SmartSDLC AI Enhanced Software Development Lifecycle

1. Introduction

- **Project title** : SmartSDLC AI Assistant
- **Team member** : S.Ajay
- **Team member** : B abishek
- **Team member** : V.Abishek
- **Team member** : K.Adhithya

2. Project overview

- **Purpose:**

· The purpose of this program is to provide an AI-powered tool that assists software developers and analysts in streamlining the software development process. It allows users to upload requirement documents (PDFs) or input raw requirement text, and then automatically analyzes and categorizes them into functional requirements, non-functional requirements, and technical specifications. Additionally, the program can generate ready-to-use source code snippets in multiple programming languages based on user-provided requirements. By combining requirement analysis and code generation in one platform with an easy-to-use Gradio interface, the program reduces manual effort, speeds up development, and improves clarity in software projects

- **Fearture:**

1. PDF Requirement Extraction – Upload PDF documents and automatically extract requirement text.

2. Requirement Analysis – Categorizes input into functional, non-functional, and technical specifications.

3. AI-Powered Code Generation – Generates code snippets in multiple languages (Python, Java, C++, JavaScript, etc.) based on requirements.

4. Dual Input Support – Accepts both direct text input and PDF uploads for flexibility.

5. Interactive UI with Gradio – Easy-to-use tab-based interface for switching between analysis and code generation.

6. Customizable Code Output – Users can choose the target programming language for generated code.

7. GPU/CPU Support – Optimized to run on both GPU (if available) and CPU environments.

8. Error Handling – Provides meaningful error messages when PDFs cannot be read.

9. Real-time Results – Instant display of requirement analysis and generated code within the interface.

10. Shareable Application – Can be shared via public Gradio link for collaborative use.

3. Architecture

1.Input Layer

PDF Upload Module → Uses PyPDF2 to extract raw text from uploaded PDF requirement documents.
Text Input Module → Allows users to manually type or paste software requirements.

2. Processing Layer

Tokenizer & Model Loader → Loads the IBM Granite AI model (AutoTokenizer, AutoModelForCausalLM) for NLP tasks.
Requirement Analysis Module
Takes extracted text/prompt.
Constructs an analysis prompt for the model.
AI organizes requirements into functional, non-functional, technical categories.
Code Generation Module
Takes user requirement + selected language.
AI generates code snippet accordingly.

3. Model Execution Layer

PyTorch Runtime → Executes the Granite AI model either on GPU (CUDA) or CPU depending on availability.
Response Generator → Post-processes AI output (removes prompt text, formats response).

4. Output Layer

Requirements Analysis Output → Displays organized requirements in a textbox.
Code Generation Output → Displays generated code snippet in the chosen programming language.

5. User Interface Layer (Gradio)

Tabs Layout
Code Analysis Tab → For uploading PDFs / entering requirements and viewing categorized analysis.
Code Generation Tab → For requirement input + language selection + generated code display.
Interactive Components → File upload, textbox, dropdown, and buttons connected to backend functions.

4. Setup Instructions

1. System Requirements

OS: Windows, macOS, or Linux
Python: 3.9 or later

Hardware:
CPU (works fine)
GPU (recommended for faster inference, supports CUDA)
Internet connection (required to download model + run Gradio app)

2. Install Required Packages

Open a terminal (or Google Colab cell) and run:

```
pip install torch transformers gradio -q
```

torch → Deep learning framework for running models.
transformers → Hugging Face library to load IBM Granite model.
gradio → To create the web-based interface.

5. Folder Structure

app.py → Single entry point for running the Gradio app.

modules/ → Keeps core logic (analysis, code generation, PDF handling) separate and reusable.

tests/ → Ensures correctness of functions.

docs/ → For proper documentation (can be used in reports/presentations).

assets/ → Sample files & screenshots.

6. Running the Application

5. User Interface Layer (Gradio)

Tabs Layout
Code Analysis Tab → For uploading PDFs / entering requirements and viewing categorized analysis.
Code Generation Tab → For requirement input + language selection + generated code display.
Interactive Components → File upload, textbox, dropdown, and buttons connected to backend functions.

3. AI Model Layer (Core Inference)

IBM Granite Model (granite-3.2-2b-instruct) loaded via transformers.
Runs on GPU (float16) if available, else CPU (float32).
Uses generate() to create responses with controlled sampling (temperature=0.7).

7. API Documentation

1. requirement_analysis(pdf_file, prompt_text)

Description

Analyzes software requirements provided via PDF document or manual text input, and organizes them into categories:

Functional Requirements

Non-Functional Requirements

Technical Specifications

Parameters

Name	Type	Description
pdf_file	file / None	PDF file containing requirements. If None, prompt_text is used.
prompt_text	str	Raw requirement text input. Used only if pdf_file is not provided.

2. code_generation(prompt, language)

Description

Generates source code in a specified programming language based on the provided requirement description.

Parameters

Name	Type	Description

prompt	str	Requirement or feature description for which code is needed.
language	str	Target programming language (Python, Java, JavaScript, C++, etc.).

3. extract_text_from_pdf(pdf_file)

Description

Extracts text from a given PDF file.

Parameters

Name	Type	Description
pdf_file	file	PDF file to extract text from.

4. generate_response(prompt, max_length=1024)

Description

Core function that interacts with the Granite AI model to generate responses based on prompts.

Parameters

Name	Type	Default	Description
prompt	str	required	Input text for the AI model.

max_length	int	1024	Maximum token length for generated response.

8. Authentication

For personal use / testing → Gradio’s auth is enough.

For team use → Use a custom authentication layer (tokens, JWT).

For production / enterprise → Deploy behind a secure API gateway (NGINX, FastAPI with OAuth2, etc.

9. User Interface

1. Main Layout

Header → Title: “AI Code Analysis & Generator” (Markdown heading).

Two Tabs:

1. Code Analysis
2. Code Generation

2. Code Analysis Tab

Left Column (Inputs):
 File Upload → Upload a PDF document containing requirements (.pdf).
 Textbox → Enter requirements manually if no PDF is available.
 Button: “Analyze” → Runs requirement analysis.

Right Column (Output):
 Textbox (Large) → Displays analyzed requirements organized into:
Functional requirements
Non-functional requirements
Technical specifications

3. Code Generation Tab

Left Column (Inputs):
 Textbox → Write the requirement/feature you want code for.
 Dropdown Menu → Choose target programming language (Python, Java, C++, JavaScript, C#, PHP, Go, Rust).
 Button: “Generate Code” → Runs AI-based code generation.

Right Column (Output):
 Textbox (Large) → Displays generated code snippet in the chosen language.

4. Interaction Flow

1. User either uploads a PDF or types requirements.
2. Clicks Analyze → AI organizes requirements.
3. Switch to Code Generation tab.
4. Enters requirement + selects language.
5. Clicks Generate Code → AI outputs code.

5. UI Characteristics

Tabbed Layout → Clear separation between analysis and generation tasks.
Responsive Design → Works in browser, no installation needed.
Shareable Link → If share=True, UI can be accessed by others remotely.

10. Testing

1. extract_text_from_pdf(pdf_file)

Input: valid PDF → should return correct extracted text.
Input: corrupted/empty PDF → should return error message.

2. generate_response(prompt, max_length)

Input: short prompt → should return valid AI-generated text.
Input: very long prompt → should truncate correctly without crashing.

3. requirement_analysis(pdf_file, prompt_text)

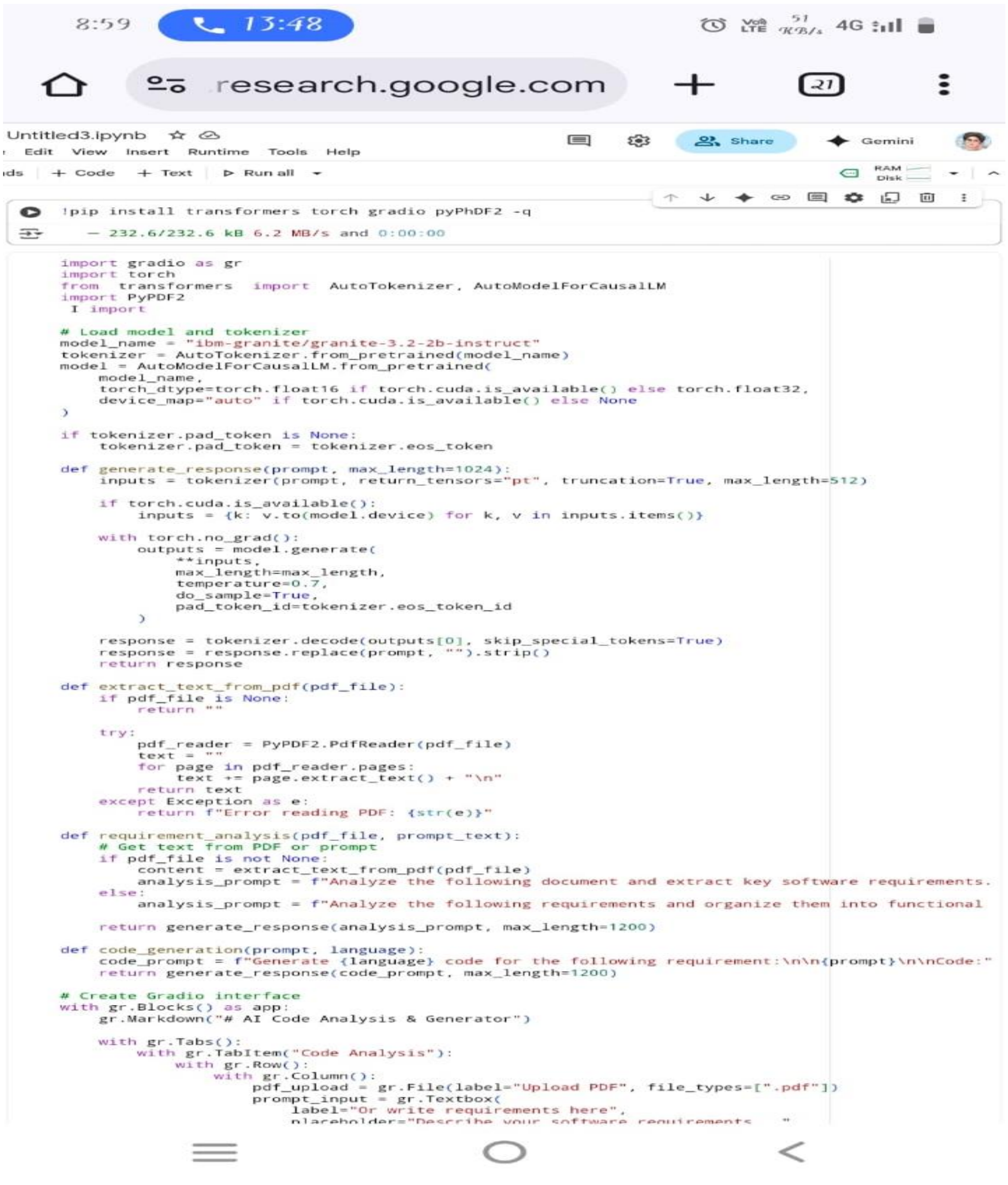
Input: sample text → should categorize into functional, non-functional, technical.
Input: only PDF → should work without prompt_text.

4. code_generation(prompt, language)

Input: valid prompt + "Python" → should return Python code.
Input: unsupported language → should return an error or fallback.

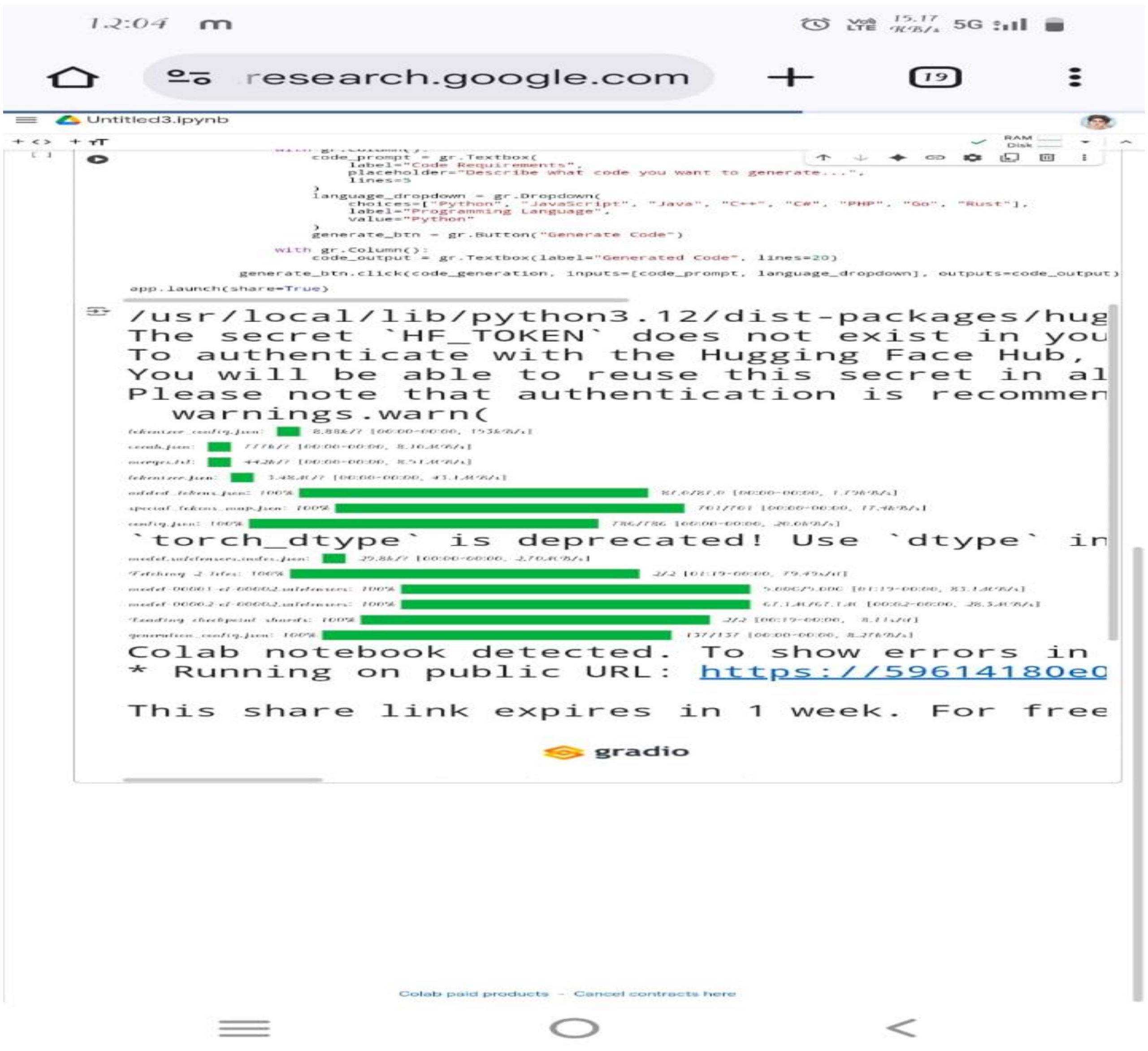
11. screen shots

Input



```
!pip install transformers torch gradio pyPhDF2 -q
  — 232.6/232.6 kB 6.2 MB/s and 0:00:00
```

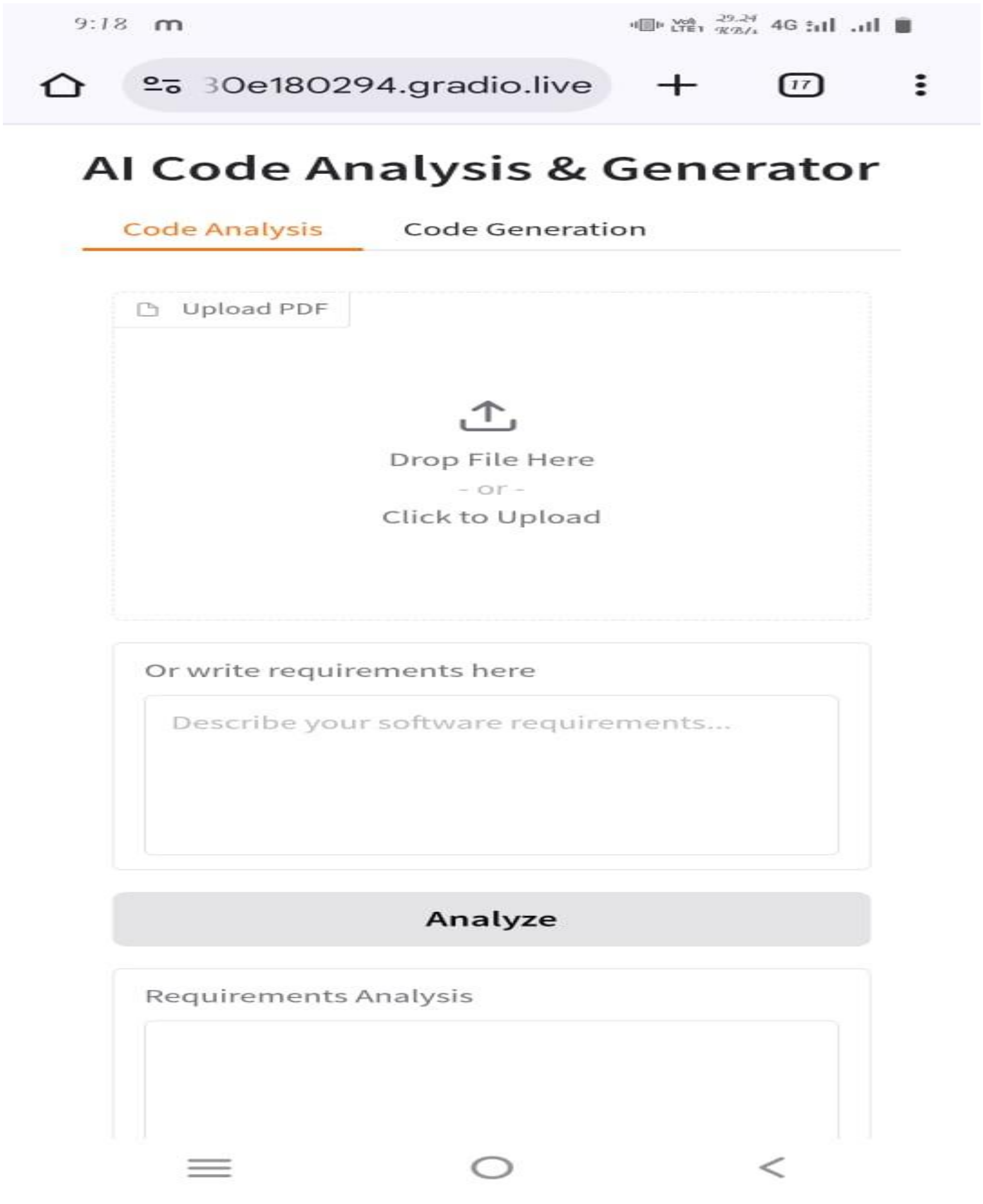
```
import gradio as gr
import torch
from  transformers  import  AutoTokenizer, AutoModelForCausalLM
import PyPDF2
 I import

# Load model and tokenizer
model_name = "ibm-granite/granite-3.2-2b-instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
    device_map="auto" if torch.cuda.is_available() else None
)

if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

def generate_response(prompt, max_length=1024):
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512)

    if torch.cuda.is_available():
        inputs = {k: v.to(model.device) for k, v in inputs.items()}

    with torch.no_grad():
        outputs = model.generate(
            **inputs,
            max_length=max_length,
            temperature=0.7,
            do_sample=True,
            pad_token_id=tokenizer.eos_token_id
        )

    response = tokenizer.decode(outputs[0], skip_special_tokens=True)
    response = response.replace(prompt, "").strip()
    return response

def extract_text_from_pdf(pdf_file):
    if pdf_file is None:
        return ""

    try:
        pdf_reader = PyPDF2.PdfReader(pdf_file)
        text = ""
        for page in pdf_reader.pages:
            text += page.extract_text() + "\n"
        return text
    except Exception as e:
        return f"Error reading PDF: {str(e)}"

def requirement_analysis(pdf_file, prompt_text):
    # Get text from PDF or prompt
    if pdf_file is not None:
        content = extract_text_from_pdf(pdf_file)
        analysis_prompt = f"Analyze the following document and extract key software requirements.
    else:
        analysis_prompt = f"Analyze the following requirements and organize them into functional

    return generate_response(analysis_prompt, max_length=1200)

def code_generation(prompt, language):
    code_prompt = f"Generate {language} code for the following requirement:\n\n{prompt}\n\nCode:"
    return generate_response(code_prompt, max_length=1200)

# Create Gradio interface
with gr.Blocks() as app:
    gr.Markdown("# AI Code Analysis & Generator")

    with gr.Tabs():
        with gr.TabItem("Code Analysis"):
            with gr.Row():
                with gr.Column():
                    pdf_upload = gr.File(label="Upload PDF", file_types=[".pdf"])
                    prompt_input = gr.Textbox(
                        label="Or write requirements here",
                        placeholder="Describe your software requirements..."
```



```
            code_prompt = gr.Textbox(
                label="Code Requirements",
                placeholder="Describe what code you want to generate...",
                lines=5
            )
            language_dropdown = gr.Dropdown(
                choices=["Python", "JavaScript", "Java", "C++", "C#", "PHP", "Go", "Rust"],
                label="Programming Language",
                value="Python"
            )
            generate_btn = gr.Button("Generate Code")

        with gr.Column():
            code_output = gr.Textbox(label="Generated Code", lines=20)

    generate_btn.click(code_generation, inputs=[code_prompt, language_dropdown], outputs=code_output)

app.launch(share=True)
```

```
/usr/local/lib/python3.12/dist-packages/hug
The secret `HF_TOKEN` does not exist in you
To authenticate with the Hugging Face Hub,
You will be able to reuse this secret in al
Please note that authentication is recommen
  warnings.warn(
tokenizer_config.json:  8.88k/? [00:00<00:00, 153kB/s]
vocab.json:  777k/? [00:00<00:00, 8.16MB/s]
merges.txt:  442k/? [00:00<00:00, 8.51MB/s]
tokenizer.json:  3.48M/? [00:00<00:00, 43.1MB/s]
added_tokens.json: 100%  87.0/87.0 [00:00<00:00, 1.79kB/s]
special_tokens_map.json: 100%  701/701 [00:00<00:00, 17.4kB/s]
config.json: 100%  786/786 [00:00<00:00, 20.0kB/s]
`torch_dtype` is deprecated! Use `dtype` in
model.safetensors.index.json:  29.8k/? [00:00<00:00, 2.70MB/s]
Fetching 2 files: 100%  2/2 [01:19<00:00, 79.49s/it]
model-00001-of-00002.safetensors: 100%  5.00G/5.00G [01:19<00:00, 83.1MB/s]
model-00002-of-00002.safetensors: 100%  67.1M/67.1M [00:02<00:00, 28.5MB/s]
Loading checkpoint shards: 100%  2/2 [00:19<00:00,  8.11s/it]
generation_config.json: 100%  137/137 [00:00<00:00, 8.27kB/s]
Colab notebook detected. To show errors in
* Running on public URL: https://59614180e0

This share link expires in 1 week. For free
```

Output



9:18

30e180294.gradio.live

AI Code Analysis & Generator

Code Analysis | Code Generation

Upload PDF

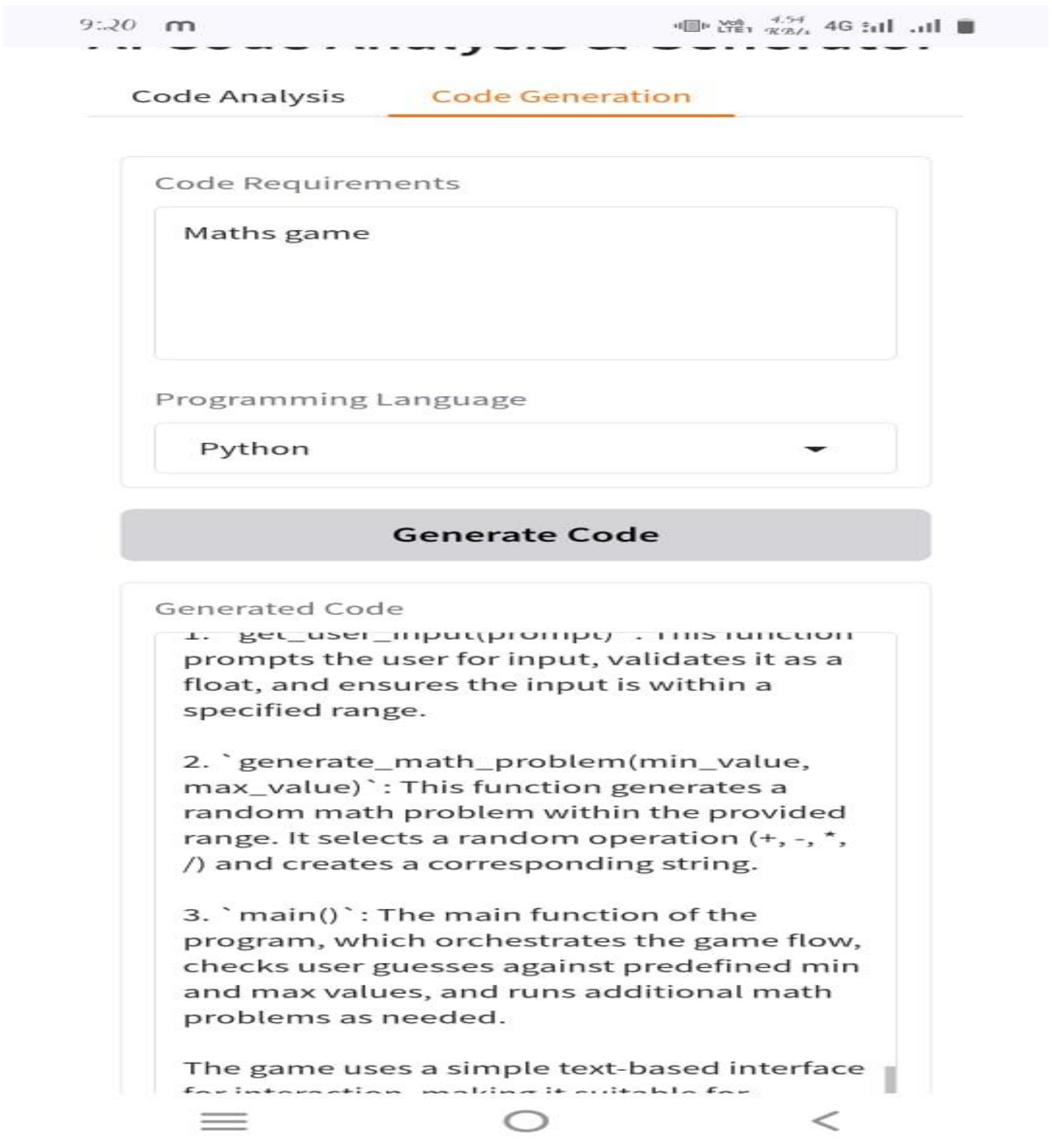
Drop File Here
- or -
Click to Upload

Or write requirements here

Describe your software requirements...

Analyze

Requirements Analysis



Code Requirements

Maths game

Programming Language

Python

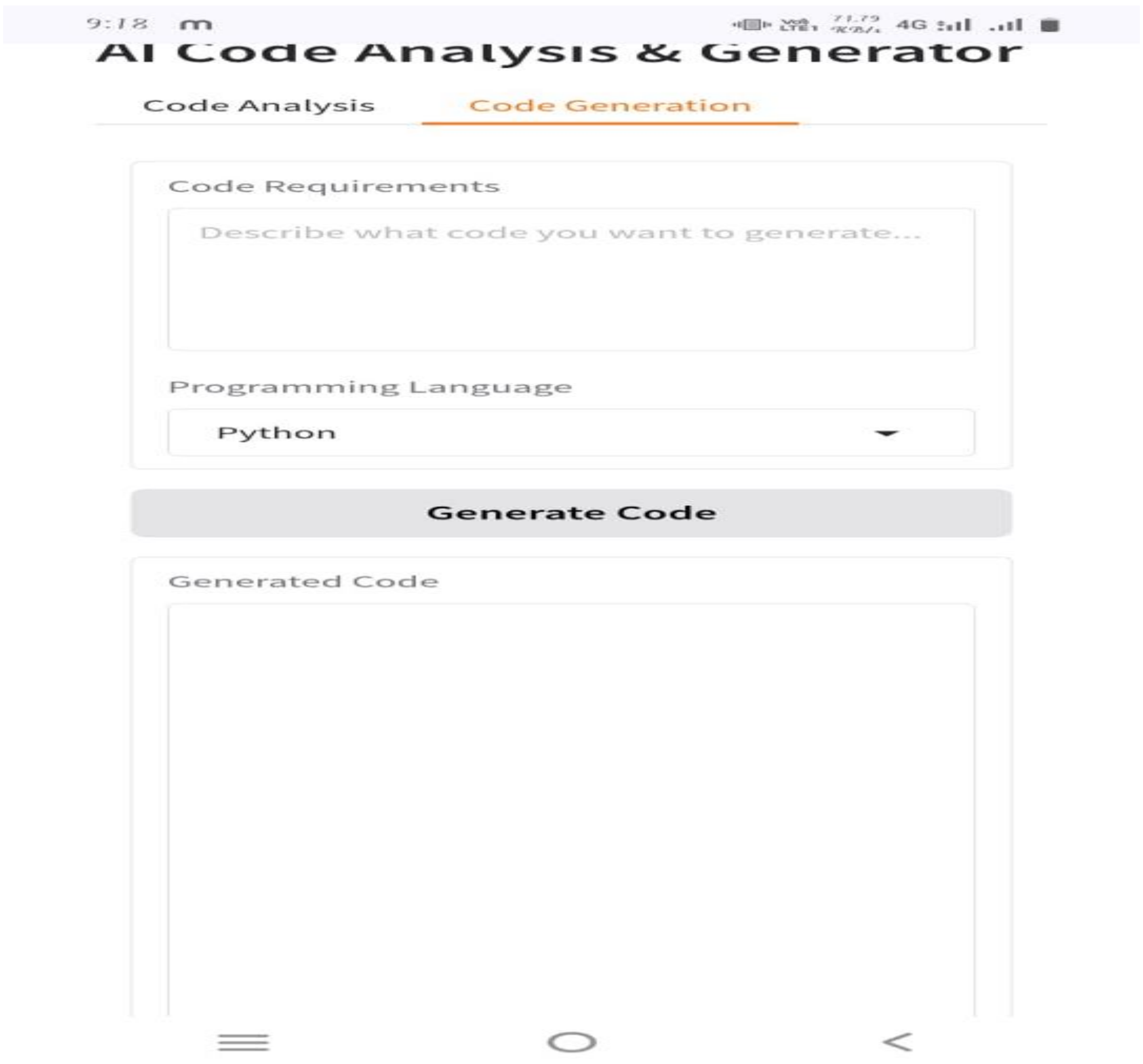
Generated Code

prompts the user for input, validates it as a float, and ensures the input is within a specified range.

2. `generate_math_problem(min_value, max_value)`: This function generates a random math problem within the provided range. It selects a random operation (+, -, *, /) and creates a corresponding string.

3. `main()`: The main function of the program, which orchestrates the game flow, checks user guesses against predefined min and max values, and runs additional math problems as needed.

The game uses a simple text-based interface



12. Known Issues

- ◆ PDF Extraction Limitations
- ◆ AI Model Hallucinations
- ◆ Performance Constraints
- ◆ Limited Error Feedback
- ◆ Authentication Weakness
- ◆ UI Limitations
- ◆ No Persistent Storage
- ◆ Language & Domain Bias

13. Future enhancement

- ◆ Advanced PDF Support
- ◆ Improved Code Generation
- ◆ Requirement Traceability
- ◆ Enhanced User Interface
- ◆ Collaboration Features
- ◆ Authentication & Security
- ◆ Model Enhancements
- ◆ Deployment Improvements
- ◆ Performance Optimization
- ◆ Integration with Development Tools