SUMMER INDUSTRIAL INTERNSHIP CREDIT COURSE

PROJECT REPORT

COURSE NAME: Generative AI in Partnership with Google

ORGANIZATION: SMARTBRIDGE

INSTITUTION: VIT

Team Members:

1. KIRTHIVASAN R - 22BCT0210
2. M. FATHIMA RIZWANA - 23BEE1180
3. KUMARAN R – 22BDS0156
4. SUJITHKUMAR S – 22BCT0040

Project Title: **CodeGenie: AI-Powered Code Generation using CodeLlama**

Problem Statement:

Scenario 1: Automated Code Writing

Description:

CodeGenie allows developers to generate code effortlessly. For instance, a programmer can input a brief description of a desired function, and CodeGenie will provide the complete code snippet, including necessary libraries and comments. This feature enables developers to save time, reduce errors, and focus on more complex tasks by automating repetitive coding tasks.

Scenario 2: Educational Tools

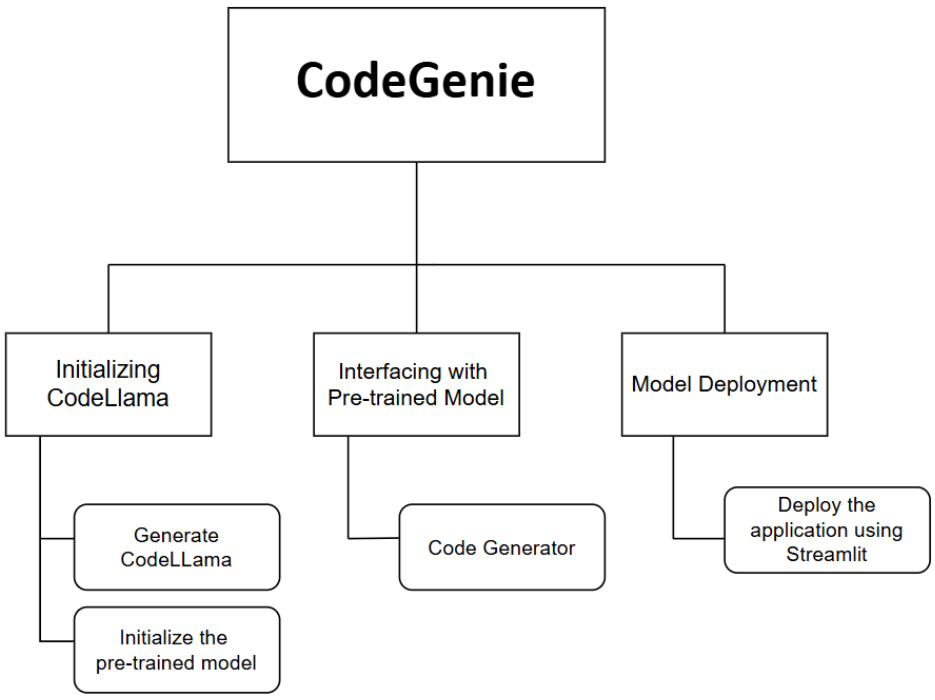
Description:

CodeGenie can be integrated into educational platforms to assist students in learning programming languages. Students can input their code assignments or projects, and CodeGenie will provide insights into code efficiency, syntax correctness, and logical flow. Additionally, it can offer suggestions for improvement and highlight specific parts of the code that are particularly well-written or problematic. This helps students learn to write clean, efficient, and error-free code, improving their coding skills and technic

How the project works?

CodeGenie allows users to enter text through a simple and clear user interface. The input is sent to the backend, where the CodeLlama model processes and understands it. Based on the prompt, CodeLlama generates the appropriate code, which is then displayed on the UI. This smooth input-to-output flow helps users get quick and accurate code solutions with minimal effort. CodeGenie supports developers by generating complete code snippets, including libraries and comments, while helping students learn by analyzing their code for syntax, logic, and efficiency. It offers suggestions and feedback, making coding easier, faster, and more effective for all users.

Steps to Build and Deploy CodeGenie:


CodeGenie
Initializing CodeLlama
Interfacing with Pre-trained Model
Model Deployment
Generate CodeLLama
Initialize the pre-trained model
Code Generator
Deploy the application using Streamlit

Requirements for the Project:

Working Space:

- ❖ **Google Colab:** Google Colab is a free cloud-based platform provided by Google that allows users to write and execute Python code in a web-based Jupyter notebook environment. It supports machine learning, data analysis, and collaboration, providing free access to GPUs and TPUs, making it ideal for students, researchers, and developers.

Foundation Model:

- ❖ **Large Language Model (LLM):** Large Language Models (LLMs) are advanced AI models trained on vast amounts of text data to understand and generate human-like language. They can perform a wide range of tasks such as text generation, summarization, translation, and code generation. In this project, we use an LLM—CodeLlama—that is specifically fine-tuned for programming, enabling it to process natural language prompts and produce accurate, structured code outputs efficiently.

- ❖ **CodeLlama:**

  CodeLlama is a specialized Large Language Model (LLM) developed by Meta, designed specifically for code-related tasks. It is fine-tuned to understand programming languages and generate accurate, high-quality code based on natural language prompts. CodeLlama supports multiple languages like Python, C++, Java, and more, making it useful for developers and learners. In this project, CodeLlama serves as the core engine that processes user input and produces relevant code snippets. Its ability to automate coding tasks helps save time, reduce errors, and enhance productivity for both developers and students.

Model for UI Development:

❖ **Streamlit:**

Streamlit is an open-source Python framework used to create interactive and user-friendly web applications for machine learning and data science projects. In this project, Streamlit is used to build the front-end interface where users can input prompts. It connects seamlessly with the backend powered by CodeLlama, allowing users to view real-time code outputs. Its simplicity and powerful UI components make it ideal for quick deployment and effective user interaction.

❖ **Required Python Libraries:**

- **Transformers:** Provided by Hugging Face, this library allows access to powerful pre-trained transformer models like CodeLlama. It is used for loading tokenizers (AutoTokenizer) and causal language models (AutoModelForCausalLM) to generate code based on user prompts.

*(**Syntax:** from transformers import AutoTokenizer, AutoModelForCausalLM)*

- **Torch:** PyTorch is a deep learning framework that supports tensor operations and model execution on both CPU and GPU. It serves as the backend for running transformer models efficiently.
  *(**Syntax:** import torch)*

- **Time:** A built-in Python module used to measure execution time or introduce delays. It is helpful in performance tracking and benchmarking.
  *(**Syntax:** import time)*

- **OS:** This module provides functions to interact with the operating system. It is useful for handling file paths, environment variables, and directory management within the application.
  *(**Syntax:** import os)*

- **localtunnel**: Creates a secure tunnel to localhost, making a local server accessible over the web. We will be using this to launch our Streamlit app from Google Colab.

Project Structure:

The **CodeGenie** project is built around two key Python files that form the core of the application: app.py and CodeModel.py. These files work together to ensure smooth interaction between the user interface and the backend code generation model.

The **app.py** file serves as the main application driver. It is responsible for rendering the Streamlit-based user interface, taking user inputs (natural language prompts), and sending them to the backend for processing. It also handles the display of the generated code output on the UI, making the overall user experience interactive and efficient.

app() Function Overview:

The app.py file contains the function app(), which defines the core logic for how the user interacts with the CodeGenie interface. It is responsible for managing the entire flow—from accepting user input to displaying the AI-generated code output.

Within this function, Streamlit is used to create a clean and interactive layout, making it simple for users to input prompts. The function captures the prompt through a structured form, ensuring inputs are handled only upon submission. To enhance user experience, a loading spinner provides feedback while the backend model processes the request.Overall, app() ensures a smooth integration of UI and backend functionality in a user-friendly manner.

The **CodeModel.py** file is the backend logic handler, where the CodeLlama model is initialized and managed. It includes the generate_code() function, which takes the user prompt from app.py, tokenizes it using the Hugging Face Transformers library, generates the corresponding code using the pre-trained model, and sends the result back to app.py.

generate_code() Function Overview:

The CodeModel.py file contains the generate_code(prompt) function, which acts as the core engine responsible for generating code from a user-provided prompt. It interfaces with the CodeLlama model, a powerful large language model fine-tuned for code generation tasks.

The function begins by initializing the tokenizer from the pre-trained CodeLlama model, which prepares the input prompt for processing. It then records the current time to track how long the generation process takes.

A text generation pipeline is set up using Hugging Face's transformers library, configured to use floating-point precision and automatically utilize available hardware (GPU or CPU). The function uses specific generation parameters such as low temperature and nucleus sampling to ensure accurate and deterministic output.

Once the code is generated, the function extracts the relevant portion of the response and returns it along with the start and end times. This design ensures both the generated code and the duration can be used effectively in the UI or for performance analysis.

Project Planning and Scheduling

1. Introduction:

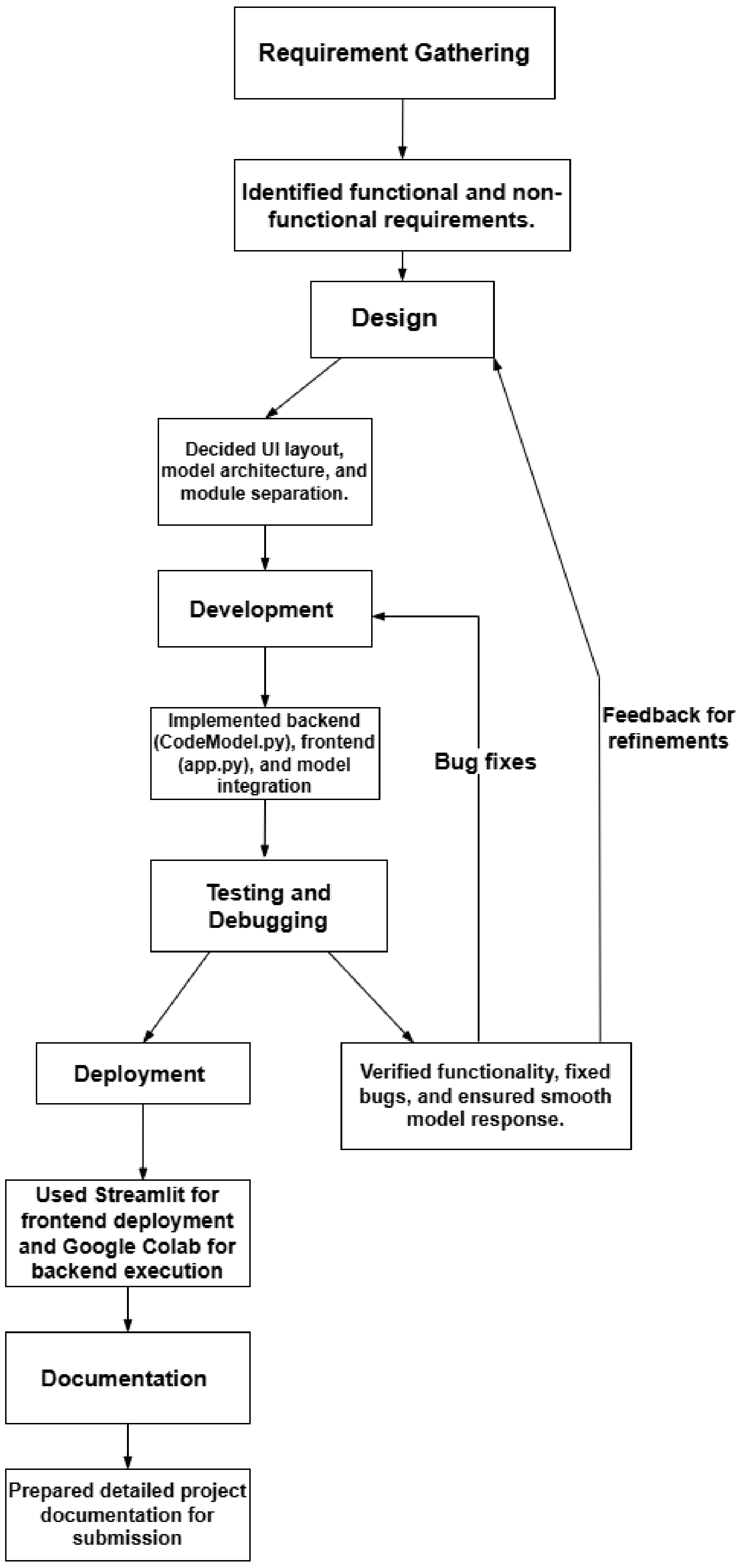
Project planning and scheduling are essential components of successful software development. For the CodeGenie project, a structured planning approach was followed to ensure timely delivery, efficient resource utilization, and clear tracking of progress from initiation to deployment.

2. Planning Objectives

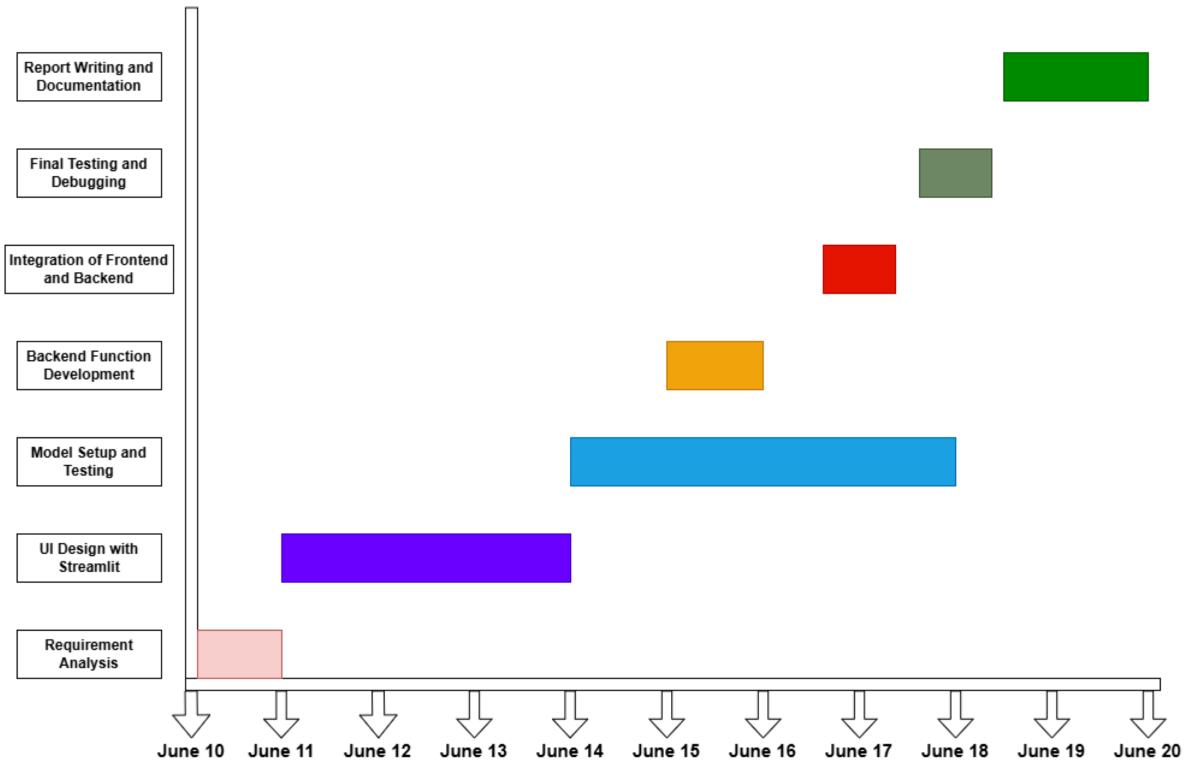
The key goals of project planning were:

i. Define the scope and deliverables of the CodeGenie project.
ii. Identify major development milestones and phases.
iii. Allocate time and resources effectively.
iv. Minimize risks and track progress at every stage.

3. Project Phases


Requirement Gathering
Identified functional and non-functional requirements.
Design
Decided UI layout, model architecture, and module separation.
Development
Implemented backend (CodeModel.py), frontend (app.py), and model integration
Testing and Debugging
Deployment
Verified functionality, fixed bugs, and ensured smooth model response.
Bug fixes
Feedback for refinements
Used Streamlit for frontend deployment and Google Colab for backend execution
Documentation
Prepared detailed project documentation for submission

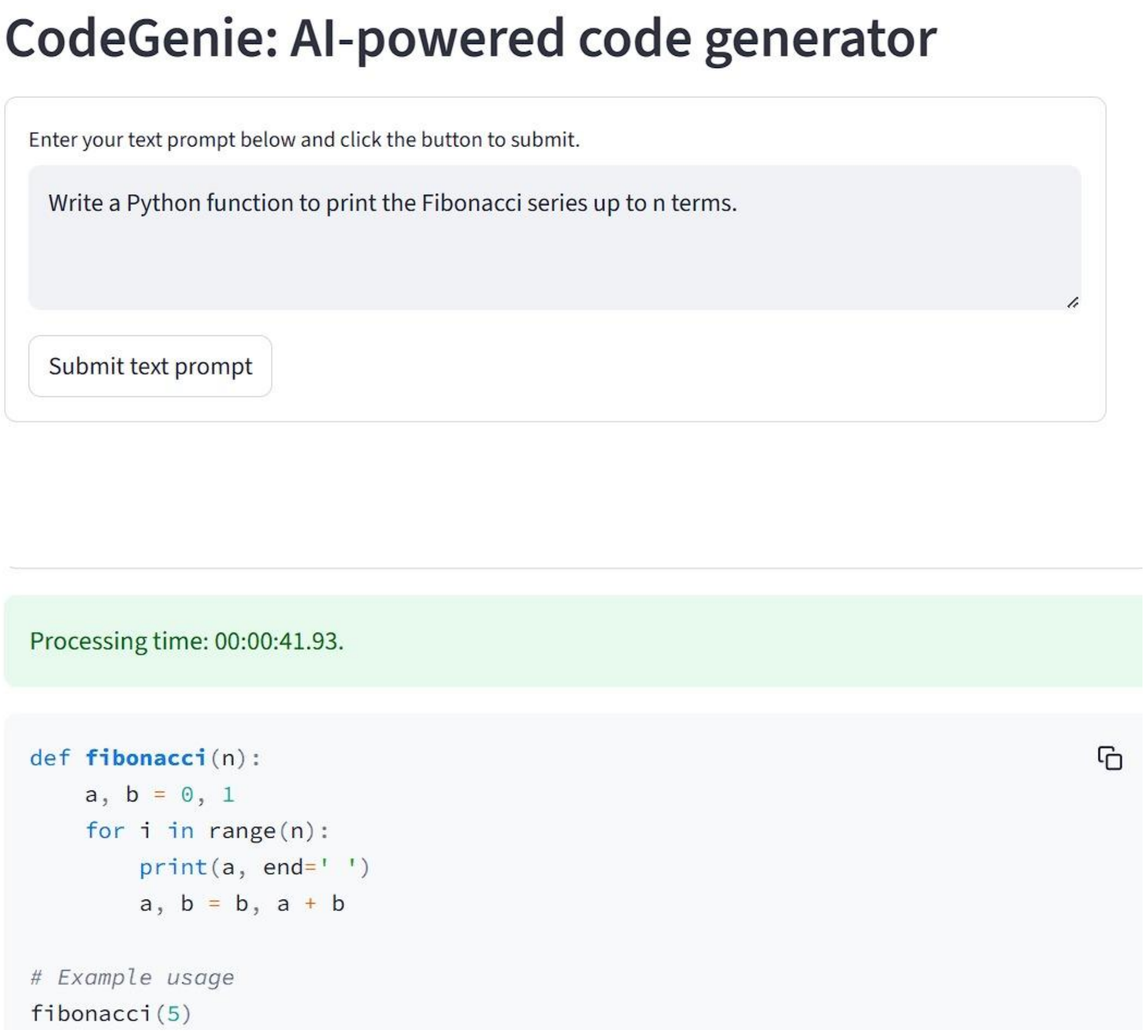
4.Scheduling Breakdown



CODE:

https://github.com/Kirthivasan1/CodeGenerationAI

TESTING:



CodeGenie: AI-powered code generator

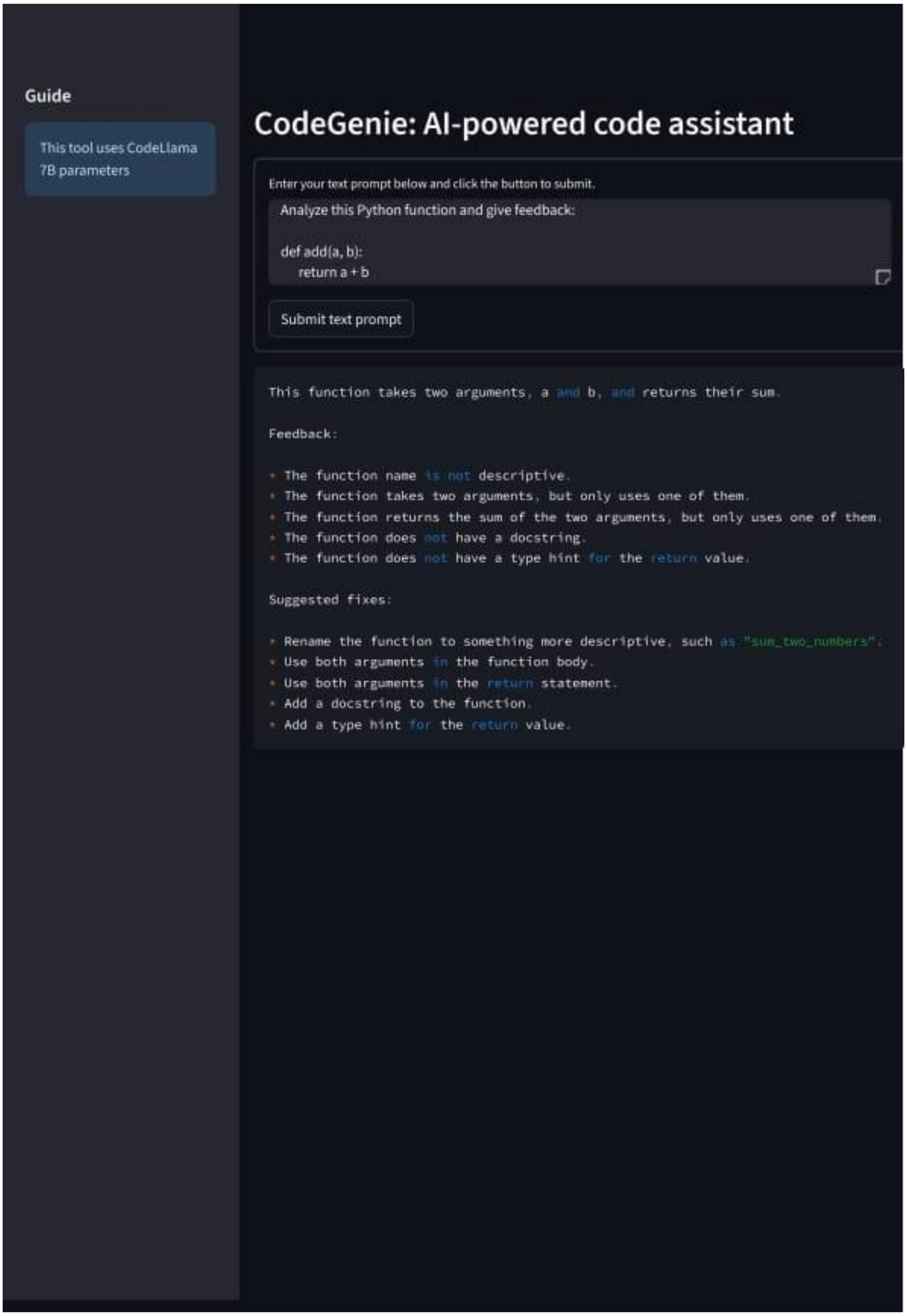
Enter your text prompt below and click the button to submit.

Write a Python function to print the Fibonacci series up to n terms.

Processing time: 00:00:41.93.

```python
def fibonacci(n):
    a, b = 0, 1
    for i in range(n):
        print(a, end=' ')
        a, b = b, a + b

# Example usage
fibonacci(5)
```



CodeGenie: AI-powered code assistant

Enter your text prompt below and click the button to submit.

```
Analyze this Python function and give feedback:

def add(a, b):
    return a + b
```

```
This function takes two arguments, a and b, and returns their sum.

Feedback:

* The function name is not descriptive.
* The function takes two arguments, but only uses one of them.
* The function returns the sum of the two arguments, but only uses one of them.
* The function does not have a docstring.
* The function does not have a type hint for the return value.

Suggested fixes:

* Rename the function to something more descriptive, such as "sum_two_numbers".
* Use both arguments in the function body.
* Use both arguments in the return statement.
* Add a docstring to the function.
* Add a type hint for the return value.
```

Advantages of CodeGenie:

i. **Time-Saving**: Automates code writing, reducing development time significantly for standard or repetitive coding tasks.
ii. **Beginner-Friendly**: Helps students learn programming by providing clean, ready-to-use code from simple text prompts.
iii. **Real-Time Output**: With Streamlit, code is generated and displayed instantly, offering an interactive user experience.
iv. **Supports Multiple Languages**: CodeLlama can handle different programming languages like Python, Java, etc.
v. **Educational Aid**: Can assist learners in understanding logic, syntax, and structure through example-driven learning.
vi. **Modular Design**: Separates UI (app.py) and model logic (CodeModel.py), making the system easy to maintain or upgrade.
vii. **Cloud-Based Execution**: Utilizes Google Colab, so no local system requirements or installations are needed.

Disadvantages of CodeGenie:

i. **Model Limitations:** Accuracy depends on the quality of the model; it may sometimes generate incomplete or incorrect code.
ii. **Internet Dependency:** Requires a stable internet connection for Google Colab and Streamlit to function.
iii. **Limited Customization**: Generated code may not always match specific formatting or style preferences.
iv. **Hardware Constraints**: Running large models on Colab may be slow or restricted due to session limits or GPU availability.
v. **Lack of Deep Understanding**: Users may rely on generated code without fully understanding the logic behind it.
vi. **Security Risk**: If not properly handled, user input may lead to potential misuse or unintended code generation.
vii. **Not Ideal for Complex Projects**: It performs well for simple or moderate tasks, but struggles with highly complex code logic.

Conclusion:

The CodeGenie project successfully demonstrates how large language models like CodeLlama can be integrated with a user-friendly interface using Streamlit to generate code from natural language prompts. By leveraging machine learning and NLP, the system allows users—especially developers and students—to automate repetitive coding tasks, gain instant code suggestions, and enhance productivity.

Through effective model initialization, backend integration, and interactive UI design, CodeGenie showcases the practical potential of AI-powered code generation in both educational and professional environments. The project also highlights how such tools can reduce human error, speed up development, and support learning by offering real-time feedback and structured code output.