demonstrate how an AI-powered assistant can be integrated into the software development lifecycle to streamline two critical phases: research and knowledge gathering. By leveraging the capabilities of a Large Language Model (LLM), this application serves as an educational tool that can explain complex concepts and generate interactive quizzes. This project showcases the potential of AI to enhance the efficiency and accessibility of learning, ultimately accelerating the development process by empowering developers with instant access to knowledge and skill assessment.

Features:

Conversational Interface

- Key Point: Natural language interaction
- Functionality: The Gradio interface allows users to ask questions or input concepts in plain language.

Concept Explanation

- Key Point: Detailed concept understanding
- Functionality: This feature leverages the ibm-granite model to provide detailed explanations of technical or academic concepts, complete with examples. This

helps developers and students quickly grasp new topics without extensive manual research.

Quiz Generator

- Key Point: Interactive skill assessment
- Functionality: This function generates 5 diverse quiz questions (multiple-choice, true/false, short answer) on a given topic. This helps users test their understanding and retain information more effectively.

Gradio UI

- Key Point: User-friendly interface
- Functionality: The application uses a Gradio web interface, which provides a simple and intuitive dashboard with different tabs for each functionality, making it easy for anyone to use.

3. Architecture

Frontend (Gradio):

The frontend is built with Gradio, which provides an interactive web interface for the user. It offers two main tabs: one for "Concept Explanation" and one for "Quiz Generator." The interface handles user input and displays the AI-generated output in real time.

LLM Integration (Hugging Face C IBM Granite):

The core of the application is a pre-trained LLM, ibm-granite/granite-3.2-2b-instruct, accessed through the Hugging Face transformers library. This model is responsible for all natural language understanding and generation tasks.

Core Components:

- AutoTokenizer and AutoModelForCausalLM: These are used to load the pre-trained ibm-granite model and its tokenizer, enabling the application to process text input and generate text output.
- generate_response(prompt, max_length): This is the main function that communicates with the LLM. It takes a user prompt, tokenizes it, generates a response using the model, and then decodes the output. It includes error handling and ensures efficient use of the model.
- concept_explanation(concept): This function crafts a specific prompt asking the model to "Explain the concept of [concept] in detail with an example." It then calls generate_response to get the output.

- quiz_generator(concept): This function formulates a prompt to "Generate 5 quiz questions about [concept] with different quiz types... Give me the answers at the end." It also uses generate_response to get the final output.

The architecture is designed to be lightweight and efficient, with a direct connection between the Gradio UI and the core LLM functions.

4. Setup Instructions

Prerequisites:

- Python 3.9 or later
- pip
- A stable internet connection to download the model from Hugging Face.
- A system with sufficient RAM (at least 8GB recommended) and optionally a CUDA-enabled GPU for faster performance.

Installation Process:

1. Clone the repository: (You would typically use git clone [repository_url], but for a standalone script, you can simply save the code as a .py file.)
2. Install dependencies:
   Open your terminal or command prompt and run the following command to install the required libraries:
   pip install gradio torch transformers
3. Run the application:
   Navigate to the directory where you saved the Python file and run it from your terminal:
   python your_file_name.py
4. Launch the UI:
   The script will automatically launch a Gradio server and provide a local URL (e.g., http://127.0.0.1:7860). Open this URL in your web browser to access the AI assistant.

5. Folder Structure

For this single-script project, a simple structure is sufficient:

your_file_name.py: The main Python script that contains all the application logic, including the Gradio UI and the LLM integration.

6. Running the Application

To start the project, simply run the Python script from your terminal. The Gradio server will launch, and you will be able to interact with the application through your web browser. You can navigate between the two tabs—"Concept Explanation" and "Quiz Generator"—to use the different features.

The application is designed for a seamless, real-time experience, where each query to the AI assistant is processed and the response is displayed dynamically on the frontend.

7. API Documentation

This project is a single, self-contained application rather than a client-server architecture. However, the core functions can be thought of as internal API endpoints.

concept_explanation(concept):

- Input: A string representing the concept to be explained.
- Output: A string containing a detailed explanation with examples, generated by the LLM.

quiz_generator(concept):

- Input: A string representing the topic for the quiz.
- Output: A string containing 5 quiz questions of various types with their answers.

8. Authentication

This version of the project is designed as an open-source, local demonstration and does not require user authentication. It is intended to be run in a private, controlled environment for development and personal use. Future enhancements could include:

- **Token-based authentication:** For a multi-user environment, a system with JWT or API keys could be implemented to manage user access.
- **Role-based access:** Different user roles (e.g., student, instructor) could be created with varying permissions.

G. User Interface

The interface is built with Gradio, prioritizing simplicity and functionality. It is designed to be accessible to technical and non-technical users alike. The UI features:

- **A clear title:** "Educational AI assistant"
- **Tabs:** Separating the two main functionalities to avoid clutter.
- **Text boxes:** For both input (e.g., "Enter a concept") and output (e.g., "Explanation").
- **Buttons:** Labeled "Explain" and "Generate quiz" to initiate the AI functions.

The design focuses on clarity, ensuring that the user flow is intuitive and straightforward.

10. Testing

Testing for this project can be done in multiple phases:

- **Unit Testing:** Verify the functionality of core Python functions like generate_response. This involves checking for proper input handling, error conditions, and output formatting.
- **Manual Testing:** Interact with the Gradio UI to test the chat and quiz functionalities. This ensures that the application responds correctly to user inputs and that the outputs are accurate and well-formatted.
- **Edge Case Handling:** Test with malformed or unusual inputs to see how the application handles them. For example, inputting a very long or nonsensical query to check for stability.

11. Known Issues

- **Hallucinations:** As with all large language models, the ibm-granite model may occasionally generate factually incorrect or nonsensical information.
- **Performance:** Model inference, especially on a CPU, can be slow. The use of a GPU is highly recommended for a better user experience.

12. Future Enhancements

- **User Sessions:** Implement a feature to track user history and save previous conversations.
- **Feedback Mechanism:** Add a way for users to provide feedback on the AI's responses, which could be used to fine-tune the model.
- **API Integration:** Create a backend with a framework like Flask or FastAPI to expose the AI functions as a REST API, allowing them to be integrated into other applications.
- **Support for Multiple Models:** Allow users to select different language models (e.g., from different providers) to compare performance and output.
- **Expanded Functionality:** Add more AI-powered tools relevant to the SDLC, such as code snippet generation, code review assistance, or bug detection.