CodeDocAI: Automated Code Documentation Assistant using Gemini Model

Project Description

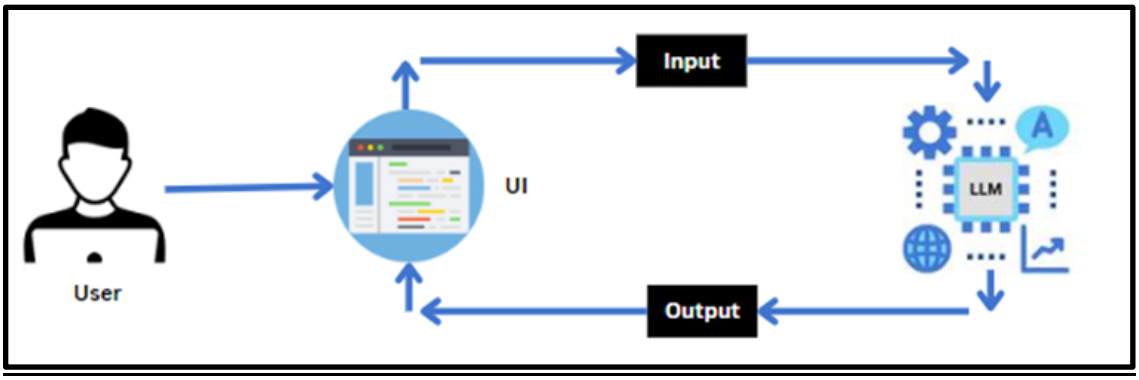
CodeDocAI is an advanced project powered by a Large Language Model (LLM) designed to seamlessly generate comprehensive and accurate documentation from code snippets. This innovative system aims to simplify the documentation process by allowing developers to obtain detailed explanations of their code, enhancing understanding and maintainability without needing to write extensive comments manually. CodeDocAI can be utilized across various scenarios, providing robust solutions to different user needs.

Scenario 1: Software Development
CodeDocAI allows software developers to obtain detailed documentation by simply inputting their code. For instance, a developer can input a function, and CodeDocAI will generate an explanation of its purpose, parameters, and return values. This feature enables developers to maintain high-quality documentation effortlessly, improving code readability and collaboration.

Scenario 2: Code Review
With CodeDocAI, code reviewers can quickly understand the functionality and structure of the code they are reviewing. By inputting the code into the system, reviewers receive a detailed explanation, which helps them identify potential issues and improvements more efficiently. This reduces the time spent on understanding code and enhances the overall quality of the review process.

Scenario 3: Educational Tools
CodeDocAI can be integrated into educational platforms to assist students by providing detailed explanations of code snippets. Students can input code examples, and the LLM generates documentation that explains the logic and functionality. This helps students learn how to write well-documented code and understand complex concepts more intuitively, facilitating a more effective learning experience.

Technical Architecture



Project Flow:

- User interacts with the UI to enter the input.
- User input is collected from the UI and transmitted to the backend using the Google API key.
- The input is then forwarded to the Gemini Pro pre-trained model via an API call.
- The Gemini Pro pre-trained model processes the input and generates the output.
- The results are returned to the frontend for formatting and display.

To accomplish this, we have to complete all the activities listed below:

- **Requirements Specification**
  - Create a requirements.txt file to list the required libraries.
  - Install the required libraries
- **Initialization of Google API Key**
  - Generate Google API Key
  - Initialize Google API Key
- **Interfacing with Pre-trained Model**
  - Load the Gemini Pro pre-trained model
  - Implement a function to get gemini response
- **Model Deployment**
  - Integrate with Web Framework
  - Host the Application

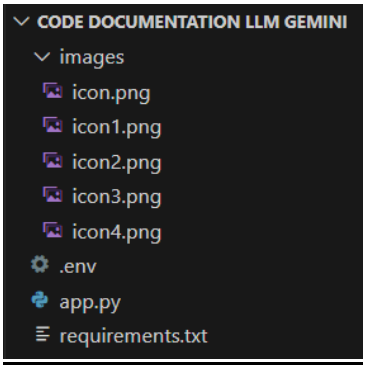
Prior Knowledge:

You must have the prior knowledge of the following topics to complete this project.

- Generative AI Concepts
- NLP: https://www.tutorialspoint.com/natural_language_processing/index.htm
- Generative AI: https://en.wikipedia.org/wiki/Generative_artificial_intelligence
- About Gemini: https://deepmind.google/technologies/gemini/#introduction
- Gemini API: https://ai.google.dev/gemini-api/docs/get-started/python
- Gemini Demo: https://colab.research.google.com/github/google/generative-ai-docs/blob/main/site/en/gemini-api/docs/get-started/python.ipynb
- Streamlit: https://www.geeksforgeeks.org/a-beginners-guide-to-streamlit/

Project Structure

Create the Project folder which contains files as shown below:



- images folder: It is established to store the images utilized in the user interface.
- .env file: It securely stores the Google API key.
- app.py: It serves as the primary application file housing both the model and Streamlit UI code.
- requirements.txt: It enumerates the libraries necessary for installation to ensure proper functioning.
- Additionally, ensure proper file organization and adhere to best practices for version control.

Milestone 1: Requirements Specification

Specifying the required libraries in the requirements.txt file ensures seamless setup and reproducibility of the project environment, making it easier for others to replicate the development environment.

Activity 1: Create a requirements.txt file to list the required libraries.

```
# libraries need to be installed
streamlit
streamlit_extras
google-generativeai
python-dotenv
Pillow
```

- streamlit: Streamlit is a powerful framework for building interactive web applications with Python.
- streamlit_extras: Additional utilities and enhancements for Streamlit applications.
- google-generativeai: Python client library for accessing the GenerativeAI API, facilitating interactions with pre-trained language models like Gemini Pro.
- python-dotenv: Python-dotenv allows you to manage environment variables stored in a .env file for your Python projects.
- Pillow: Pillow is a Python Imaging Library (PIL) fork that adds support for opening, manipulating, and saving many different image file formats.

Activity 2: Install the required libraries

- Open the terminal.
- Run the command: pip install -r requirements.txt
- This command installs all the libraries listed in the requirements.txt file.

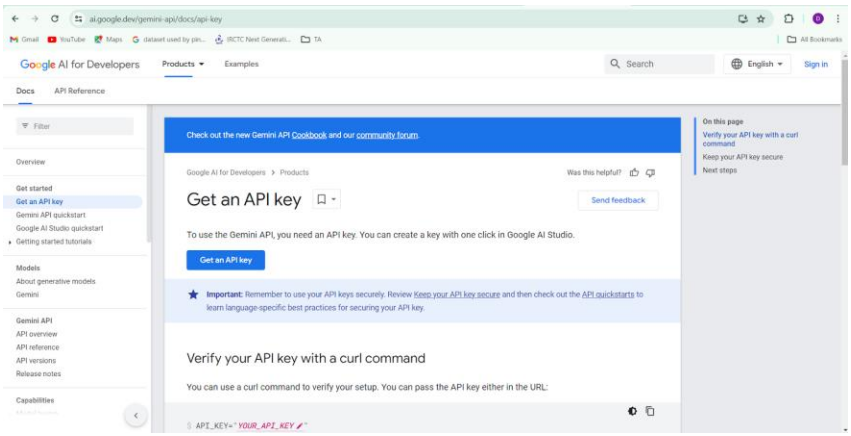
Milestone 2: Initialization of Google API Key

The Google API key is a secure access token provided by Google, enabling developers to authenticate and interact with various Google APIs. It acts as a form of identification, allowing users to access specific Google services and resources. This key plays a crucial role in authorizing and securing API requests, ensuring that only authorized users can access and utilize Google's services.

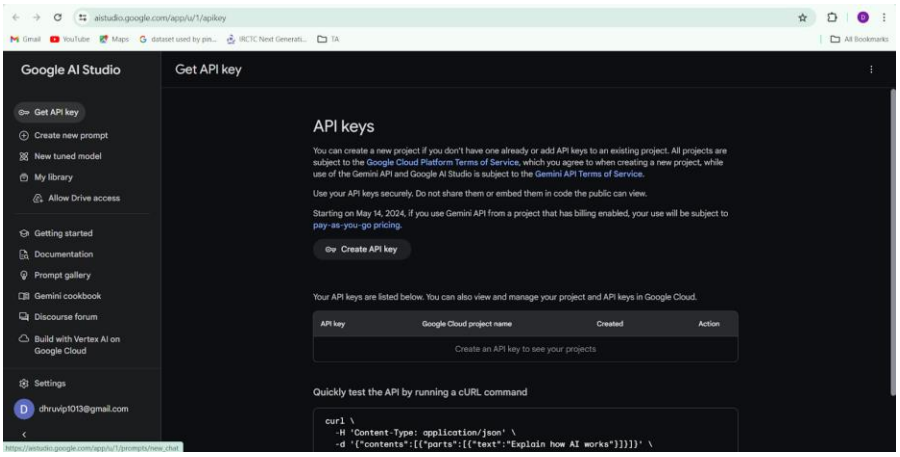
Activity 1: Generate Google API Key

Click the provided link to access the following webpage.

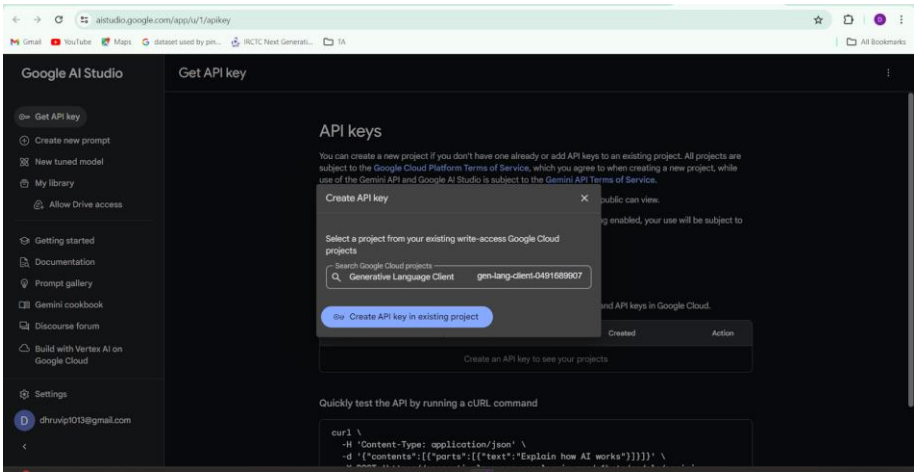
Link: https://ai.google.dev/gemini-api/docs/api-key



After signing in to your account, navigate to the 'Get an API Key' option. Clicking on this option will redirect you to another webpage as shown below.



Next, click on 'Create API Key' and choose the generative language client as the project. Then, select 'Create API key in existing project'.



Copy the newly generated API key as it is required for loading the Gemini Pro pre-trained model.

Activity 2: Initialize Google API Key

```
GOOGLE_API_KEY = "<Enter the copied Google API Key>"
```

- Create a .env file and define a variable named GOOGLE_API_KEY.
- Assign the copied Google API key to this variable.
- Paste the API key obtained from the previous steps here.

Milestone 3: Interfacing with Pre-trained Model

To interface with the pre-trained model, we'll start by creating an app.py file, which will contain both the model and Streamlit UI code.

Activity 1: Load the Gemini Pro pre-trained model

```
from dotenv import load_dotenv
import streamlit as st
from streamlit_extras import add_vertical_space as avs
import google.generativeai as genai
import os
from PIL import Image

load_dotenv()

genai.configure(api_key=os.getenv("GOOGLE_API_KEY"))

model = genai.GenerativeModel('gemini-pro')
```

- The code begins by importing necessary libraries and modules, including dotenv, Streamlit, os, GenerativeAI from Google, PIL (Python Imaging Library), and a custom module for adding vertical space in Streamlit.
- It loads environment variables from the .env file using the load_dotenv() function.
- The GenerativeAI module is configured with the Google API key stored in the environment variable GOOGLE_API_KEY.
- A GenerativeModel object named "model" is created using the Gemini Pro pre-trained model from Google.

- The code is essentially setting up the environment, configuring the GenerativeAI module with the API key, and loading the Gemini Pro model for generating responses to user inputs in the Streamlit app.

Activity 2: Implement a function to get gemini response

```
# Define a function to get the documentation from the LLM
def get_gemini_response(code_snippet):
    prompt = f"""You are an advanced AI language model capable of generating comprehensive and
    accurate documentation for code snippets. Your task is to read a given code snippet and produce
    detailed documentation, explaining the purpose, functionality, parameters, return values, and any
    other relevant information.:\n\n{code_snippet}
    Follow these guidelines:
    Purpose: Describe what the code does.
    Functionality: Explain how the code works, including any important logic or algorithms used.
    Parameters: List and describe each parameter, including its type and purpose.
    Return Values: Detail what the code returns, including its type and purpose.
    Examples: Provide examples of how the code might be used, if applicable.
    Edge Cases: Mention any edge cases or potential errors and how the code handles them.
    """
    response = model.generate_content([prompt])
    return response.text
```

- The function get_gemini_response takes a code snippet as a parameter.
- The prompt variable holds a detailed prompt using an f-string, providing guidelines for the Gemini model to generate documentation for the given code_snippet. The prompt includes sections for Purpose, Functionality, Parameters, Return Values, Examples, and Edge Cases.
- The function calls the generate_content method of the model object to generate a response.
- The generated response is returned as text.

Milestone 4: Model Deployment

We deploy our model using the Streamlit framework, a powerful tool for building and sharing data applications quickly and easily. With Streamlit, we can create interactive web applications that allow users to interact with our models in real-time, providing an intuitive and seamless experience.

Activity 1: Integrate with Web Framework

The webpage is organized into four main sections to provide users with a comprehensive experience:

- Introduction:

```python
# Streamlit app layout
st.set_page_config(page_title="CodeDocAI", layout="wide")

st.title("CodeDocAI")
avs.add_vertical_space(1)

col1, col2 = st.columns([1, 3])
with col1:
    img = Image.open("images/icon.png")
    st.image(img)

with col2:
    st.header("Effortless Code Documentation at Your Fingertips")
    st.markdown("""<p style='text-align: justify;'>
                Introducing CodeDocAI, an advanced project powered by a Large Language Model (LLM) that
                effortlessly generates comprehensive and accurate documentation from code snippets. Designed
                to simplify the documentation process, CodeDocAI provides developers with detailed explanations
                of their code, enhancing understanding and maintainability without the need for manual comments.
                Whether you are a software developer seeking high-quality documentation, a code reviewer aiming
                to quickly understand and improve code, or an educator helping students learn coding concepts,
                CodeDocAI offers robust solutions tailored to your needs. Experience seamless code documentation
                with CodeDocAI and transform the way you interact with your code.
                </p>""", unsafe_allow_html=True)



col1, col2, col3 = st.columns([1, 2, 1])
with col2:
    img1 = Image.open("images/icon1.png")
    st.image(img1, width = 400)

avs.add_vertical_space(8)
```

- This Streamlit app layout is divided into three main sections.
- In the first section, the page configuration is set with the title "CodeDocAI" and a wide layout. Following this, a title "CodeDocAI" is displayed, followed by a vertical space.
- The second section comprises two columns, with the first column containing an image loaded from the file "images/icon.png", and the second column containing a header and markdown text introducing CodeDocAI.
- Finally, the third section also consists of three columns, with the middle column containing an image loaded from "images/icon1.png".
- This layout provides a visually appealing presentation of CodeDocAI's features and benefits.

- Offering:

```
col1, col2 = st.columns([2, 3])

with col1:
    img2 = Image.open("images/icon2.png")
    st.image(img2)

with col2:
    st.header("Wide Range of Offerings")
    st.write("- Seamless documentation generation from code snippets")
    st.write("- Detailed explanations of code logic and functionality")
    st.write("- Enhanced understanding and maintainability of code")
    st.write("- No need for extensive manual comments")
    st.write("- Suitable for software development, code review, and educational purposes")
    st.write("- High-quality documentation effortlessly obtained")
    st.write("- Improved code readability and collaboration")
    st.write("- Time-saving for code reviewers")
    st.write("- Enhanced learning experience for students")
    st.write("- Intuitive and effective documentation process")
```

- This code segment creates a two-column layout using Streamlit's `columns` function, with the first column displaying an image loaded from "images/icon2.png" and the second column containing a header followed by a list of offerings provided by CodeDocAI.

- Code Documentation Application:

```
col1, col2 = st.columns([2, 1])
with col1:
    st.header("Present Your Code Slice")
    st.subheader("- Explore code snippets effortlessly")
    st.subheader("- Simplify complex code comprehension")
    st.subheader("- Enhance code documentation in just a few clicks")

with col2:
    img3 = Image.open("images/icon3.png")
    st.image(img3, width = 400)

code_snippet = st.text_area("Paste your code snippet here", height=100)

if st.button("Generate Documentation"):
    if code_snippet:
        with st.spinner("Generating documentation..."):
            documentation = get_gemini_response(code_snippet)
            st.subheader("Generated Documentation")
            st.write(documentation)
    else:
        st.error("Please enter a code snippet to generate documentation.")
```

- This code block sets up a two-column layout using Streamlit's `columns` function.
- In the left column, it displays a header titled "Present Your Code Slice" followed by three subheaders listing the benefits of using CodeDocAI.
- Meanwhile, the right column contains an image loaded from "images/icon3.png".
- Below the columns, there's a text area where users can input their code snippets, along with a button labeled "Generate Documentation".
- Upon clicking the button, the system generates documentation for the provided code snippet using the Gemini model and displays it below.
- This layout effectively presents the features of CodeDocAI while providing a user-friendly interface for generating code documentation.

- FAQ:

```
col1, col2 = st.columns([2, 3])

with col1:
    avs.add_vertical_space(5)
    img4 = Image.open("images/icon4.png")
    st.image(img4, use_column_width=True)

with col2:
    st.write("Q: What is CodeDocAI?")
    st.write("""A: CodeDocAI is an advanced project powered by a Large Language Model (LLM)
             designed to generate comprehensive documentation from code snippets automatically.""")
    avs.add_vertical_space(3)
    st.write("Q: How does CodeDocAI simplify the documentation process?")
    st.write("""A: CodeDocAI allows developers to obtain detailed explanations of their code
             without needing to write extensive comments manually, thereby enhancing understanding
             and maintainability.""")
    avs.add_vertical_space(3)
    st.write("Q: In what scenarios can CodeDocAI be used?")
    st.write("""A: CodeDocAI can be utilized across various scenarios, including software development,
             code review, and educational purposes.""")
```

- This code snippet creates a two-column layout using Streamlit's `columns` function.
- In the first column, it adds vertical space and displays an image loaded from "images/icon4.png".
- The second column contains three sections, each presenting a frequently asked question along with its corresponding answer about CodeDocAI.
- Overall, this layout offers an organized and informative FAQ section for users to quickly understand the key aspects of CodeDocAI.

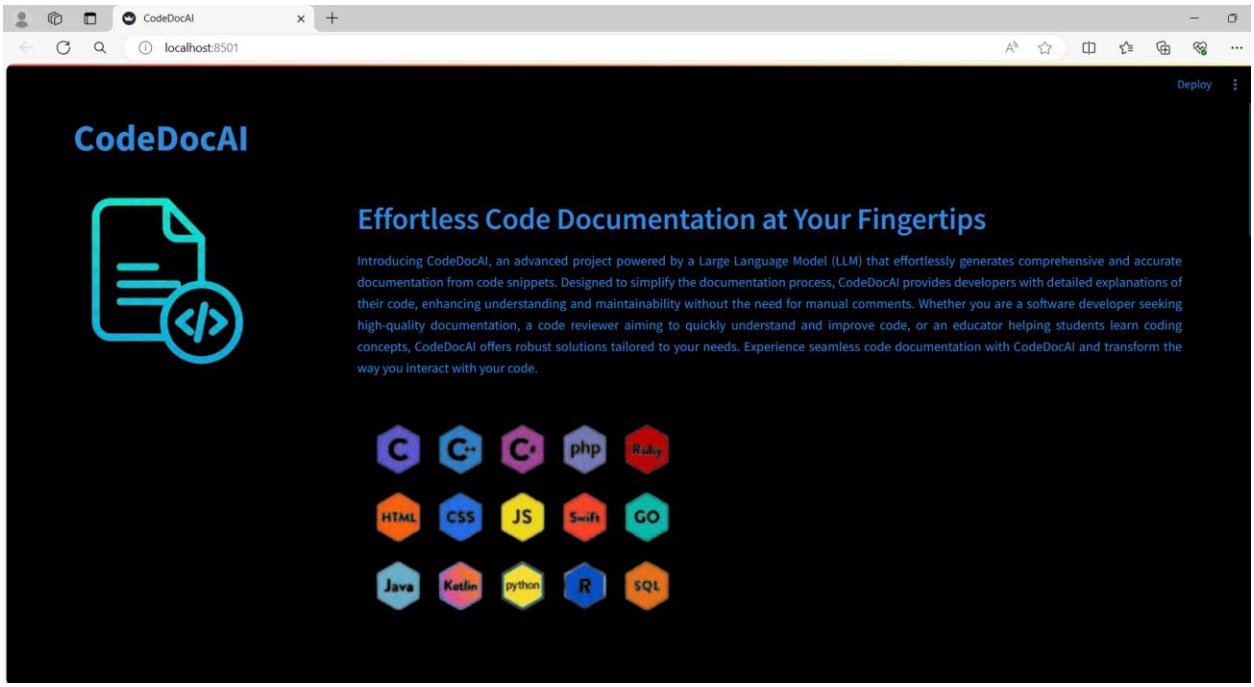
Activity 2: Host the Application

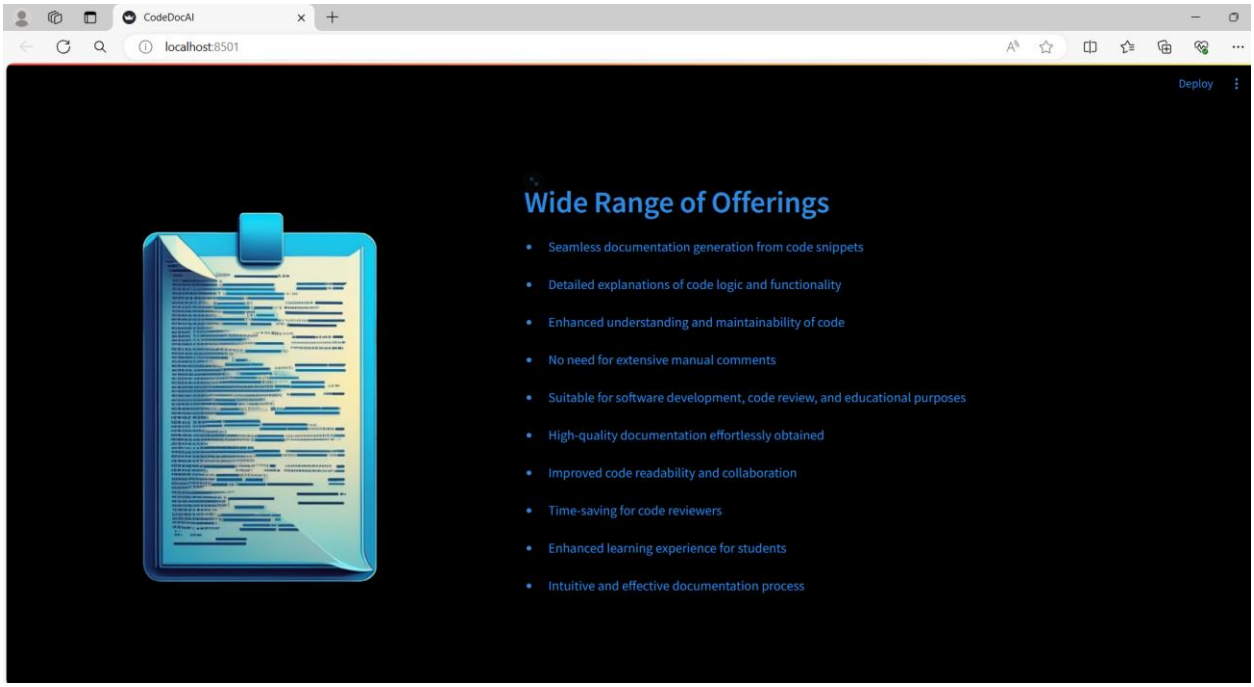
Launching the Application:

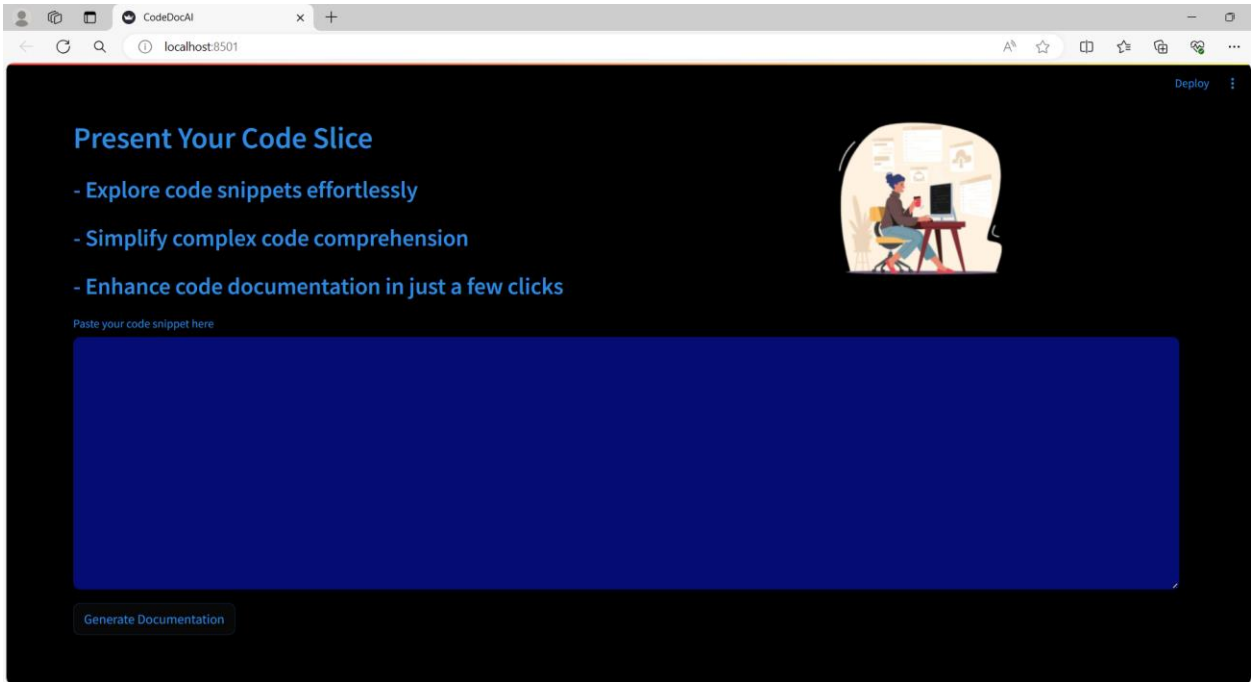
- To host the application, go to the terminal, type - streamlit run app.py
- Here app.py refers to a python script.

```
PS C:\Users\dhurvi patel\Desktop\SCIT academic\Smart_Bridge_Internship\SB_Company work\May_Month\Code Documentation LLM Gemini> streamlit run app.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.29.80:8501
```

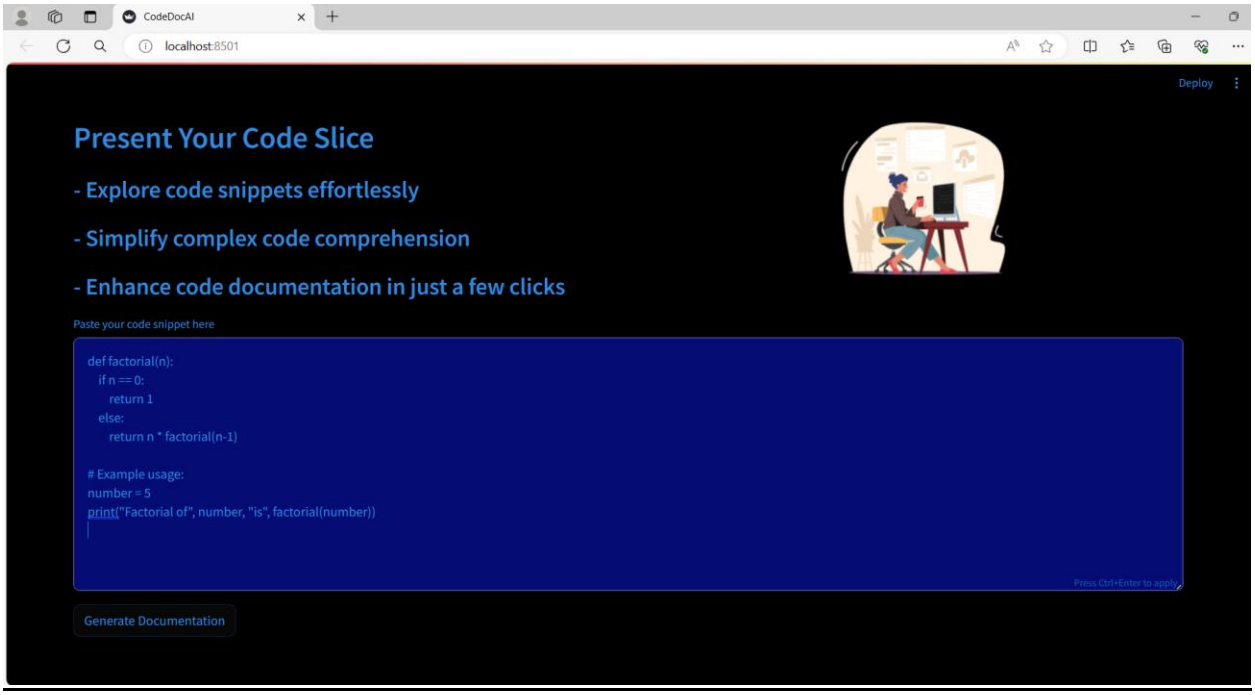
Run the command to get the below results



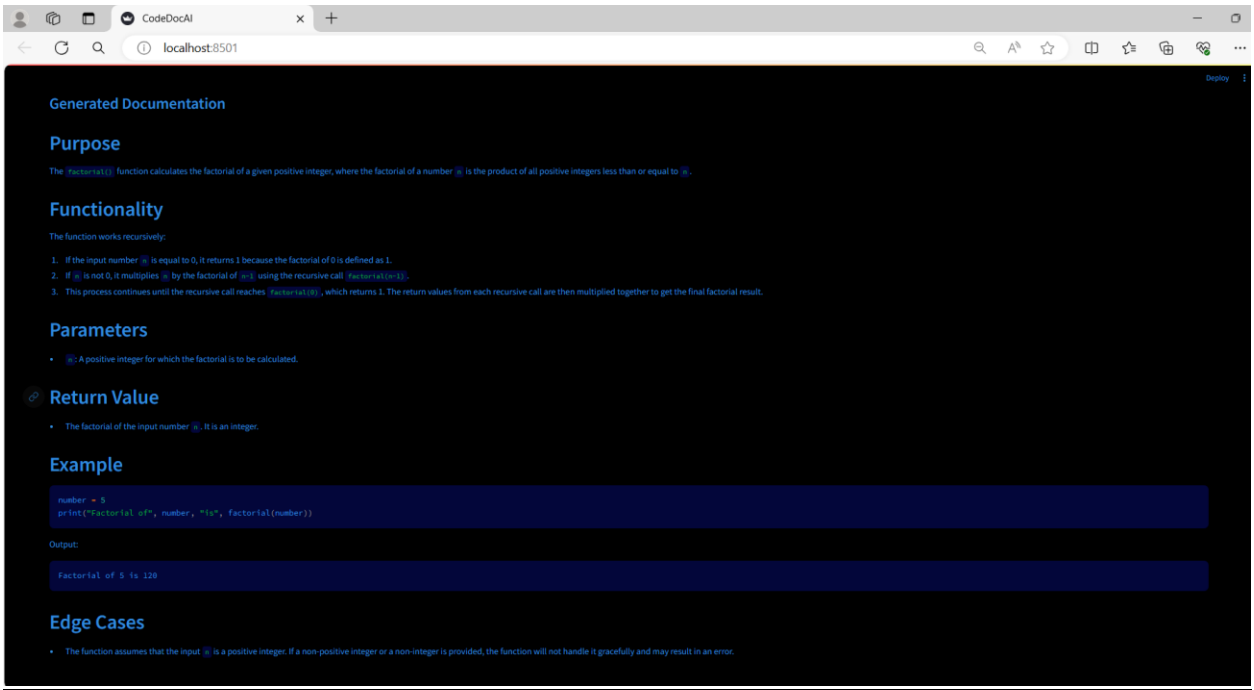

CodeDocAI
localhost:8501
Deploy
Wide Range of Offerings
Seamless documentation generation from code snippets
Detailed explanations of code logic and functionality
Enhanced understanding and maintainability of code
No need for extensive manual comments
Suitable for software development, code review, and educational purposes
High-quality documentation effortlessly obtained
Improved code readability and collaboration
Time-saving for code reviewers
Enhanced learning experience for students
Intuitive and effective documentation process


CodeDocAI
localhost:8501
Deploy
Present Your Code Slice
- Explore code snippets effortlessly
- Simplify complex code comprehension
- Enhance code documentation in just a few clicks
Paste your code snippet here
Generate Documentation

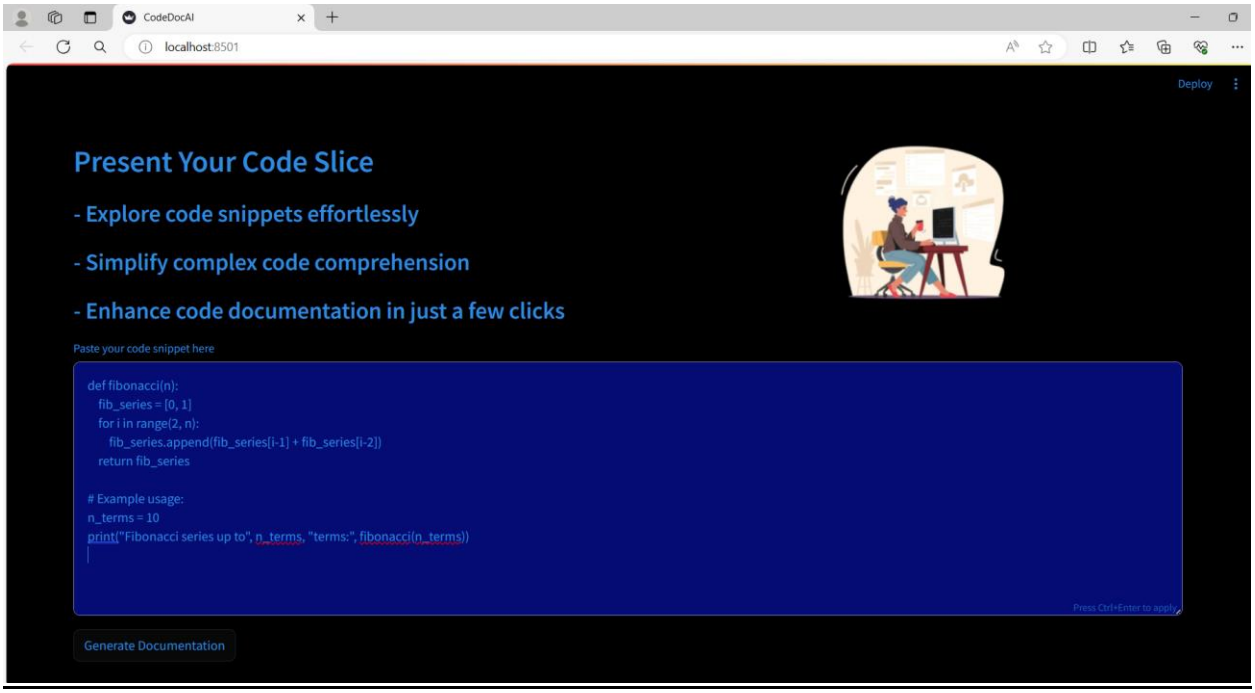
INPUT 1



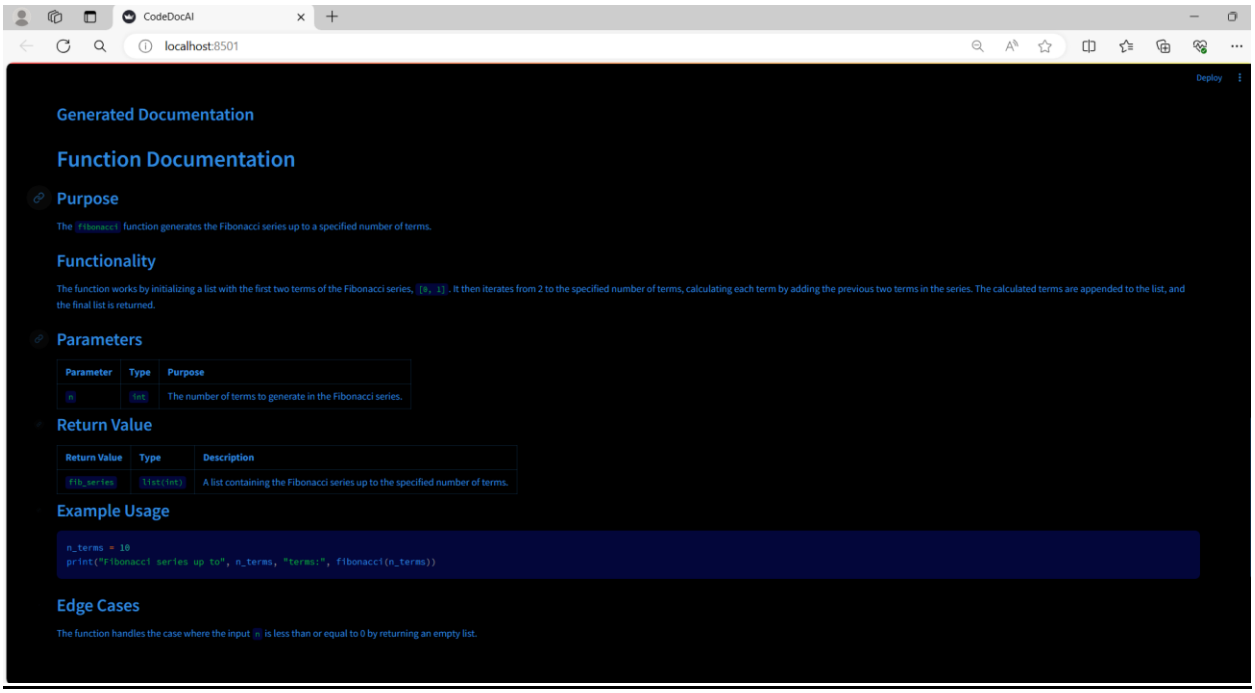
OUTPUT 1



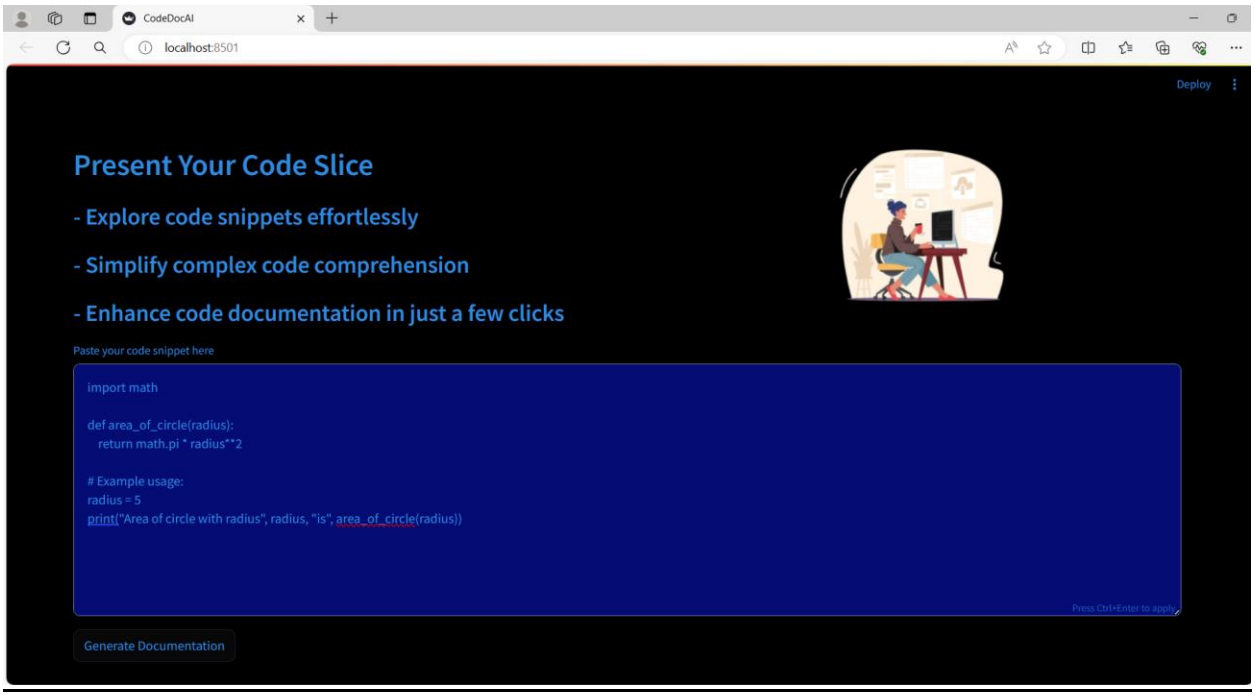
INPUT 2



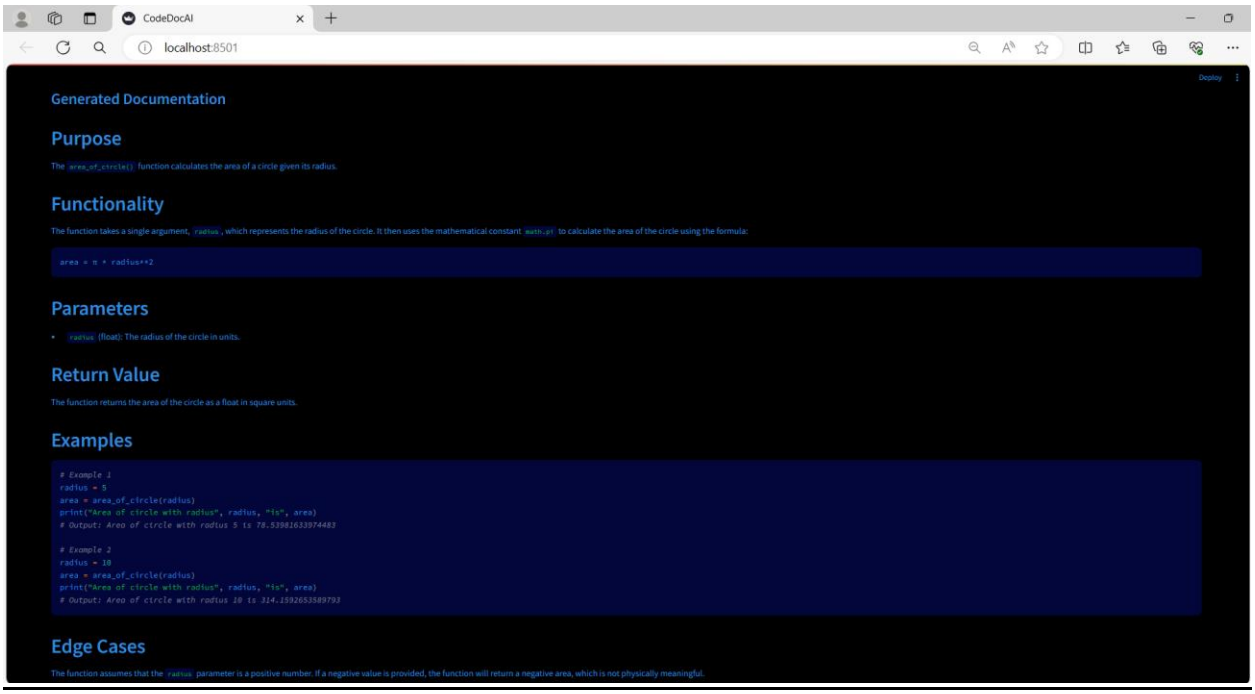
OUTPUT 2

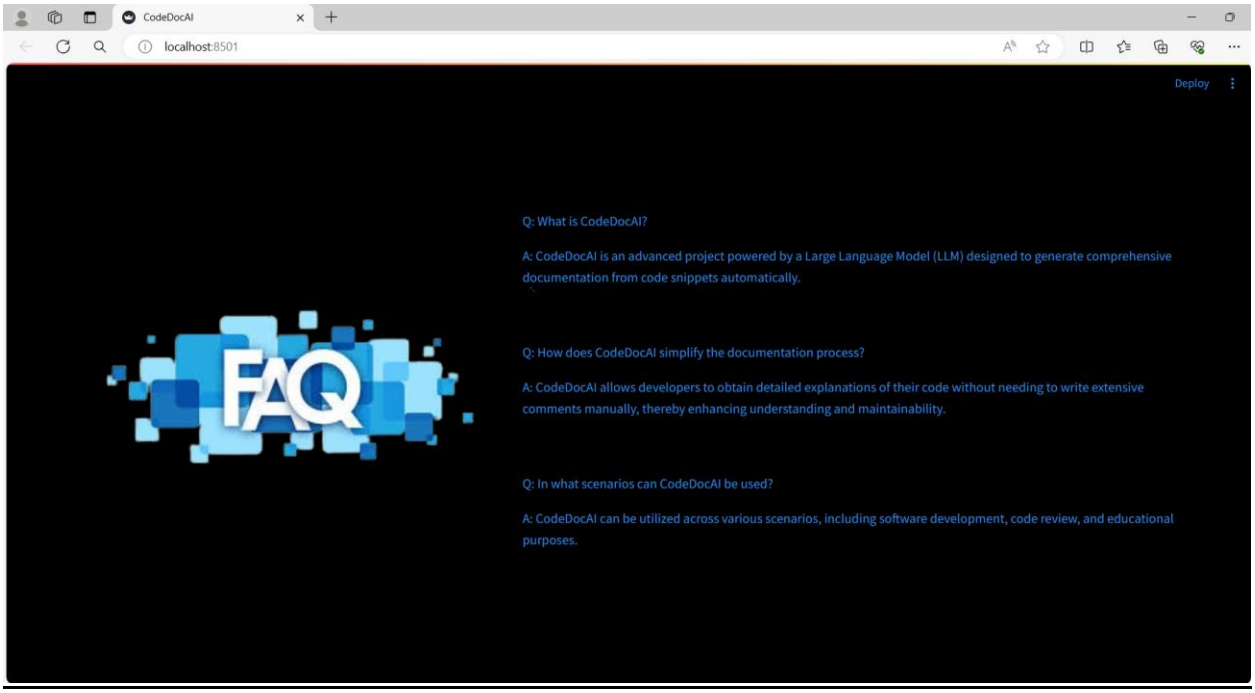


INPUT 3


CodeDocAI
localhost:8501
Deploy
Present Your Code Slice
- Explore code snippets effortlessly
- Simplify complex code comprehension
- Enhance code documentation in just a few clicks
Paste your code snippet here
import math
def area_of_circle(radius):
return math.pi * radius**2
# Example usage:
radius = 5
print("Area of circle with radius", radius, "is", area_of_circle(radius))
Press Ctrl+Enter to apply
Generate Documentation

OUTPUT 3


CodeDocAI
localhost:8501
Deploy
Generated Documentation
Purpose
The area_of_circle() function calculates the area of a circle given its radius.
Functionality
The function takes a single argument, radius, which represents the radius of the circle. It then uses the mathematical constant math.pi to calculate the area of the circle using the formula:
area = π * radius**2
Parameters
radius (float): The radius of the circle in units.
Return Value
The function returns the area of the circle as a float in square units.
Examples
# Example 1
radius = 5
area = area_of_circle(radius)
print("Area of circle with radius", radius, "is", area)
# Output: Area of circle with radius 5 is 78.53981633974483
# Example 2
radius = 10
area = area_of_circle(radius)
print("Area of circle with radius", radius, "is", area)
# Output: Area of circle with radius 10 is 314.1592653589793
Edge Cases
The function assumes that the radius parameter is a positive number. If a negative value is provided, the function will return a negative area, which is not physically meaningful.


CodeDocAI
localhost:8501
Deploy
FAQ
Q: What is CodeDocAI?
A: CodeDocAI is an advanced project powered by a Large Language Model (LLM) designed to generate comprehensive documentation from code snippets automatically.
Q: How does CodeDocAI simplify the documentation process?
A: CodeDocAI allows developers to obtain detailed explanations of their code without needing to write extensive comments manually, thereby enhancing understanding and maintainability.
Q: In what scenarios can CodeDocAI be used?
A: CodeDocAI can be utilized across various scenarios, including software development, code review, and educational purposes.