Building an AI-Powered Code Review System with Open Source Models (Groq AI with `llama3`) and LangSmith for debugging

This project automates the process of reviewing code by using AI to analyze and improve it. Instead of manually checking for best practices, improvements, documentation, and test cases, the system takes a user's code as input and provides structured feedback.

To build this system, I used **LangGraph** to structure the workflow, **Groq AI's `llama3` model** for generating AI-based responses, and **LangSmith** for debugging. The user interface was built using **Streamlit**, allowing users to interact with the system easily.

Why Structure the Process with LangGraph?

Instead of making a single LLM call to review code, I broke the process into multiple steps. This approach helps in several ways:

1. **Better Organization** – Each part of the review (feedback, improvement suggestions, documentation, and test case generation) is handled separately, making the responses more structured and detailed.
2. **Easier Debugging** – Using LangSmith, I can track how the AI processes each step and identify any errors more efficiently.
3. **Improved Accuracy** – Instead of overwhelming the AI with a single request, breaking it down into smaller tasks leads to better and more accurate responses.

Workflow Breakdown

The system follows a clear step-by-step process to ensure a high-quality code review.

1. Code Review

The LLM first analyzes the provided code and gives feedback on its readability, efficiency, security, and overall quality. It highlights issues such as bad coding practices, unnecessary complexity, or potential security risks.

2. Suggesting Improvements

Based on the review feedback, the LLM suggests improvements. This could include optimizing the logic, fixing inefficiencies, or making the code more readable and maintainable.

3. Generating Documentation

Proper documentation is crucial for making code understandable for future developers. The LLM generates docstrings and inline comments, ensuring that each function, class, and complex logic section is well explained.

4. Creating Test Cases

To ensure the reviewed code works as expected, the LLM generates relevant test cases. These test cases help validate different scenarios, making the code more robust and reliable.

User Experience and Interface

I built a simple interface using **Streamlit**, where users can input their code snippet. When they submit the code for review, the system processes it and displays:

- **Review Comments** – AI-generated feedback on the quality of the code.
- **Improvement Suggestions –** Actionable recommendations for making the code better.
- **Generated Documentation –** Proper docstrings and inline comments to enhance readability.
- **Test Cases –** Suggested unit tests for validating the code's functionality.

The interface ensures that even non-technical users can get meaningful insights about their code.

Debugging with LangSmith

These generated results are not always perfect. To improve accuracy, I integrated **LangSmith**, which helps debug and track how the LLMs responds at each stage. This allows me to:

- Monitor the input and output of every step.
- Identify and fix inconsistencies in LLM responses.
- Improve the overall performance of the system.

By using LangSmith, I can refine the workflow and make sure the LLMsI provides high-quality, reliable code reviews.

Final Thoughts

This project automates and streamlines the code review process, making it faster and more efficient. By leveraging LLMs, developers can get instant feedback, meaningful improvements, clear documentation, and proper test cases.

Instead of manually reviewing code, this system allows users to **paste their code, click a button, and receive a structured review within seconds**. With **LangGraph** managing the workflow, **Groq AI** generating the responses, and **LangSmith** ensuring accuracy, the system provides an intelligent and well-organized solution for code reviews.