CodeGenie: AI-powered code generator using LLM

## Abstract

Genie Code Generator is a Streamlit-based web application that uses state-of-the-art NLP models to generate code snippets from natural language prompts. This tool leverages the 'Salesforce/codegen-350M-multi' model from Hugging Face’s Transformers library to provide language-specific code solutions in Python, Java, C, and C++. It simplifies software development by translating human-readable instructions into syntactically correct code blocks.

## Objective

- To create an AI-driven code generation assistant.  
 - To support developers in rapidly prototyping functions or programs in multiple languages.  
 - To explore and integrate NLP-based transformer models for real-world software development tasks.

## Tools & Technologies

|  |  |
| --- | --- |
| Technology | Purpose |
| Python | Backend logic |
| Streamlit | Frontend UI |
| HuggingFace Transformers | Model hosting and inference |
| PyTorch | Deep learning framework |
| CodeGen (Salesforce) | Pre-trained causal language model |

## Project Structure

Gen\_AI\_CodeGenie/  
 ├── app.py # Streamlit frontend  
 ├── CodeModel.py # AI code generation logic  
 └── codeGenie.ipynb # Notebook to run and test the components

## Team Members and Contributions

* **Chinnola Akash Rao** – **Team Lead, Backend Developer, Deployment & Testing**  
   Led the project architecture and planning. Contributed to backend development (CodeModel.py) and was responsible for integrating and executing the project through CodeGenie.ipynb, including deployment using LocalTunnel in Google Colab.
* **Sneha Tripathi** – **Frontend Developer**  
   Worked on building the user interface in Streamlit (app.py) to collect prompts and display AI-generated code.
* **Lakshya Hirani** – **Frontend Developer**  
   Contributed to UI structure, layout, and Streamlit app interaction elements.
* **N Hari Vyshnavi** – **Backend Developer**  
   Assisted in building and explaining the backend model (CodeModel.py), using the Salesforce CodeGen model to generate code based on natural language prompts.

## Code and Explanation

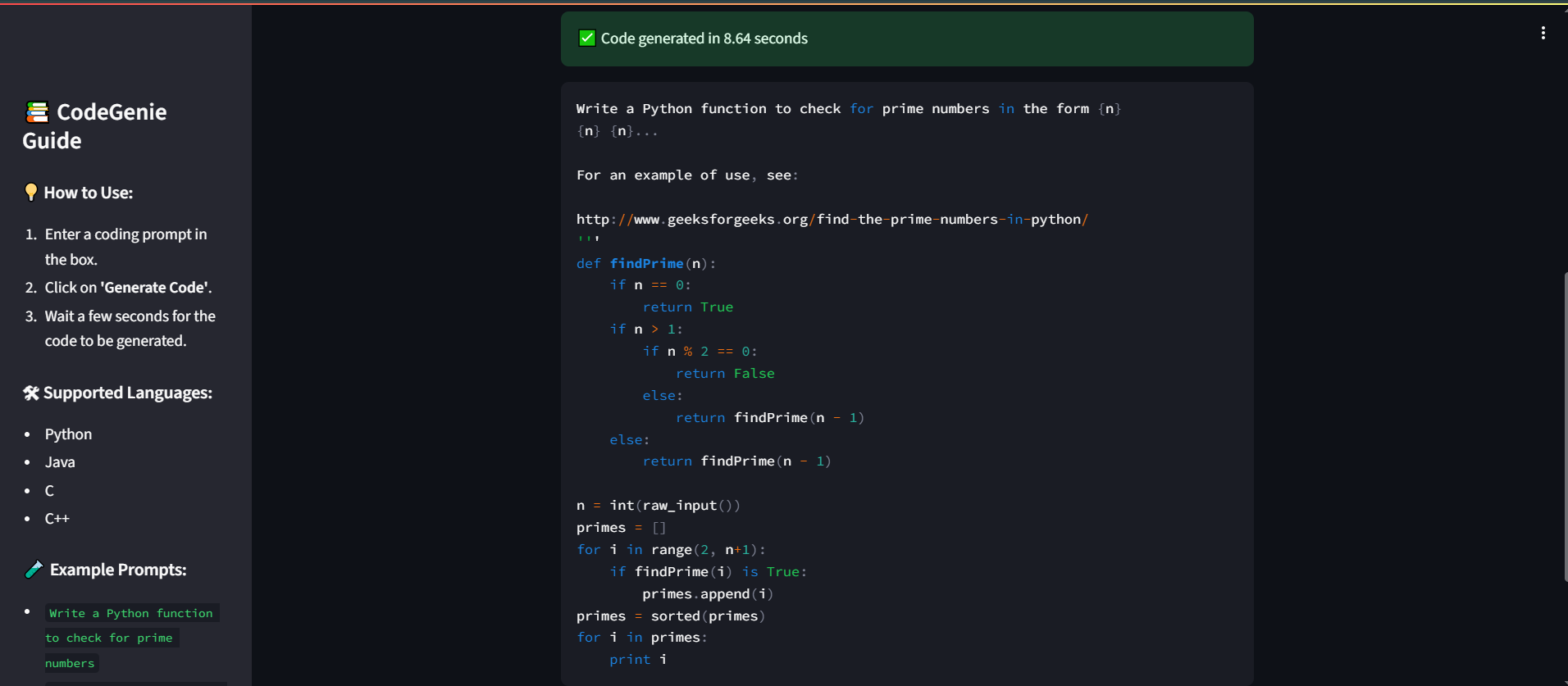
### CodeModel.py – AI Model Inference:

**from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline**  
 **import torch**  
 **import time**  
  
 **def generate\_code(prompt):**  
 **model\_id = "Salesforce/codegen-350M-multi"**  
 **tokenizer = AutoTokenizer.from\_pretrained(model\_id)**  
 **model = AutoModelForCausalLM.from\_pretrained(model\_id)**  
  
 **pipe = pipeline("text-generation", model=model, tokenizer=tokenizer, device=0)**  
  
 **start = time.time()**  
 **sequences = pipe(prompt, max\_length=128, do\_sample=True, temperature=0.7)**  
 **end = time.time()**  
  
 **output = sequences[0]['generated\_text']**  
 **return output, start, end**

### app.py – Streamlit Web Interface:

**import streamlit as st**  
 **import time**  
 **import os**  
 **from CodeModel import generate\_code**  
  
 **st.set\_page\_config(page\_title="CodeGenie", page\_icon="🧠")**  
 **st.sidebar.title("📚 CodeGenie Guide")**  
 **st.sidebar.markdown("""**  
 **### 💡 How to Use:**  
 **1. Enter a coding prompt in the box.**  
 **2. Click on \*\*'Generate Code'\*\*.**  
 **3. Wait a few seconds for the code to be generated.**  
 **""")**  
  
 **st.title("🧠 CodeGenie: AI-Powered Code Generator")**  
 **st.markdown("Turn your ideas into code using an AI model!")**  
 **prompt = st.text\_area("🔤 Enter your code prompt:", height=200)**  
 **if st.button("🚀 Generate Code"):**  
 **if prompt.strip() == "":**  
 **st.warning("⚠️ Please enter a valid prompt.")**  
 **else:**  
 **with st.spinner("Generating code... please wait"):**  
 **try:**  
 **output, start, end = generate\_code(prompt)**  
 **st.success(f"✅ Code generated in {end - start:.2f} seconds")**  
 **st.code(output, language="python")**  
 **except Exception as e:**  
 **st.error(f"❌ An error occurred: {str(e)}")**  
 **if st.button("🛑 Shutdown App"):**  
 **st.warning("Shutting down the Streamlit server...")**  
 **os.system("pkill -f streamlit")**





## Features

- Multi-language support (Python, Java, C, C++).  
 - Fast and readable code generation.  
 - Streamlit shutdown via UI.  
 - Model can be switched with larger CodeGen variants if needed.

## Limitations

- Limited to 128 tokens of code.  
 - Generated code may occasionally need manual corrections.  
 - Performance depends on system’s GPU availability.

## Future Enhancements

- Add language selector for output.  
 - Extend model support to CodeGen-2 or GPT-Code.  
 - Allow saving generated code to files.  
 - Add syntax highlighting based on language.

## Conclusion

The Genie Code Generator project demonstrates the practical integration of large language models in software development workflows. With an intuitive frontend and a powerful transformer backend, it showcases how AI can bridge the gap between intention and implementation in coding.