# SmartSDLC – AI-Enhanced Software Development Lifecycle

# Project Documentation

## 1.Introduction:

- Project title: SmartSDLC – AI-Enhanced Software Development Lifecycle.
- Team member: Lavanya J
- Team member: Kaviya Priya M
- Team member: Keerthana C
- Team member: Keerthana Priya M

## 2.project overview:

### Purpose:

The purpose of this code is to assist in software development by using artificial intelligence to make tasks faster and easier. It helps in analyzing software requirements and converting them into a structured format that is easy to understand. The system also supports automatic code generation based on the given requirements. It reduces manual effort by providing ready-to-use code snippets in different programming languages. This improves productivity and saves time for developers and learners. The project also promotes accuracy in understanding requirements and creating code. Overall, the code serves as a smart assistant that simplifies the software development process.

### Features:

**Code Analysis:**

In code analysis we can upload either pdf or text of our requirements, so that the system will display it based on,

- ➢ functional requirements
- ➢ non-functional requirements

- technical specifications.

**Code Generation*:***

In code generation part we can generate the code in various programming language we needed like

- C
- python
- C++
- Java, etc.

**User Interface:**

The interface is very simple to use two tabs, that is one for

- analyzing the code
- generating the code.

**Customizable*:***

It is customizable because the users can give their own requirements and generate the code based on their preferrable programming language.

**Saves time:**

Helps in quickly analyzing the requirements and generating useful code, and it reduces the manual time.

## 3.Architecture:

**Frontend (Gradio):**

The frontend is developed using Gradio, which provides a simple and interactive web-based interface. Users can easily upload PDFs or type requirements, choose a programming language, and generate results. The interface is organized into two main tabs:

Code Analysis Tab – For uploading documents or entering requirements and extracting functional, non-functional, and technical specifications.

Code Generation Tab – For generating programming code in different languages based on the given requirements.

**Backend (Google Colab + Python):**

The backend runs on Google Colab, where the Python environment handles model loading, request processing, and interaction with the AI model. It processes inputs, communicates with the Granite model, and returns structured results back to the Gradio UI.

**LLM Integration (IBM Granite):**

The project integrates the IBM Granite LLM for natural language understanding and generation. The model is responsible for analyzing software requirements, classifying them into categories, and generating code snippets in multiple programming languages.

**PDF Processing (PyPDF2):**

The system uses PyPDF2 to extract text from uploaded PDF documents. Extracted content is then passed to the Granite model for requirement analysis and processing.

## 4.Setup Instructions:

***Prerequisites:***

- Google account to access the google colab.
- Good internet support to install the models and the libraries from the cloud.
- A hugging face account for loading the IBM granite model.

***Installation process:***

- Open the google colab.

- Create a new note book.
- Change the runtime to t4-GPU.
- In first cell, install the libraries.
- After installation, copy the code into the next cell and run it.

## 5.Folder Structure:

Since the project is run entirely in google colab, there is only one main file that is, smartSDLC.ipynb.

***Structure:***

```
project/
  └─- smartSDLC.ipynb
```

## 6.Running The Application:

Here we run the code in the google colab.

- Execute the code cell with app.launch(share=True).
- Colab will display a public Gradio link like:

  (Running on public URL: https://xxxxx.gradio.live)

- Click the link, the web app opens in a new tab.
- The tab contains two functionalities
- Use the two available functionalities:
  - Code Analysis → Upload a PDF or type requirements → Click Analyze → Get categorized requirements.
  - Code Generation → Enter requirement + choose programming language → Click Generate Code → AI generates the code.

## 7.API Documentation:

Backend APIs available include:

**POST /analyze-requirements** – Accepts either a PDF file or plain text describing software requirements and responds with an AI-generated analysis organized into functional, non-functional, and technical specifications.

**POST /generate-code** – Accepts a requirement description along with a selected programming language and returns AI-generated source code based on the input.

**POST /extract-pdf-text** – Accepts a PDF file and returns the extracted plain text for further processing or analysis.

Each endpoint is powered by the IBM Granite model through the Hugging Face API and tested within the Google Colab environment for easy inspection and quick trial during development.

## 8.User Interface:

- **Google Colab Notebook** – The entire project runs inside a single Colab notebook with a Gradio interface.
- Two Tabs in Gradio are:
  - ➢ Code Analysis – In this users can upload a PDF or type requirement text so that the AI analyzes and organizes it into functional, non-functional, and technical specifications.
  - ➢ Code Generation – Users enter a requirement and select a programming language by that the AI generates the corresponding code.
- **Textboxes for Input/Output** – Simple boxes to type requirements and display analyzed results or generated code.
- **Buttons to Trigger Actions** – "Analyze" and "Generate Code" buttons execute the AI model and show the output.

- **Public/Local Link** – Gradio provides a shareable link so, the interface can be accessed from any browser or device.

## 9.Testing:

**Function Testing** – Checks that the Code Analysis tab correctly extracts and organizes requirements from both PDF uploads and manual text input.

**Code Generation Testing** – Verify that the Code generation tab produces code in different languages like Python, JavaScript, Java, C, C++ based on the user's requirement.

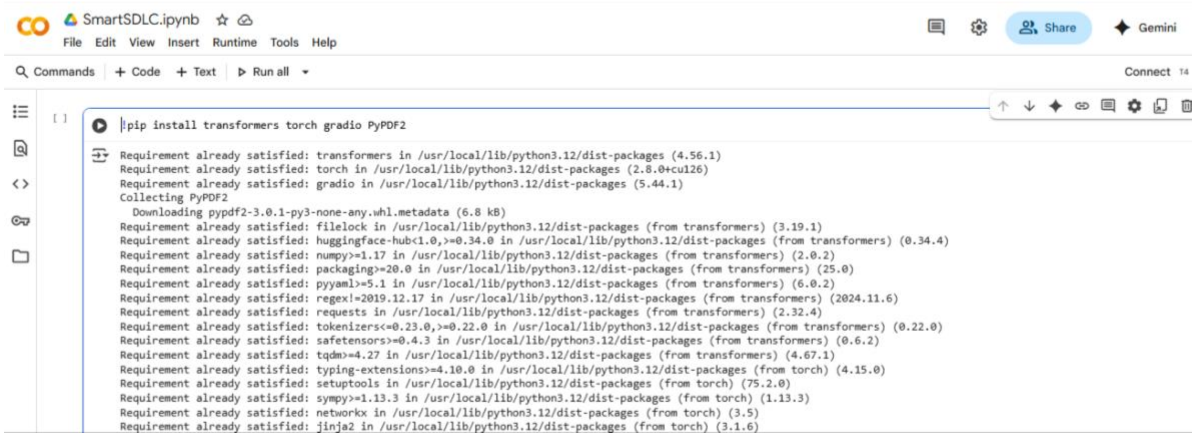
**Model Response Check** – Confirmed that the AI model provides meaningful and readable output without crashes.

**Interface Testing** – Ensured all buttons ("Analyze" and "Generate Code") work properly and display results in the output textboxes.

**Cross-Device Access** – Tested the public Gradio link on mobile and desktop browsers to confirm smooth operation

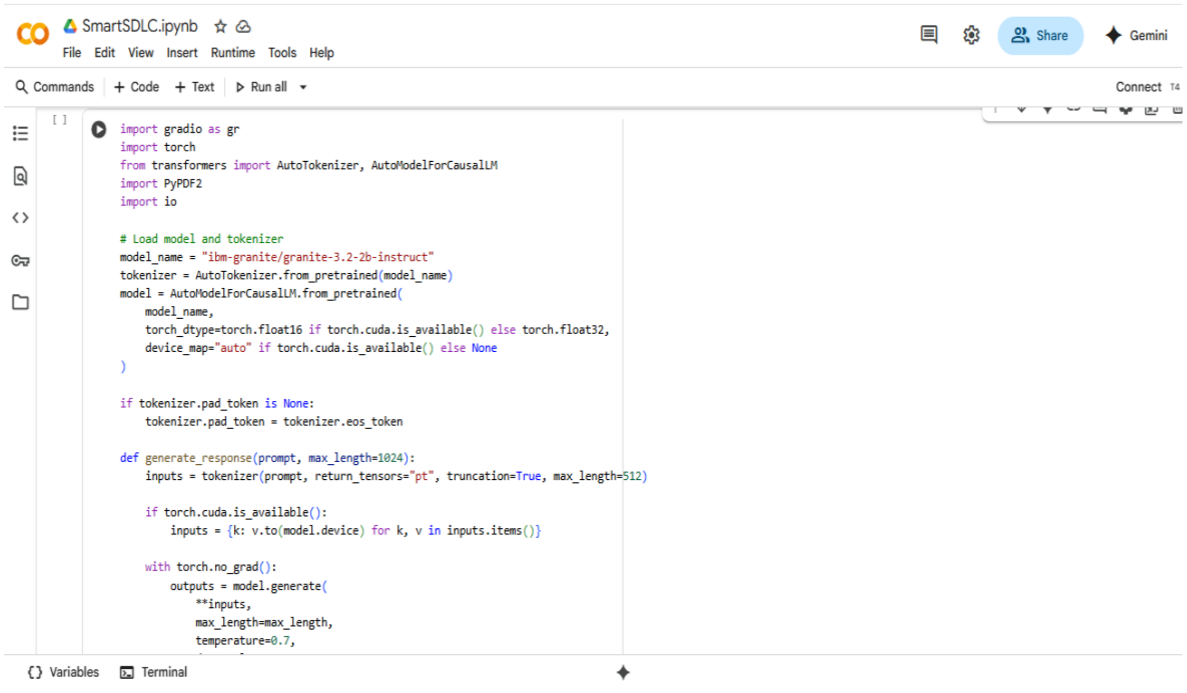
## 10.Screenshots:

***Program:*** First Cell,

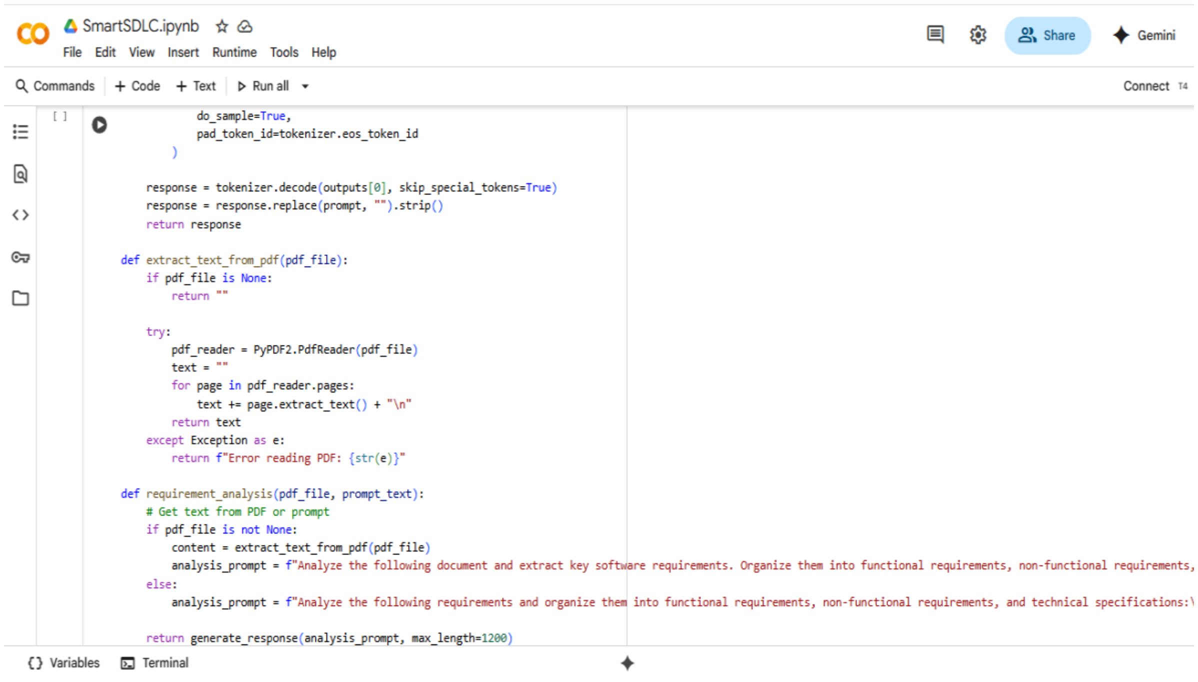


```
!pip install transformers torch gradio PyPDF2

Requirement already satisfied: transformers in /usr/local/lib/python3.12/dist-packages (4.56.1)
Requirement already satisfied: torch in /usr/local/lib/python3.12/dist-packages (2.8.0+cu126)
Requirement already satisfied: gradio in /usr/local/lib/python3.12/dist-packages (5.44.1)
Collecting PyPDF2
  Downloading pypdf2-3.0.1-py3-none-any.whl.metadata (6.8 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from transformers) (3.19.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.34.0 in /usr/local/lib/python3.12/dist-packages (from transformers) (0.34.4)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.12/dist-packages (from transformers) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from transformers) (25.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.12/dist-packages (from transformers) (2024.11.6)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from transformers) (2.32.4)
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in /usr/local/lib/python3.12/dist-packages (from transformers) (0.22.0)
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.12/dist-packages (from transformers) (0.6.2)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.12/dist-packages (from transformers) (4.67.1)
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.12/dist-packages (from torch) (4.15.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from torch) (75.2.0)
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-packages (from torch) (1.13.3)
Requirement already satisfied: networkx in /usr/local/lib/python3.12/dist-packages (from torch) (3.5)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from torch) (3.1.6)
```

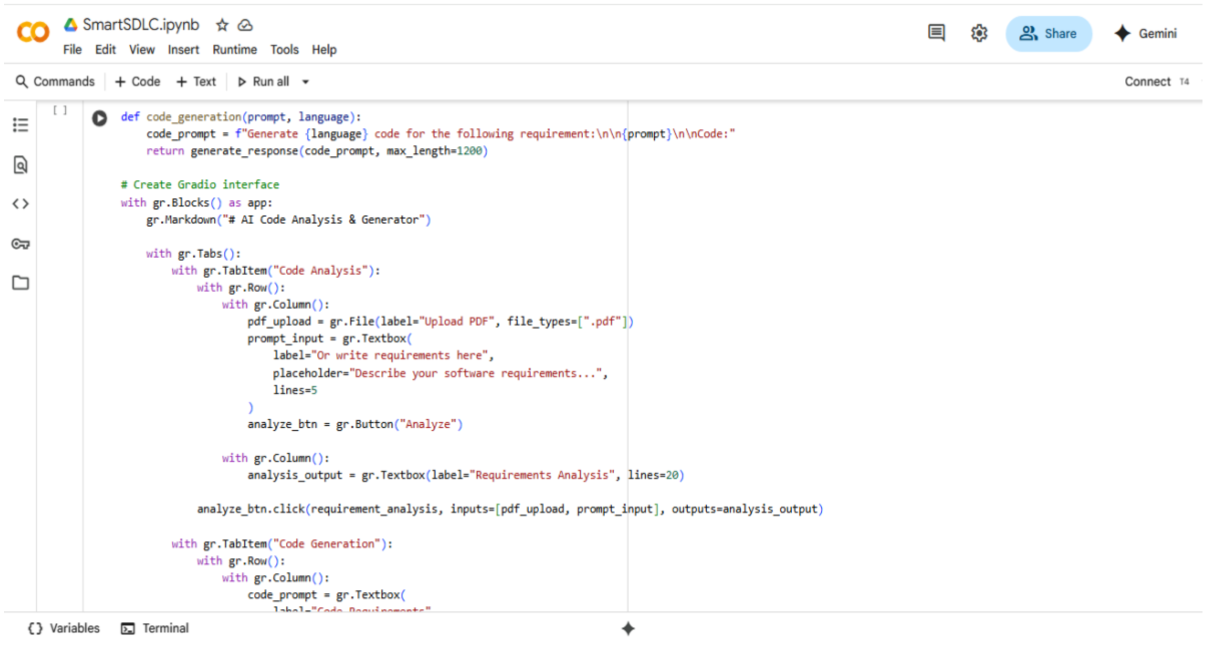
Second cell,



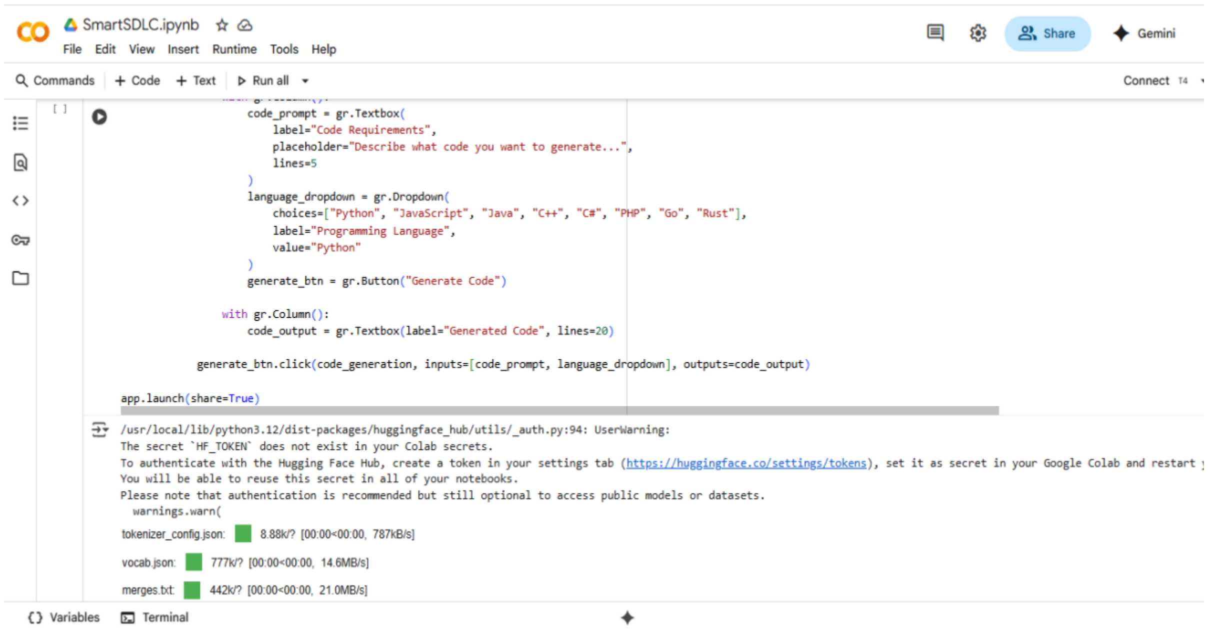
```python
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
import PyPDF2
import io

# 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,
```



```python
            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. Organize them into functional requirements, non-functional requirements,
    else:
        analysis_prompt = f"Analyze the following requirements and organize them into functional requirements, non-functional requirements, and technical specifications:\

    return generate_response(analysis_prompt, max_length=1200)
```

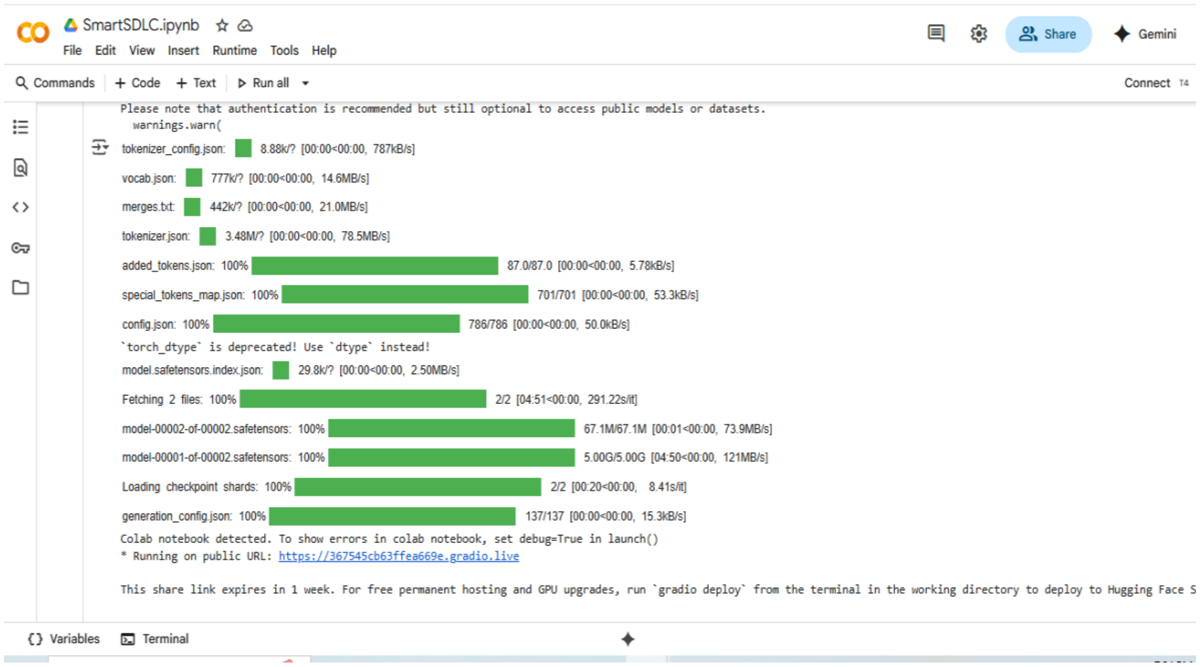


```
def code_generation(prompt, language):
    code_prompt = f"Generate {language} code for the following requirement:\n\n{prompt}\n\nCode:"
    return generate_response(code_prompt, max_length=1200)

# 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...",
                        lines=5
                    )
                    analyze_btn = gr.Button("Analyze")

                with gr.Column():
                    analysis_output = gr.Textbox(label="Requirements Analysis", lines=20)

            analyze_btn.click(requirement_analysis, inputs=[pdf_upload, prompt_input], outputs=analysis_output)

        with gr.TabItem("Code Generation"):
            with gr.Row():
                with gr.Column():
                    code_prompt = gr.Textbox(
```



```
                    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)
```

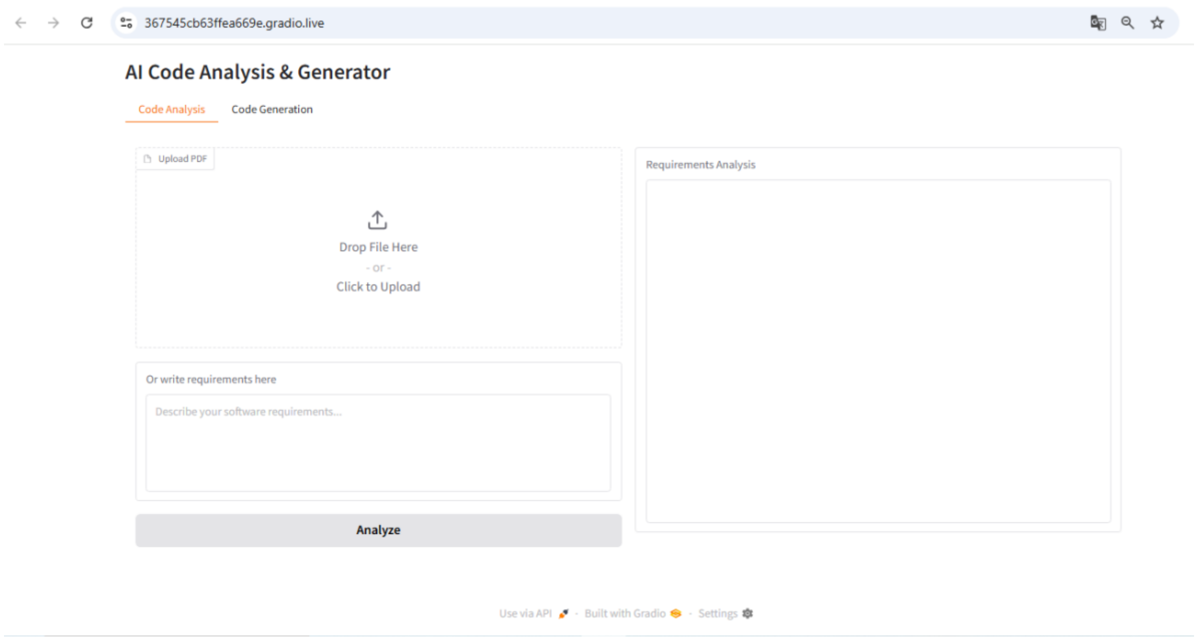
```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 🟩 8.88k/? [00:00<00:00, 787kB/s]
vocab.json: 🟩 777k/? [00:00<00:00, 14.6MB/s]
merges.txt: 🟩 442k/? [00:00<00:00, 21.0MB/s]
```



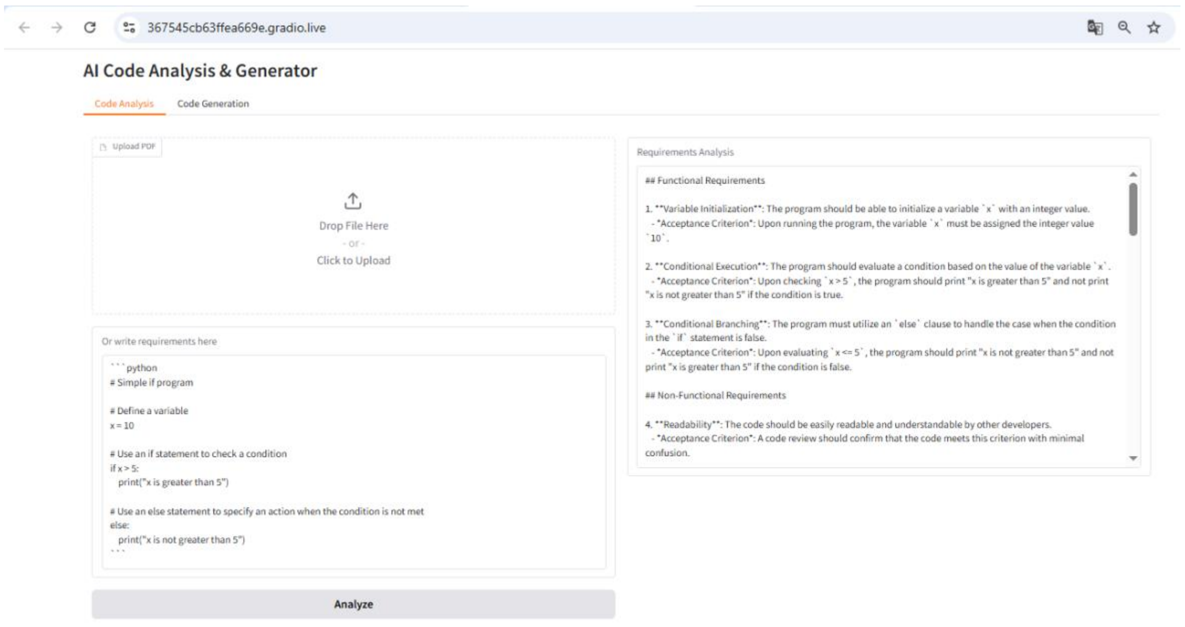
```
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json:     8.88k/? [00:00<00:00, 787kB/s]
vocab.json:     777k/? [00:00<00:00, 14.6MB/s]
merges.txt:     442k/? [00:00<00:00, 21.0MB/s]
tokenizer.json:     3.48M/? [00:00<00:00, 78.5MB/s]
added_tokens.json: 100%     87.0/87.0 [00:00<00:00, 5.78kB/s]
special_tokens_map.json: 100%     701/701 [00:00<00:00, 53.3kB/s]
config.json: 100%     786/786 [00:00<00:00, 50.0kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors.index.json:     29.8k/? [00:00<00:00, 2.50MB/s]
Fetching 2 files: 100%     2/2 [04:51<00:00, 291.22s/it]
model-00002-of-00002.safetensors: 100%     67.1M/67.1M [00:01<00:00, 73.9MB/s]
model-00001-of-00002.safetensors: 100%     5.00G/5.00G [04:50<00:00, 121MB/s]
Loading checkpoint shards: 100%     2/2 [00:20<00:00, 8.41s/it]
generation_config.json: 100%     137/137 [00:00<00:00, 15.3kB/s]
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://367545cb63ffea669e.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face S
```

## ***Output:***

first view of the screen,


367545cb63ffea669e.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
Use via API · Built with Gradio · Settings

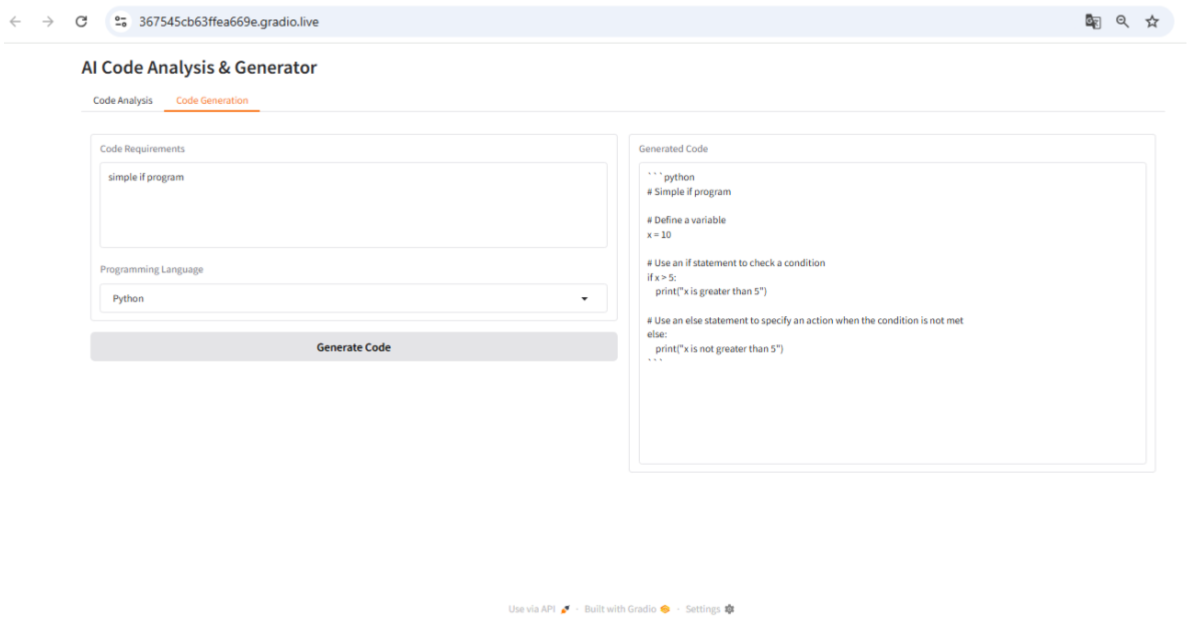
Second view of the output screen, **Code Analysis**.



~~~
```python
# Simple if program

# Define a variable
x = 10

# Use an if statement to check a condition
if x > 5:
   print("x is greater than 5")

# Use an else statement to specify an action when the condition is not met
else:
   print("x is not greater than 5")
```
~~~

```
## Functional Requirements

1. **Variable Initialization**: The program should be able to initialize a variable `x` with an integer value.
  - *Acceptance Criterion*: Upon running the program, the variable `x` must be assigned the integer value
`10`.

2. **Conditional Execution**: The program should evaluate a condition based on the value of the variable `x`.
  - *Acceptance Criterion*: Upon checking `x > 5`, the program should print "x is greater than 5" and not print
"x is not greater than 5" if the condition is true.

3. **Conditional Branching**: The program must utilize an `else` clause to handle the case when the condition
in the `if` statement is false.
  - *Acceptance Criterion*: Upon evaluating `x <= 5`, the program should print "x is not greater than 5" and not
print "x is greater than 5" if the condition is false.

## Non-Functional Requirements

4. **Readability**: The code should be easily readable and understandable by other developers.
  - *Acceptance Criterion*: A code review should confirm that the code meets this criterion with minimal
confusion.
```

Third view of the screen, **Code Generation**.



~~~
```python
# Simple if program

# Define a variable
x = 10

# Use an if statement to check a condition
if x > 5:
   print("x is greater than 5")

# Use an else statement to specify an action when the condition is not met
else:
   print("x is not greater than 5")
```
~~~

## 11.Known Issues:

- May run slowly if internet is weak or model size is large.
- Sometimes generates inaccurate or incomplete analysis/code.
- No login or authentication, so anyone with the link can access.
- Limited to languages supported by the model.
- Requires Google Colab to run, so it can’t work offline.

## 12.Future Enhancements:

Some of the future enhancement that can be added are,

- Add user authentication (login system) for better security.
- Provide a download option to save generated code or analysis.
- Improve the AI to give more accurate and detailed results.
- Allow direct deployment as a website or mobile app for easier access.
- Improve UI with better formatting and themes.
- Add support for more file format like(DOCX, TXT).