Summary Document: AI Code Analysis & Generator

Overview

The provided Python script builds an **interactive AI-powered tool** using **Gradio**, **PyTorch**, and **Hugging Face Transformers**. The tool enables users to:

- **Analyze software requirements** (from PDF files or text input).
- **Generate code automatically** based on natural language descriptions.

The underlying AI model is ibm-granite/

granite-3.2-2b-instruct, a causal language model optimized for instruction-following tasks.

Key Components

1. Model and Tokenizer Initialization

- Loads the **Granite model** (ibm-granite/granite-3.2-2b-instruct) and its tokenizer.

- Configures the model to use:

- **GPU with float16** if CUDA is available.

- **CPU with float32** otherwise.

Ensures that the tokenizer has a valid

padding token.

2. Core Functions

a. generate_response(prompt, max_length=1024)

- Tokenizes the input prompt.
- Sends it to the model for generation.
- Decodes and cleans the response.
- Returns AI-generated text.

b. extract_text_from_pdf(pdf_file)

- Reads uploaded PDF files using PyPDF2.

- Extracts text from each page.
- Handles errors gracefully (e.g., corrupted or unreadable PDFs).

c. requirement_analysis(pdf_file, prompt_text)

- If a PDF is uploaded → extracts content and creates an **analysis prompt**.
- If no PDF is uploaded → uses the given text requirements.
- Calls the model to organize requirements into:
- Functional requirements

- Non-functional requirements
- Technical specifications

d. code_generation(prompt, language)

- Creates a **code generation prompt** tailored to the selected programming language.
- Asks the model to generate corresponding code.

3. Gradio User Interface

The app is built using gr.Blocks() and provides two **tabs**:

Tab 1: Code Analysis

- Inputs:
- PDF upload (.pdf files).
- Textbox for requirement input.
- Action:
- Button (Analyze) triggers requirement extraction.
- Output:
- Textbox displaying organized requirements.

Tab 2: Code Generation

- Inputs:

- Textbox describing code requirements.
- Dropdown to choose a programming language (Python, JavaScript, Java, C++, C#, PHP, Go, Rust).
- Action:
- Button (Generate Code) triggers code generation.
- Output:
- Textbox showing generated code.

4. App Deployment

- Runs locally and provides a **shareable public URL** (share=True).

This allows remote usage without hosting setup.

Purpose & Use Cases

- **Requirement Analysis**: Helps software engineers and analysts extract structured requirements from documents.

- **Automatic Code Generation**: Assists developers by converting natural language requirements into working code snippets.

- **Education & Training**: Useful for students learning software engineering concepts and programming.

Limitations

- Output quality depends on the AI model’s training and may need human review.

- PDF extraction accuracy varies based on formatting.

- Large or complex requirements may exceed token limits.

✅ **In short**:
This script delivers an **AI assistant for software engineering tasks**, combining **requirement analysis** and **code generation** in a simple **Gradio web app**.