ajitdash Microsoft

Apr 30 2023 0

...

Integrating OpenAI with Streamlit: With Example Source Code Explainer

Integrating OpenAI with Streamlit: With Example Source Code Explainer

Abstract:

This blog explores the integration of OpenAI with Streamlit, demonstrating an example of how the interaction works. The blog will cover the topic of the source code explainer, which converts source code to simple text (explains in simple text) using OpenAI's GPT-3.5 . The blog will also discuss the importance of integrating artificial intelligence technologies like OpenAI with user interfaces like Streamlit to provide users with seamless and powerful tools to work with. Integrating OpenAI with Streamlit is a powerful combination that allows developers to build sophisticated and intelligent applications.

In this blog, we explore how this integration works and provide an example of a source code explainer that uses OpenAI's GPT-3.5 language model. Also discuss the installation process, challenges, and best practices for integrating OpenAI with Streamlit.

By the end of this blog, you'll have a solid understanding of how to integrate OpenAI with Streamlit and create powerful language processing applications. Whether you're a data scientist, machine learning engineer, or just interested in the intersection of AI and development, this blog is for you.

Introduction:

OpenAI is a leading artificial intelligence research laboratory that develops cutting-edge language models. Streamlit, on the other hand, is a popular open-source Python library that allows developers to build interactive and data-driven applications quickly. Combining the two provides a powerful platform for building intelligent applications that can understand and respond to user inputs.

Integrating OpenAI with Streamlit :

OpenAI's GPT-3.5 is one of the most powerful language models in the world, capable of natural language processing, translation, and even code generation. Streamlit, on the other hand, is a popular open-source framework for creating interactive data science and machine learning applications. Before, we explore how to integrate OpenAI with Streamlit to create a powerful source code explainer that can convert code into simple text and explain it in steps .

Lets First, we'll walk through the installation process for OpenAI and Streamlit, including any potential challenges and best practices for integration. We'll then dive into how to use OpenAI's API to translate source code and display the results in a Streamlit application in text format with step by step explanations .

Along the way, we'll cover important topics such as token and temperature settings for OpenAI, language selection, and file handling in Streamlit. We'll also provide a detailed example of how to convert Python code to plain English, as well as explore other potential use cases for OpenAI and Streamlit integration.

Installation:

To integrate OpenAI with Streamlit, you first need to install the required libraries. You can use the pip package manager to install the following libraries:
- OpenAI
To install OpenAI, you first need to sign up for an OpenAI account and obtain an API key. You can then install the OpenAI module using the following command:
 pip install openai
- Streamlit
You can install Streamlit using pip with the following command:
pip install streamlit
- Dotenv
To install the dotenv package, you can use pip, the package installer for Python. Open your terminal or command prompt and type the following command:
pip install python-dotenv

Challenges:

One of the main challenges of integrating OpenAI with Streamlit is ensuring that your application remains secure. OpenAI requires an API key to access its language models, and you need to ensure that this key is kept secure. You should also limit access to the key to only authorized personnel.

Key without security : openai.api_key = '363e5eaaaaaabbbbbccccc'

Few ways we can keep your access key secrete.

Option # 1 - Using Streamlit secrets

This is the easiest way using Streamlit secrets :

1.Create a folder within your director where you have the code name as “**.streamlit** “

2.Create a file name as “ **secrets.toml” under the folder** “**.streamlit** “

3.Assign the key in the “ **secrets.toml”**

Path = '363e5eaaaaaabbbbbccccc'

Flow will look like this : projectfolder\streamlit\.streamlit

When you call the key within your code use this : openai.api_key = st.secrets['path']

Make sure to add the other credentials. Will look like this

```
# Set up OpenAI API credentials
openai.api_type = "azure"
openai.api_base = 'https://xxxxxx.openai.azure.com/'
openai.api_version = "2023-03-15-preview"
openai.api_key = st.secrets['path']
```

Option #2 Using Environment Variables :

1. Install the python-dotenv package: pip install python-dotenv
2. Create a .env file in the root directory of your project and add your OpenAI API key like this: OPENAI_API_KEY=<your_api_key_here>
3. In your Streamlit app, import the dotenv module and load the API key from the .env file:

```
import os
from dotenv import load_dotenv
load_dotenv()  # Load environment variables from .env file

OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")  # Get API key from environment variable
```

4. Use the OPENAI_API_KEY variable in your OpenAI API calls:

```
import openai
```

```
    openai.api_key = OPENAI_API_KEY
```

Make sure to add the other credentials. Will look like this

```
# Set up OpenAI API credentials
openai.api_type = "azure"
openai.api_base = 'https://xxxxxx.openai.azure.com/'
openai.api_version = "2023-03-15-preview"
openai.api_key = OPENAI_API_KEY
```

Note: Make sure to add this to the **.gitignore**

More About : . gitignore

.gitignore is a file that tells Git which files and directories to ignore when tracking changes in your repository. When you add files to your repository, Git will track changes to those files by default. However, there may be files or directories that you don't want to track, such as log files, temporary files, or sensitive information like API keys.

To ignore these files, you can create a .gitignore file in the root directory of your repository. This file should contain a list of file and directory names that Git should ignore. For example, if you want to ignore all files with the .log extension, you can add the following line to your .gitignore file: *.log

This will tell Git to ignore all files with the .log extension. You can also ignore entire directories by adding the directory name to your .gitignore file. For example, if you want to ignore a directory called temp, you can add the following line: temp/

Note that you can also use wildcards and patterns in your .gitignore file to ignore multiple files or directories at once. For example, you can use the * wildcard to ignore all files with a certain extension: *.txt , .env , . Or you can use the ** pattern to ignore all files and directories within a certain directory: logs/**

It's important to use a .gitignore file to avoid committing sensitive information like API keys to your repository. This can help prevent security breaches and other issues.

Best Practices:

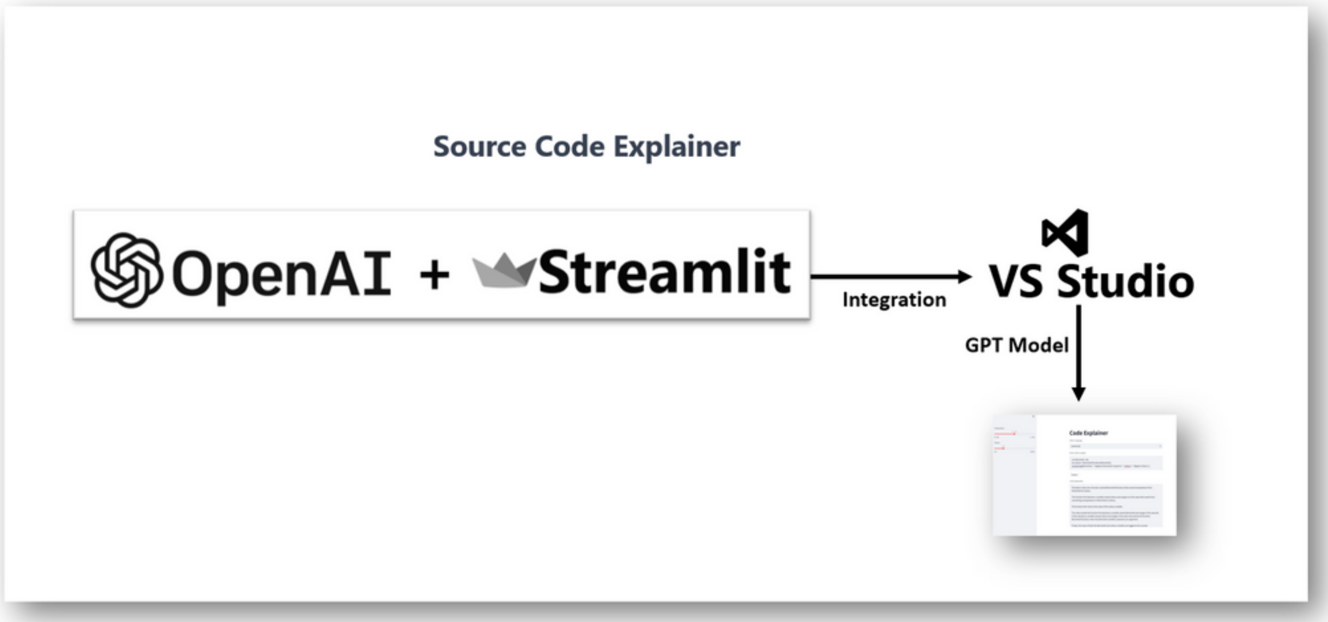
To ensure successful integration, it's important to follow these best practices:

- Keep your API key secure and limit access to authorized personnel
- Use a virtual environment to manage dependencies
- Optimize your code for performance to reduce latency
- Test your application thoroughly before deploying it to production
-Add the key files to the .gitignore file

Example: Source Code Explainer

As an example, we can build a source code explainer that converts code to Text. We can use OpenAI's GPT-3.5 language model to perform the translation. The user can add the designer code, and the explainer will convert the code and display the output with explanation.

Source Code Explainer: Using Streamlit + OpenAI (Code available in the Git)



Code used : Python-Stremlit + OpenAI API

Editor : Visual Studio Code (Also you can use Notebook , Colab)

In OpenAI, the temperature and token settings are used to fine-tune the output of the GPT models.

Temperature: Adjust the temperature between 0 to 1,

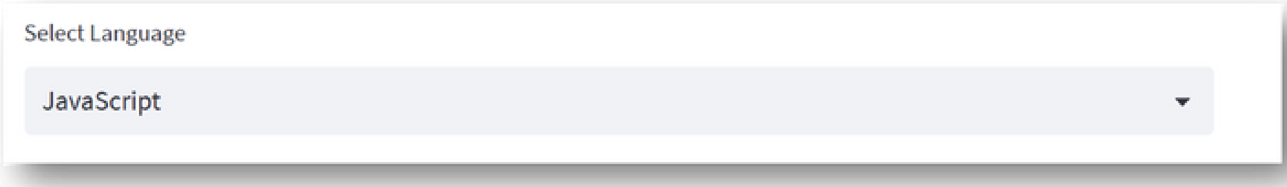
Text is generated randomly based on the temperature setting. Higher temperatures will produce more creative and diverse responses, but can also produce illogical results. Conversely, a lower temperature produces more conservative responses.
Recommend to keep low temperature then as per the need increase it

Token: Adjust between 64 to 2048
A token setting determines how many tokens (words or phrases) can be generated by the model. Having a higher token count can lead to more detailed and nuanced responses but can also cause slower performance.

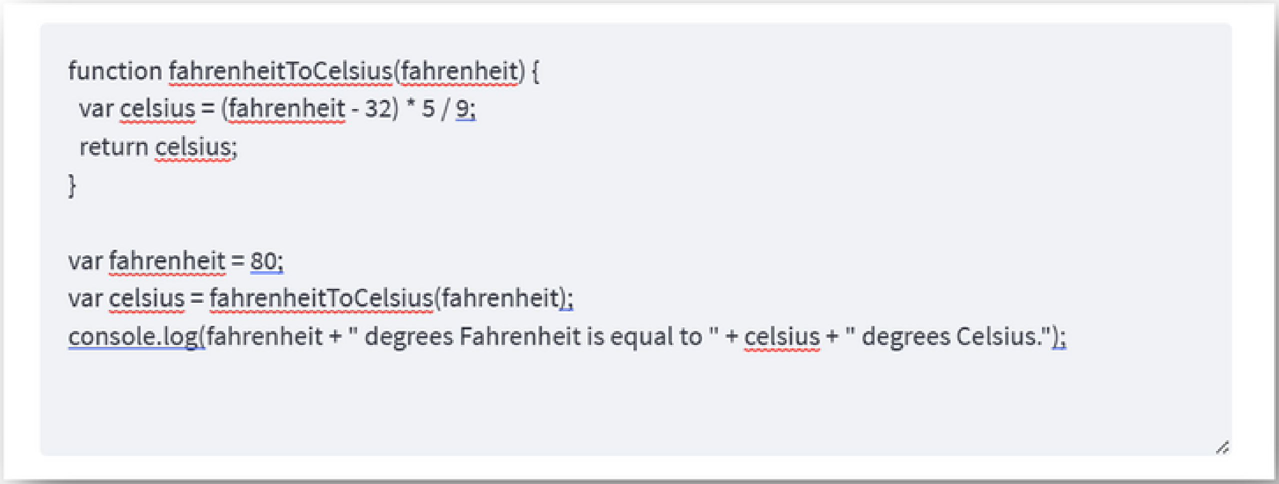
Recommend to keep low Temperature as well as less Token value then as per the need increase it

Interface: Shows a Jave Script Language "Farehentie to Centigrade Conversion " using GPT-3.5-turbo language model .

1. **Select Language drop down you can select any Language Like : Python, Spark , Go etc..**



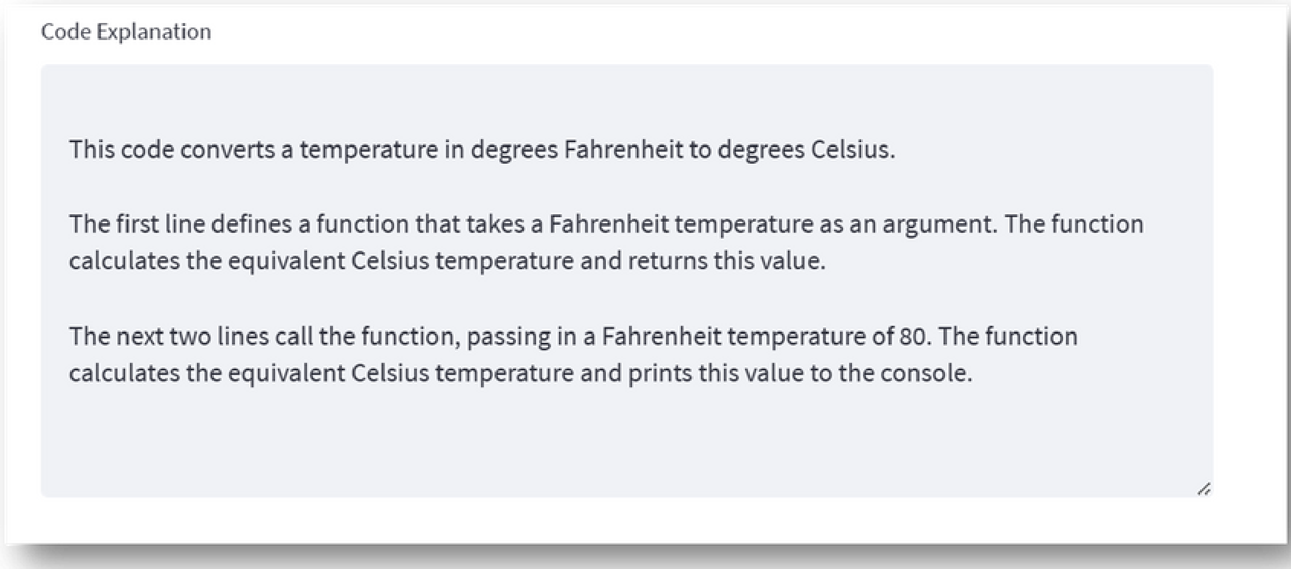
2. Input



```
function fahrenheitToCelsius(fahrenheit) {
  var celsius = (fahrenheit - 32) * 5 / 9;
  return celsius;
}

var fahrenheit = 80;
var celsius = fahrenheitToCelsius(fahrenheit);
console.log(fahrenheit + " degrees Fahrenheit is equal to " + celsius + " degrees Celsius.");
```

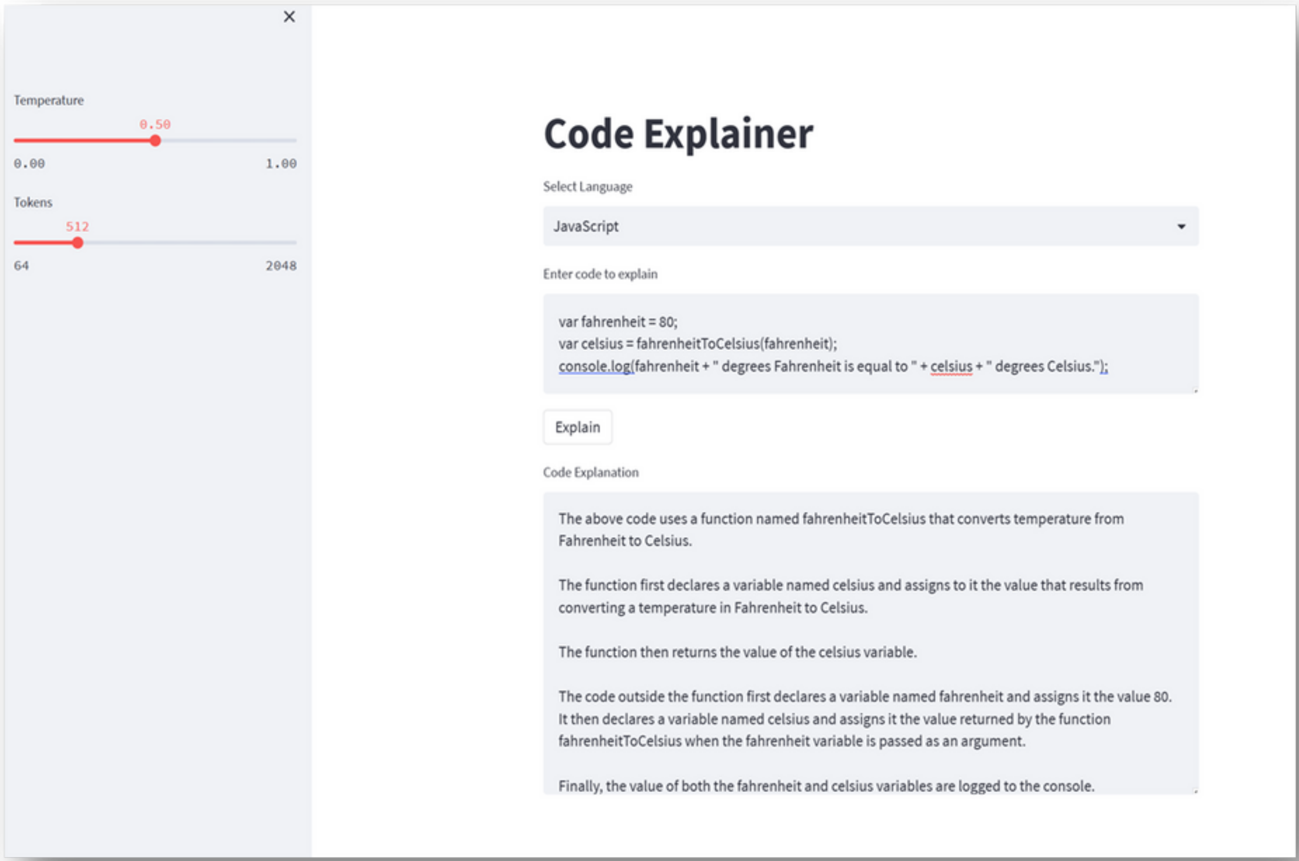
3.Output :


Code Explanation

This code converts a temperature in degrees Fahrenheit to degrees Celsius.

The first line defines a function that takes a Fahrenheit temperature as an argument. The function calculates the equivalent Celsius temperature and returns this value.

The next two lines call the function, passing in a Fahrenheit temperature of 80. The function calculates the equivalent Celsius temperature and prints this value to the console.



Code Explainer

Select Language: JavaScript

Enter code to explain

```
var fahrenheit = 80;
var celsius = fahrenheitToCelsius(fahrenheit);
console.log(fahrenheit + " degrees Fahrenheit is equal to " + celsius + " degrees Celsius.");
```

Code Explanation

The above code uses a function named fahrenheitToCelsius that converts temperature from Fahrenheit to Celsius.

The function first declares a variable named celsius and assigns to it the value that results from converting a temperature in Fahrenheit to Celsius.

The function then returns the value of the celsius variable.

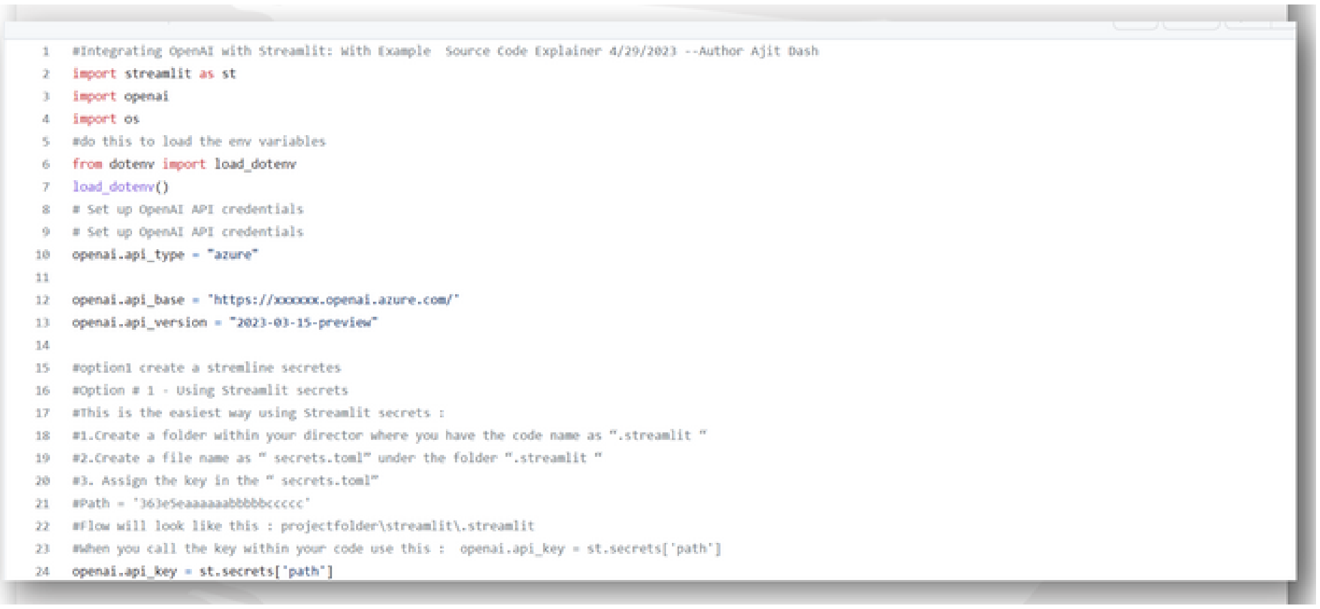
The code outside the function first declares a variable named fahrenheit and assigns it the value 80. It then declares a variable named celsius and assigns it the value returned by the function fahrenheitToCelsius when the fahrenheit variable is passed as an argument.

Finally, the value of both the fahrenheit and celsius variables are logged to the console.

Code Available In Git:

https://github.com/ajitdash/pview/blob/main/explaincode.py

https://github.com/ajitdash/pview/blob/main/tempcontrol-javasc.txt



```
#Integrating OpenAI with Streamlit: With Example  Source Code Explainer 4/29/2023 --Author Ajit Dash
import streamlit as st
import openai
import os
#do this to load the env variables
from dotenv import load_dotenv
load_dotenv()
# Set up OpenAI API credentials
# Set up OpenAI API credentials
openai.api_type = "azure"

openai.api_base = "https://xxxxxx.openai.azure.com/"
openai.api_version = "2023-03-15-preview"

#option1 create a stremline secretes
#Option # 1 - Using Streamlit secrets
#This is the easiest way using Streamlit secrets :
#1.Create a folder within your director where you have the code name as ".streamlit "
#2.Create a file name as " secrets.toml" under the folder ".streamlit "
#3. Assign the key in the " secrets.toml"
#Path = '363e5eaaaaaabbbbbccccc'
#Flow will look like this : projectfolder\streamlit\.streamlit
#When you call the key within your code use this :  openai.api_key = st.secrets['path']
openai.api_key = st.secrets['path']
```

Conclusion:

Integrating OpenAI with Streamlit provides a powerful platform for building intelligent and data-driven applications. While there are some challenges to this integration, following best practices and testing your code thoroughly can help ensure a successful implementation. The source code explainer example shows how this integration can be used to build sophisticated applications that can understand and respond to user inputs.

0 Likes

Share