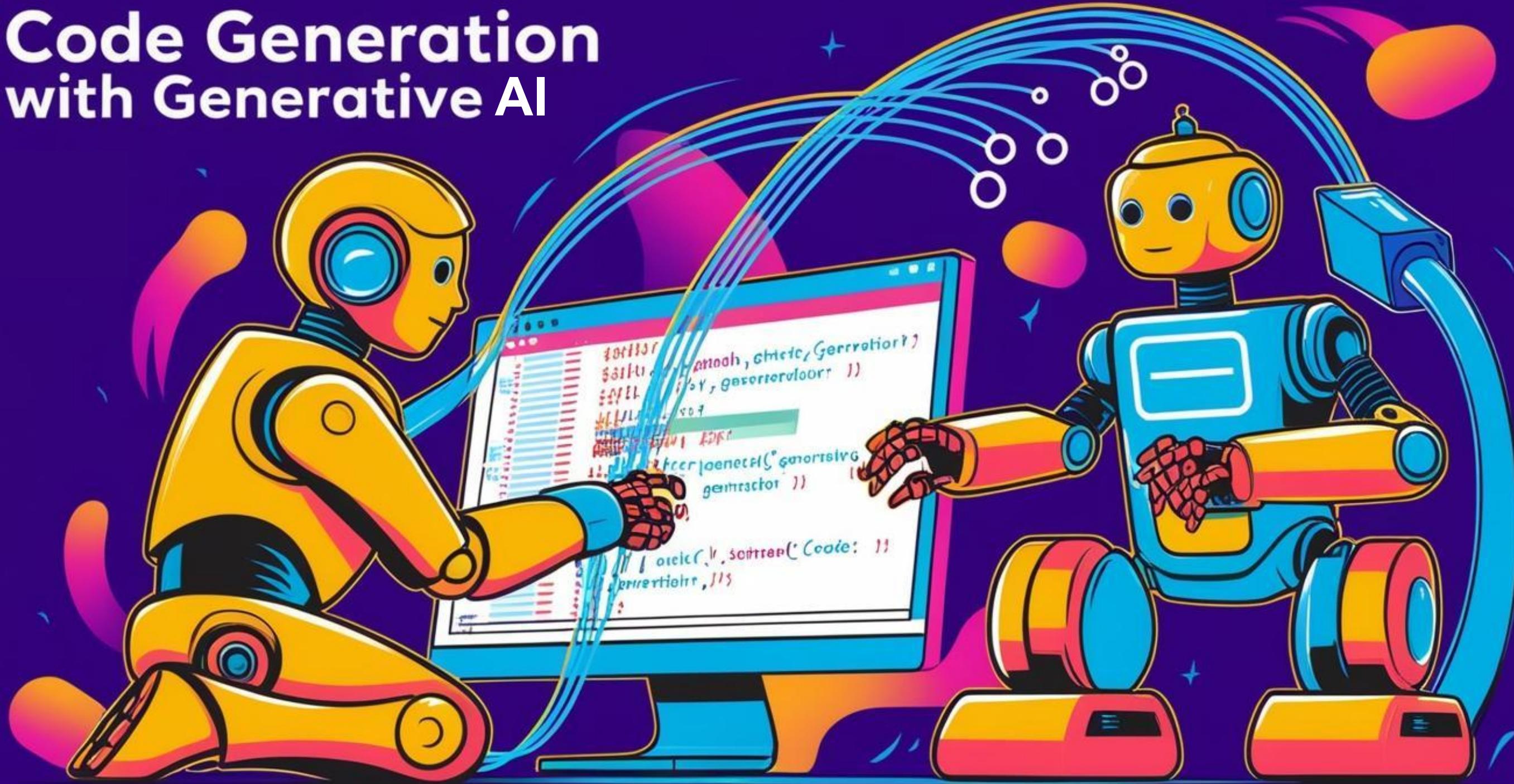


# Code Generation with Generative AI





# Outline



01. How Software Development Evolved
02. Github Survey and Experiment on Developers
03. Benefits of Code Generation with Gen AI
04. LLMs for Code Generation and Datasets used for their Training
05. AI tools for Code Generation



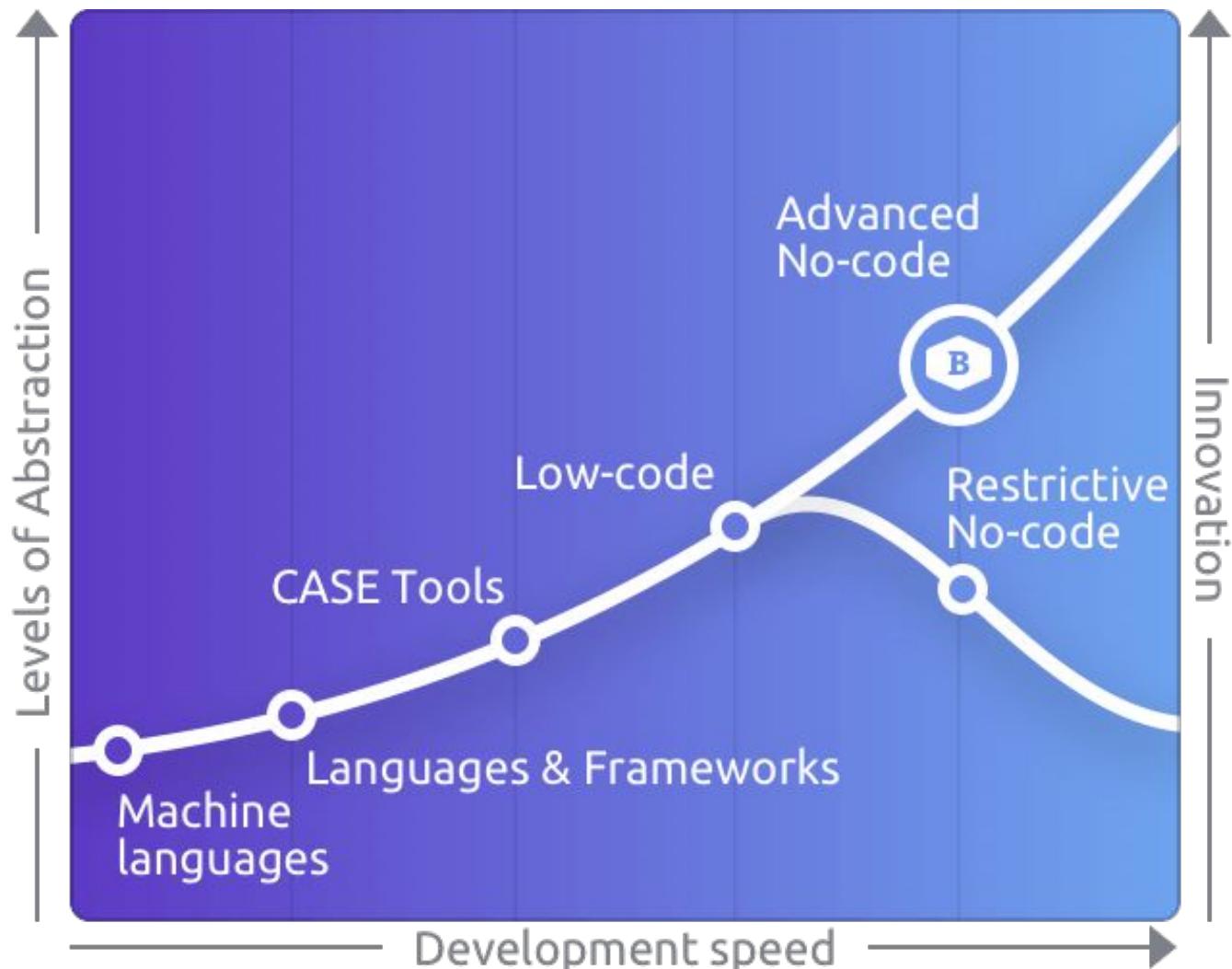
# Outline



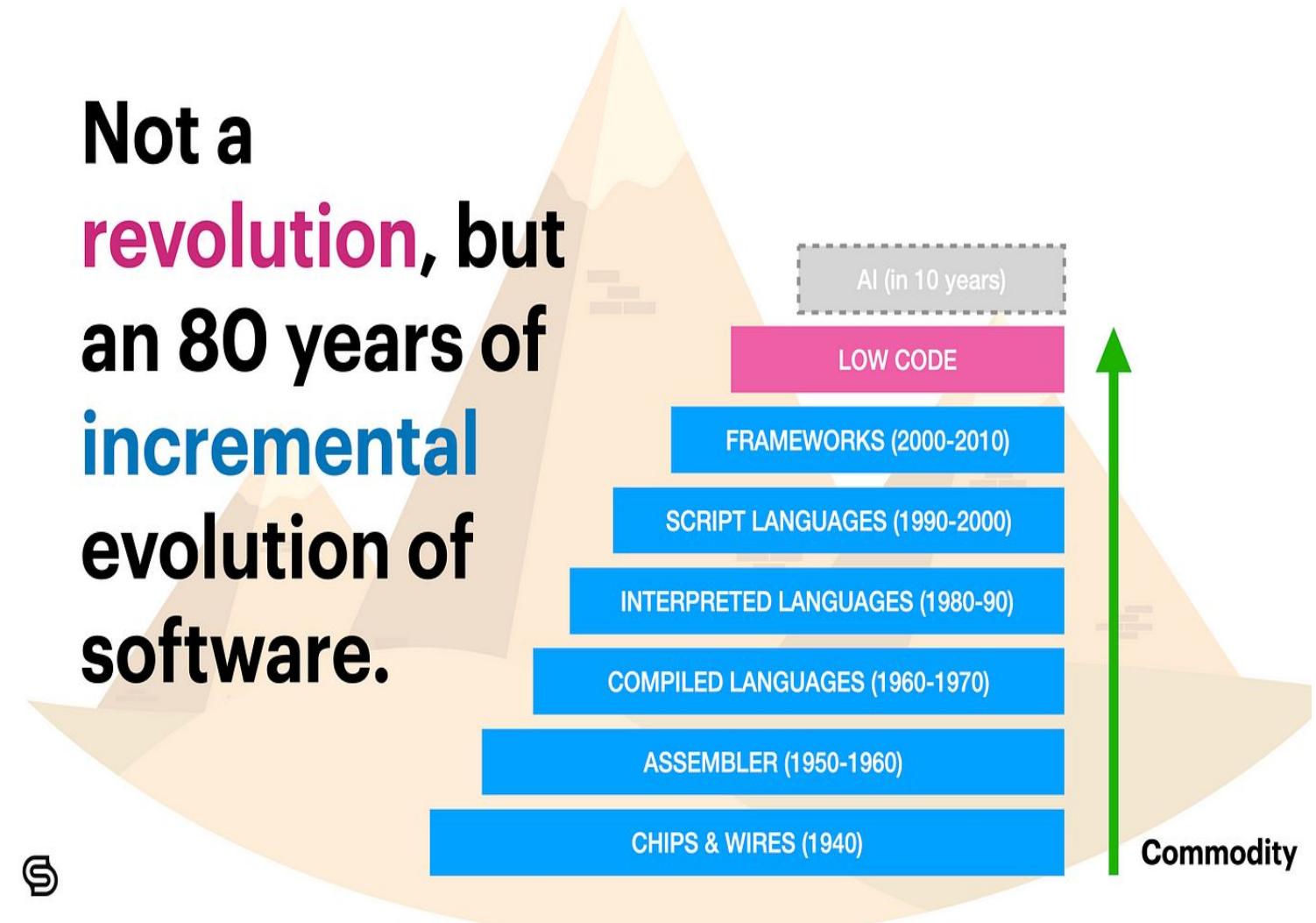
06. Challenges of Code Generation with AI
07. Prompt Engineering Tips for Code Generation
08. Github Copilot Setup and Usage
09. Use Case Demo
10. Conclusion

# How Software Development Evolved

- Coding Evolution: From manual binary coding to high-level programming languages, complexity has reduced significantly over time.
- Efficiency Growth: Each phase introduced tools that made writing, debugging, and scaling code faster and easier.
- AI's Role: Today, AI tools simplify coding further, automating tasks and enhancing productivity.

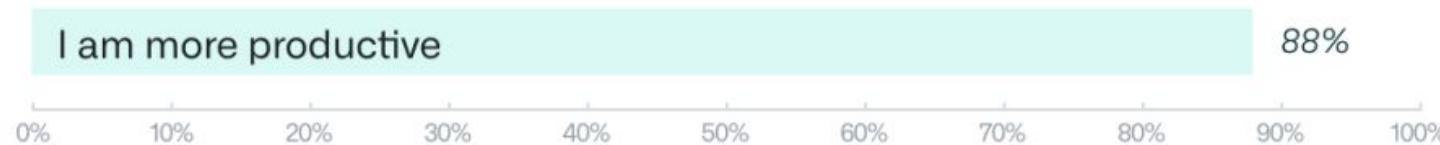


**Not a revolution, but an 80 years of incremental evolution of software.**



# When using GitHub Copilot...

## Perceived Productivity



## Satisfaction and Well-being\*



## Efficiency and Flow\*



Github surveyed more than 2,000 developers to learn at scale about their experience using GitHub Copilot and found out

Github Recruited

95

developers, and split them randomly into two groups.

gave them the task of writing a web server in JavaScript

45 Used

GitHub Copilot

78%  
finished

1 hour, 11 minutes

average to complete the task



50 Did not use

GitHub Copilot

70%  
finished

2 hours, 41 minutes

average to complete the task

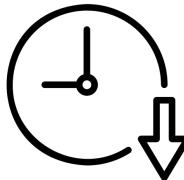


Results are statistically significant ( $P=.0017$ ) and the 95% confidence interval is [21%, 89%]

In Another Experiment Done by Github they observed

- Developers using Copilot completed tasks 55% faster and had higher completion rates than those without it.

# Benefits of AI Driven Code Generation



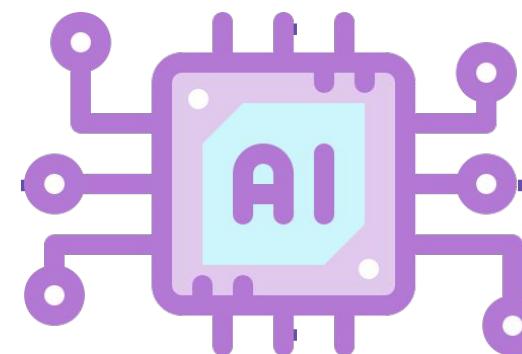
Reduced  
Development  
Time



Simplified  
Debugging



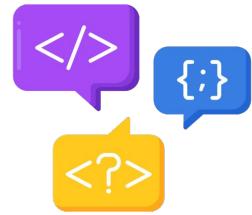
Improved  
Code  
Quality



Learning  
Assistance



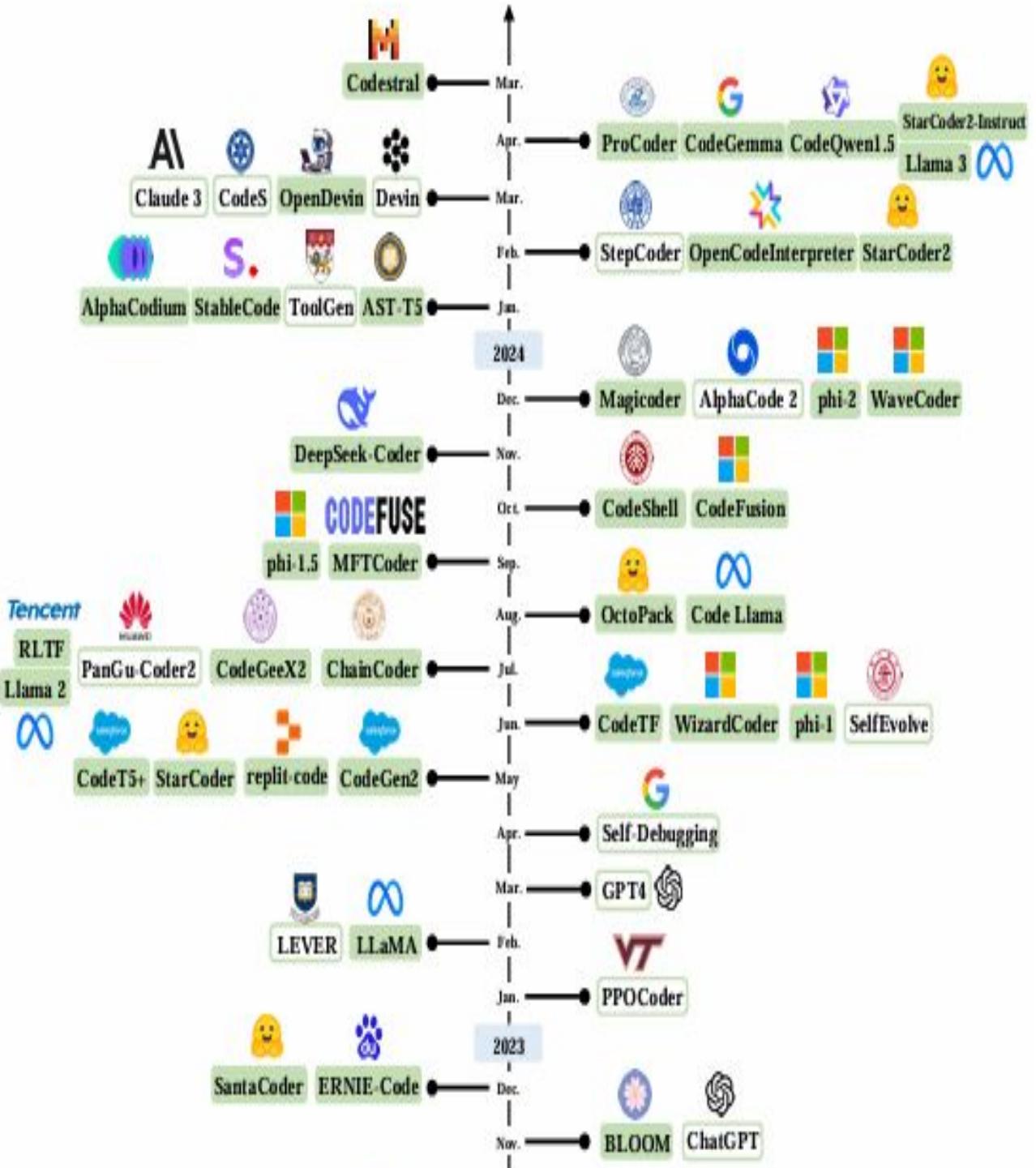
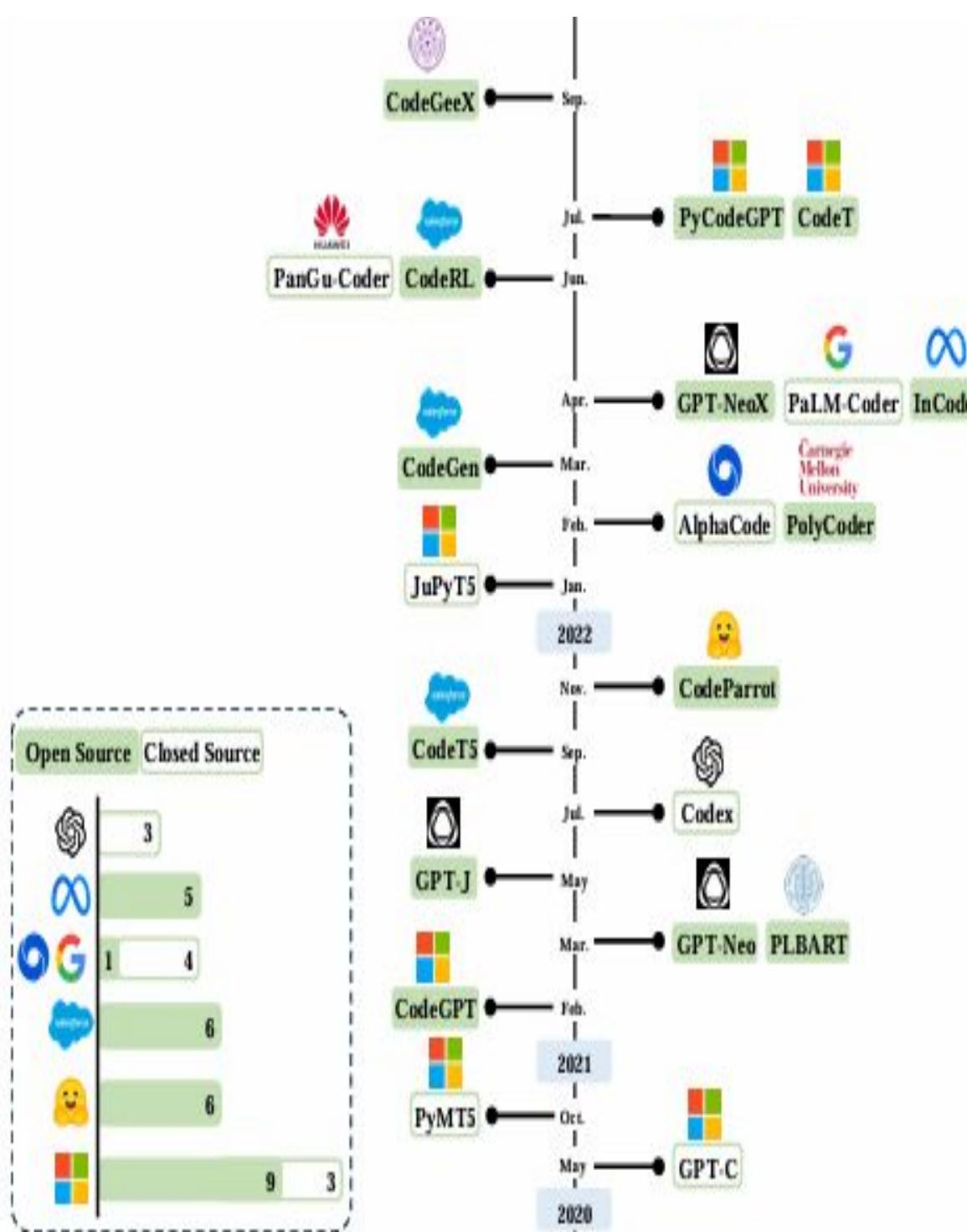
Cross  
Language  
Flexibility



Task  
Specific  
Outputs



# Large Language Models Trained for Code Generation



Source- [A Survey on Large Language Models for Code Generation](#)

# Datasets used for Training LLMs for Code Generation

<b>Dataset</b>	<b>Size (GB)</b>	<b>Files (M)</b>	<b>#PL</b>
CodeSearchNet [110]	20	6.5	6
Google BigQuery[96]	-	-	-
The Pile [78]	95	19	-
CodeParrot [254]	180	22	1
GitHub Code[254]	1,024	115	32
ROOTS [137]	163	15	13
The Stack [132]	3,136	317	30
The Stack v2 [170]	32K	3K	619

Datasets used for pre-training LLMs

#PL = Number of programming languages included in the dataset

<b>Dataset</b>	<b>Size</b>	<b>#PL</b>
CodeAlpaca-20K [43]	20k	-
CommitPackFT [187]	2GB	277
Evol-Instruct-Code-80k [225]	80k	-
evol-codealpaca-v1 [251]	110K	-
Magocoder-OSS-Instruct-75k [278]	75k	Python, Shell, TypeScript, C++, Rust, PHP, Java, Swift, C#
Self-OSS-Instruct-SC2-Exec-Filter-50k [304]	50k	Python

Datasets used for instruction tuning LLMs

Source- [A Survey on Large Language Models for Code Generation](#)

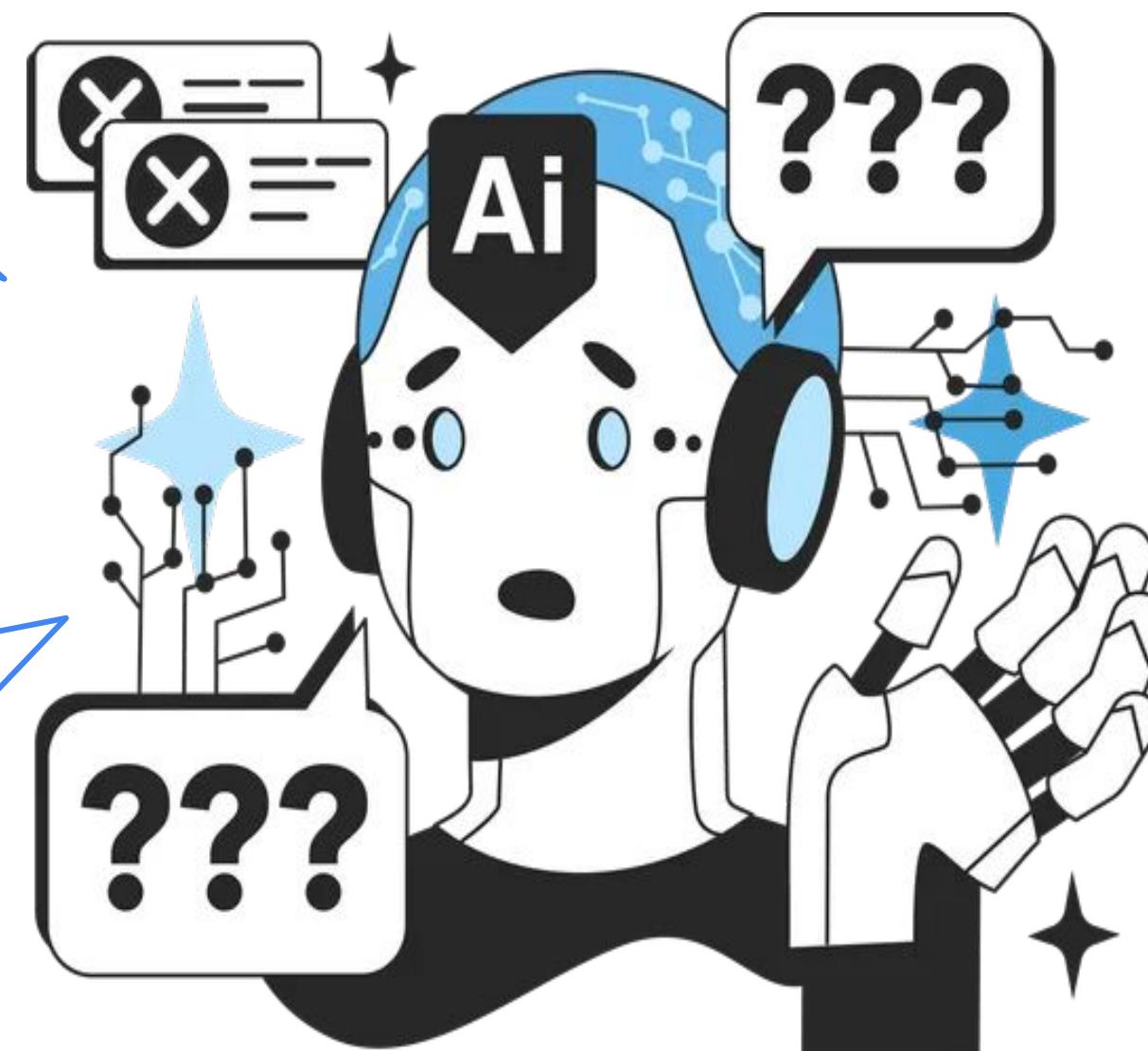
# Challenges of Code Generation with Generative AI

Understanding Natural and Programming Languages  
Both Which can be confusing

Balancing Readability, Efficiency, Correctness, and Completeness

Handling Diverse Input/Output Requirements with varying Complexities

Due to the inherent complexity of programming tasks LLMs can Hallucinate



# Prompt Engineering to the Rescue

**Be specific:** Clearly define task requirements and expected functionality.

**Specify language:** Mention the programming language in the prompt explicitly.

**Include examples:** Provide input-output examples to clarify expected behavior.

**State libraries:** List required libraries or frameworks for the task.

**Ask for comments:** Request code with inline comments for better understanding.

**Limit complexity:** Break complex tasks into smaller, manageable subtasks for clarity.

**Focus on constraints:** Specify performance, scalability, or formatting requirements.

**Iterate prompts:** Refine and retry prompts to improve the generated code.

**Use templates:** Provide partial code or templates for the model to follow.

**Request explanations:** Ask for explanations alongside code to verify logic and intent



# AI Tools For Code Generation

	 GitHub Copilot	 ChatGPT	 tabnine	 Amazon Q Developer	 Devin
<b>Supported Languages</b>	All Languages	All Popular Languages	20 Programming Languages	15 Programming Languages	All Popular Languages
<b>Integration</b>	Works with Visual Studio Code, JetBrains IDEs, and Neovim	Works through a chat interface	Works with popular IDEs like VS Code, JetBrains, Sublime Text	Integrated into IDEs such as VS Code, JetBrains, and AWS Cloud9	Works with IDEs like VS Code and JetBrains
<b>Underlying Model</b>	OpenAI Codex	GPT-4o, o1	Tabnine Model	Unknown	Diverse
<b>Price</b>	Starting at \$10 Per Month	Pro Plan Starts \$20/Month	Pro Plan Costs \$12/Month	Pro Version Costs \$19/Month	Basic Plan Starts at \$25/Month
<b>Code Completion</b>	✓	✓	✓	✓	✓
<b>Code Generation</b>	✓	✓	✓	✓	✓
<b>Debugging Assistance</b>	✓	✓	✓	✓	✓
<b>Documentation Generation</b>	✓	✓	✓	✗	✗

# Comparing Relevant Code Generation Tools

Features	ChatGPT	Amazon CodeWhisperer	GitHub Copilot
IDE Support	No IDE Support	JetBrains, Visual Studio Code, AWS Cloud9, or the AWS Lambda console	IntelliJ IDEA, Android Studio, AppCode, CLion, Code With Me Guest, DataGrip, DataSpell, GoLand, JetBrains Client, MPS, PhpStorm, PyCharm, Rider, RubyMine, WebStorm
First Release Time	Nov-30-2022	June-23-2022	Oct-29-2021
Developer	OpenAI	AWS	OpenAI-Microsoft
Providing References to Suggestions	NO	YES	NO
Explanation of Suggestions	YES	NO	NO
Providing Multiple Suggestions	NO (Theoretically user can manually ask for another suggestion.)	YES (Up to 5)	YES (Up to 10)
Training Data Source	GitHub Repositories, OpenAI Codex Dataset, other code repositories such as GitLab, Bitbucket, and SourceForge	"Vast amounts of publicly available code"	"...trained on all languages that appear in public repositories" (Fine-tuned)
Programming Languages work best with (according to the vendor)	N/A	C#, Java, JavaScript, Python, and TypeScript	C, C++, C#, Go, Java, JavaScript, PHP, Python, Ruby, Scala, and TypeScript
Multipurpose (other than programming)	YES	NO	NO
Subscription	ChatGPT Free ChatGPT Plus (\$20 per month)	Free Preview	Copilot for Students (Free) Copilot for Individuals (\$10 per month) Copilot for Business (\$19 per user, per month)
Can be Used Offline?	NO	NO	NO
Can it Access Local Files?	NO	YES	YES

Getting started with



Free Version

# What features are included in Copilot Free?

Code completion	in different IDEs Visual Studio Code, Visual Studio, JetBrains IDES, Vim/Neovim, Xcode, and Azure Data Studio
Copilot Edits	to make changes across multiple files (only in Visual Studio Code and Visual Studio)
Copilot Chat	Chat window in Visual Studio Code, Visual Studio, JetBrains IDES, and GitHub.com
Models available	Claude 3.5 Sonnet and GPT 4o

# What are the limitations of Copilot Free?

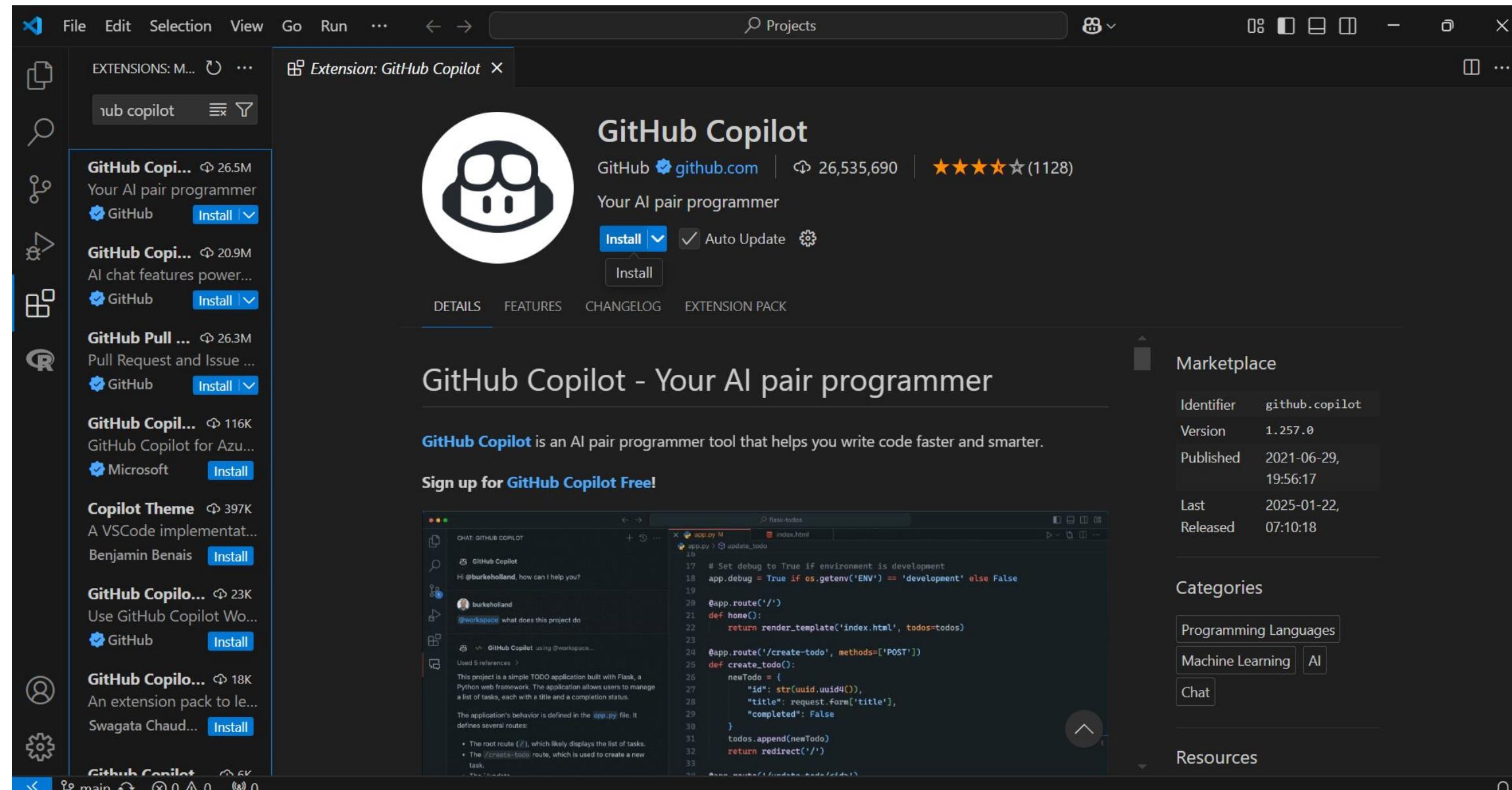
Code completions	2000 completions per month.
Copilot Chat	50 chat messages per month. This limit includes both standard chats and multi-file editing chats in VS Code and Visual Studio

When you reach these limits, you can upgrade to Copilot Pro to continue using Copilot.

# Set up Visual Studio Code with Copilot

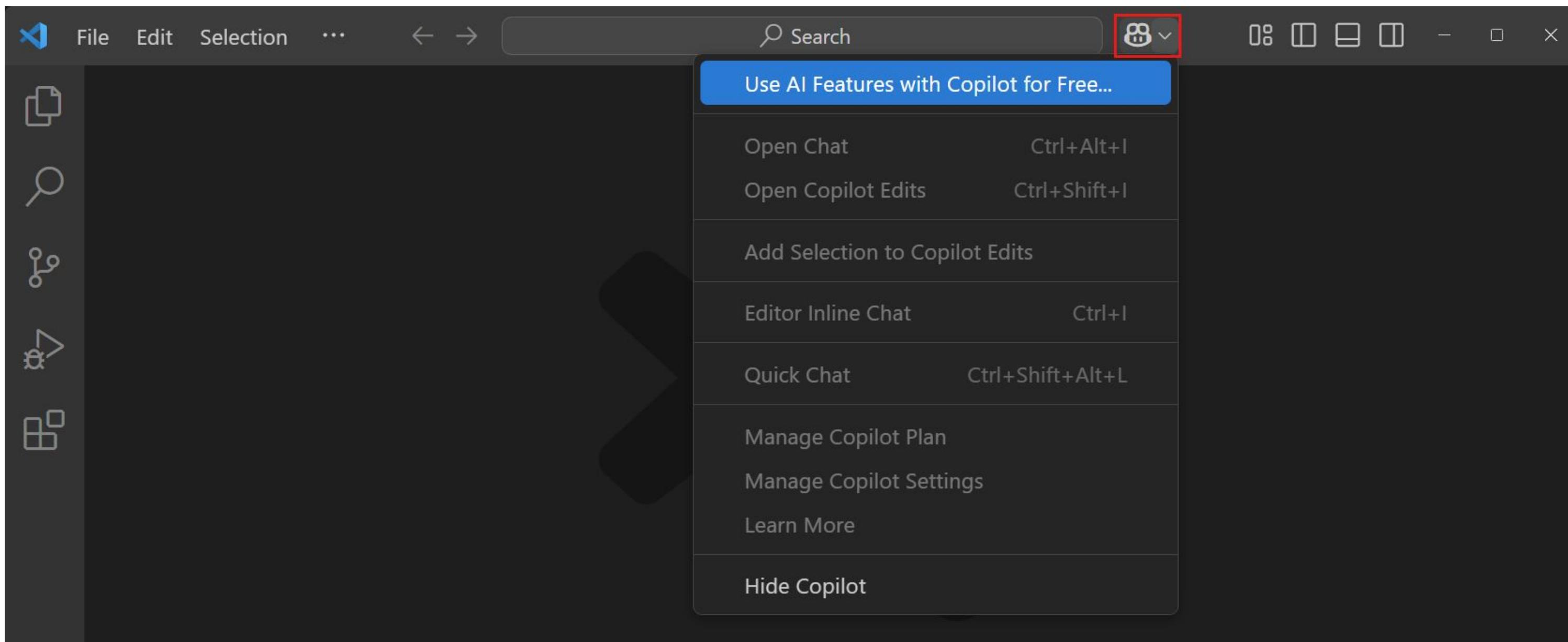
To get started with Copilot Free in Visual Studio, you need:

1. Ensure you have a recent version of VS Code
2. GitHub Copilot Extension in VS Code



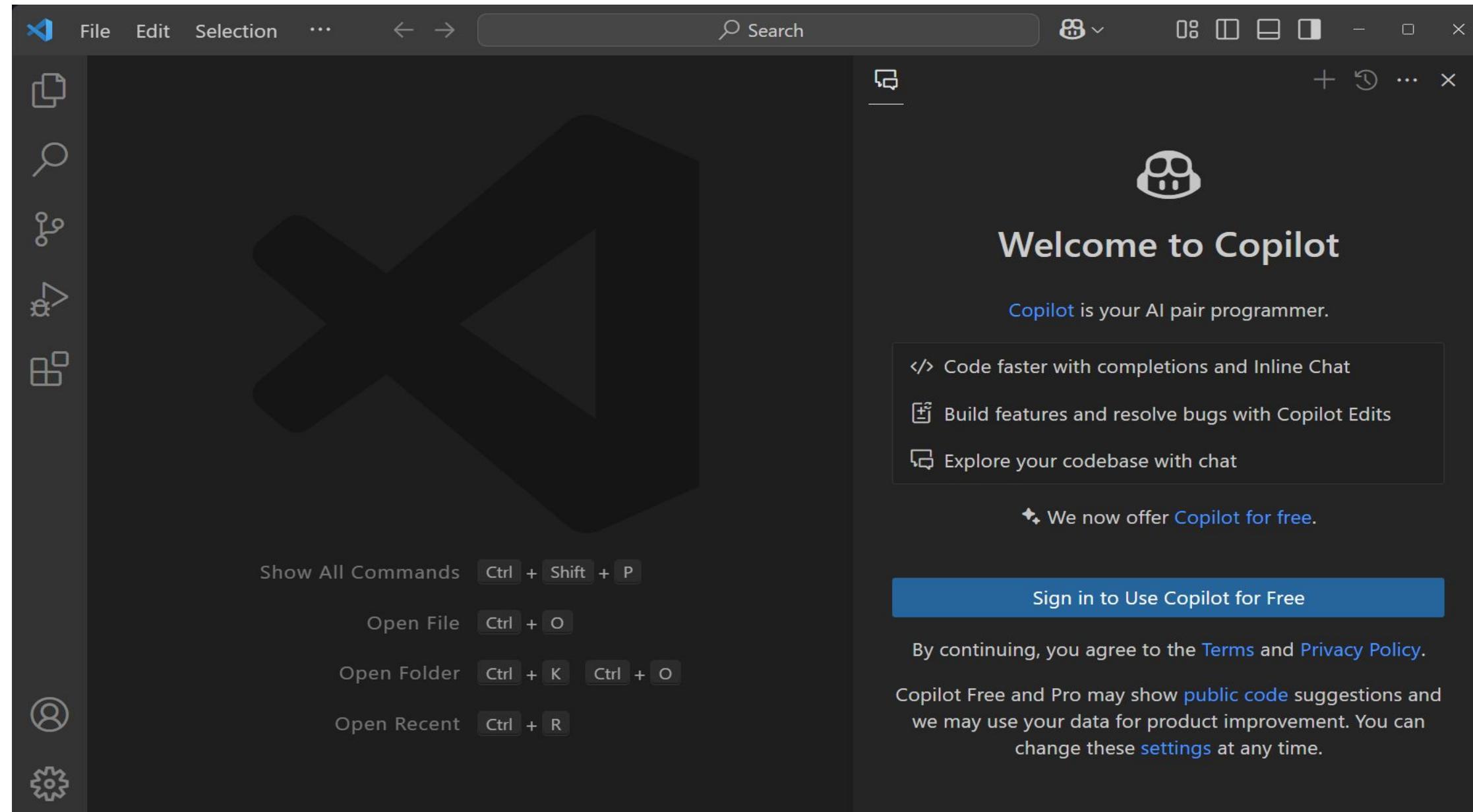
# Set up Visual Studio Code with Copilot

3. Select Use AI Features with Copilot for Free... from the Copilot menu in the title bar or from the Command Palette (Ctrl+Shift+P)



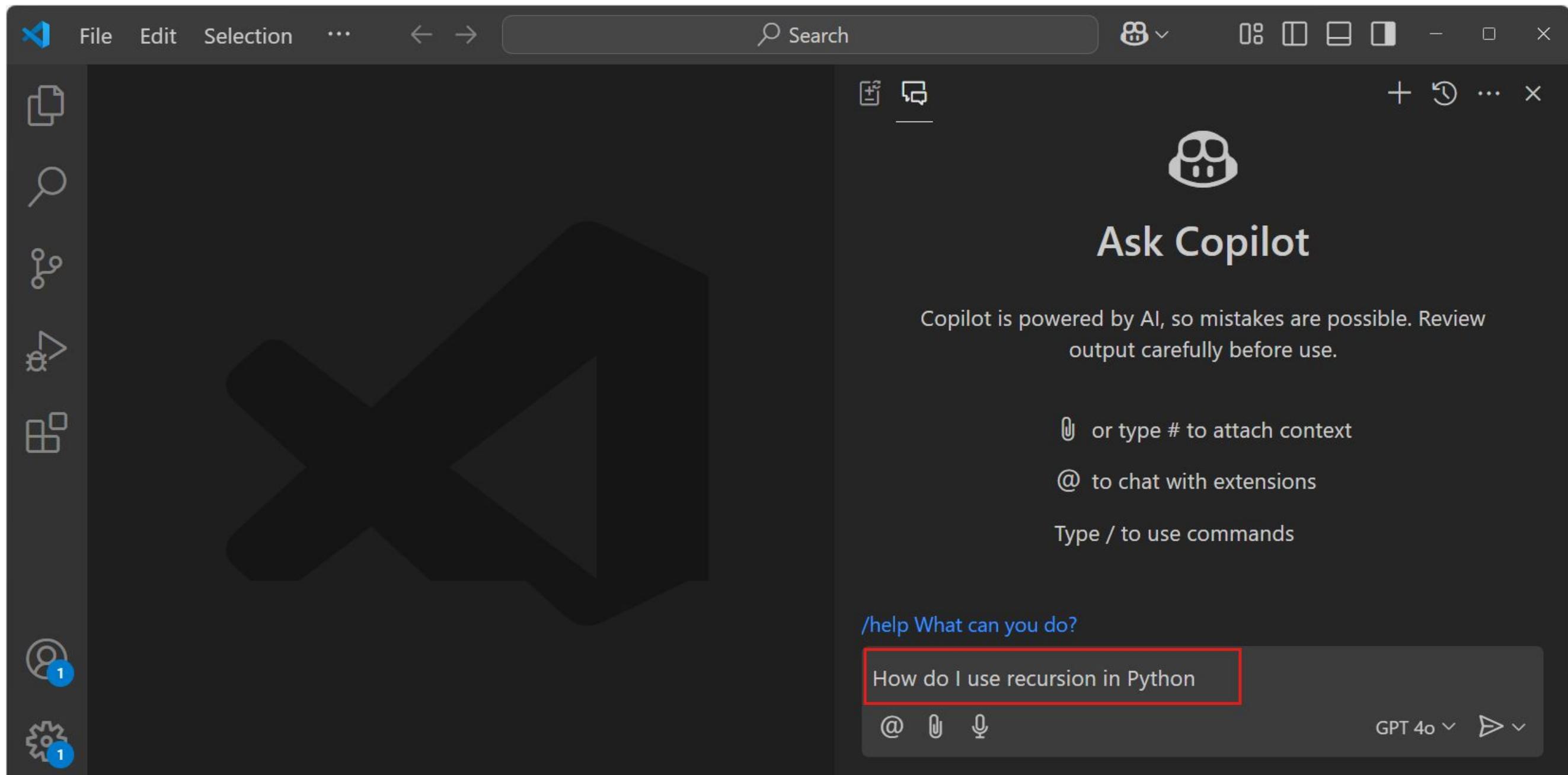
# Set up Visual Studio Code with Copilot

4. Select Sign in to Use Copilot for Free to sign in to your GitHub account and sign up for Copilot Free  
If you already have a Copilot subscription associated with your GitHub account, VS Code uses that one after you sign in.



# Set up Visual Studio Code with Copilot

5. Get started by entering a prompt in the chat input field



# Using GitHub Copilot method 1

Start **typing code** and accept suggestions with the Tab key  
Writing code for evaluating a model with help of copilot

The screenshot shows a Jupyter Notebook interface with a dark theme. The top bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar labeled 'try-github-copilot'. Below the toolbar, there are tabs for Code, Markdown, Run All, Restart, Clear All Outputs, Variables, and Outline. A status bar indicates 'venv (Python 3.10.11)'. The main area displays Python code for training a linear regression model:

```
# Train a linear regression model
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
```

A tooltip from GitHub Copilot is visible, showing the suggestion 'LinearRegression()' with a question mark icon. The bottom of the screen features a status bar with icons for Run, Stop, and Cell, along with a message: 'Press [ctrl]+[space] to ask GitHub Copilot to do something. Start typing to dismiss.'

## Using GitHub Copilot method 2 -

Use // and write your requirements Accept suggestions with the Tab key  
Generating a function that retrieves DB items by priority

```
0 references | 0 changes | 0 authors, 0 changes
39     public static void CreateTables()
40     {
41         using (var context = new TaskContext())
42         {
43             context.Database.ExecuteSqlRaw("CREATE TABLE tasks (id INT PRIMARY KEY, title VARCHAR(50), priority INT)");
44         }
45     }
46
47     [ ] I
48
49
50
51
52
53
54
55
56
57
58
59     }
60 }
```

## Using GitHub Copilot method 3 -

Use **ctrl + I** and write your requirements and click on accept to accept changes  
Asking copilot to make code readable in javascript

The screenshot shows a Visual Studio Code window titled "utils.js - Untitled (Workspace) - Visual Studio Code - Insiders". The file contains the following JavaScript code:

```
97 }
98
99 function resetFontWeight(defaultFontWeight) {
100     $('#note').css('font-weight', defaultFontWeight);
101     $('#fontWeight').val(defaultFontWeight);
102 }
103
104 function resetShowWordCountPill(defaultShowWordCountPill) {
105     defaultShowWordCountPill === 'Yes' ? $('.word-count-container').show() : $('.word-count-container').hide();
106     $('#showWordCountPill').val(defaultShowWordCountPill);
107 }
108
109 function countWords(str) {
110     return str.trim().split(/\s+/).length;
111 }
112
113 function calculateCharactersAndWords(str) {
114     const characterCount = str.length;
115     const wordCount = str !== '' ? countWords(str) : 0;
116     const wordCountText = `${characterCount} character(s), ${wordCount} word(s)`;
117
118     return wordCountText;
119 }
```

The status bar at the bottom shows: master 0 0 △ 0, Ln 102, Col 2, Spaces: 4, UTF-8, LF, {}, JavaScript.

## Using GitHub Copilot method 4 -

Use **Github copilot chat**  
to generate code and apply  
changes using apply to  
editor button

Asking copilot to create a  
realistic invoice model with  
line items in C-sharp

The screenshot shows the Visual Studio Code interface with the GitHub Copilot extension active. In the left sidebar, under the 'CHAT' section, there is a conversation with 'GitHub Copilot'. The message from Copilot reads:

Welcome @mvanhil, I'm your Copilot and I'm here to help you get things done faster. You can also [start an inline chat session](#).

I'm powered by AI, so surprises and mistakes are possible. Make sure to verify any generated code or suggestions, and [share feedback](#) so that we can learn and improve. Check out the [Copilot documentation](#) to learn more.

At the bottom of the sidebar, there is a list of commands:

- ❖ /fix the problems in my code
- ❖ /tests add unit tests for my code
- ❖ /explain how the selected code works

A text input field at the bottom says "Ask Copilot or type '/' for commands".

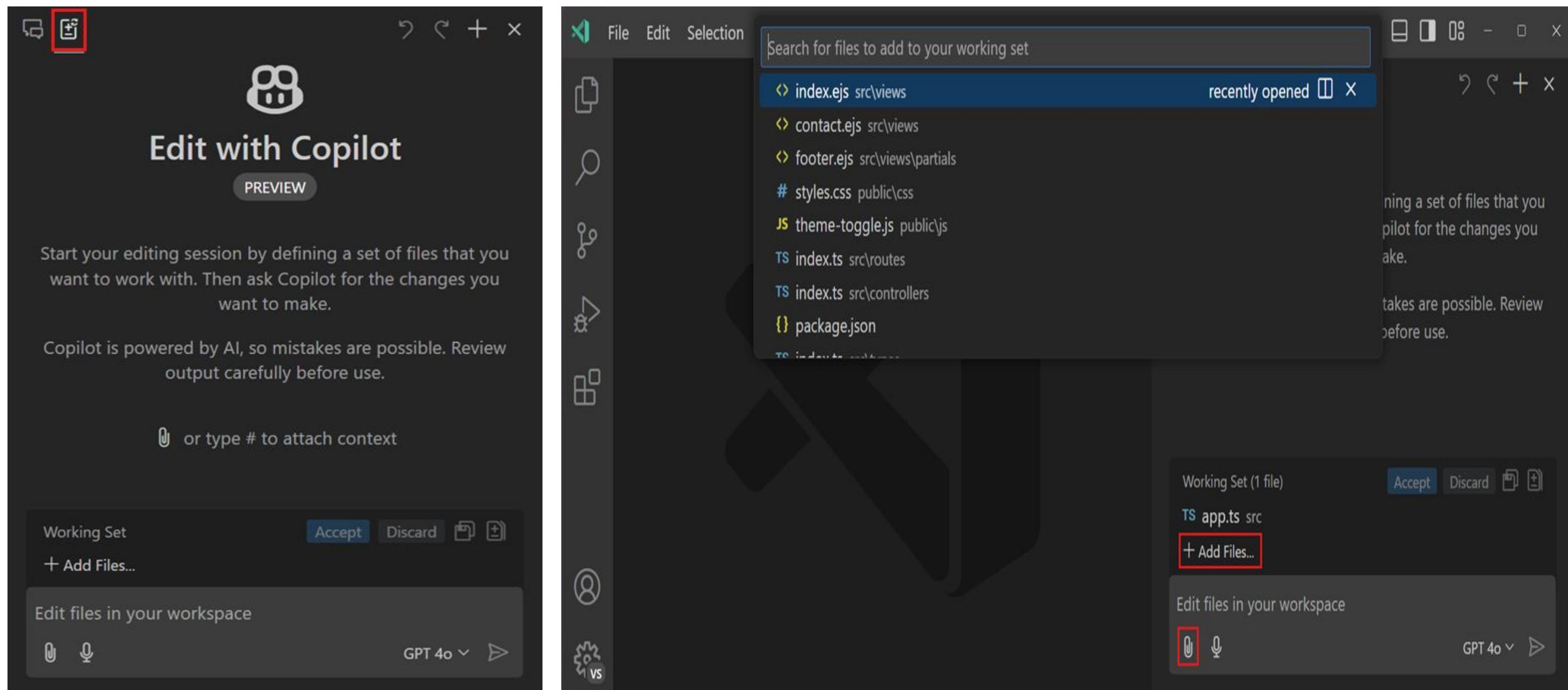
In the main editor area, a file named 'Invoice.cs' is open, showing the following code:

```
C: > Users > MarnikVanHileghem > OneDrive - Cronos > Documenten >
1 Press [Ctrl]+[I] to ask GitHub Copilot Ch
to do something. Start typing to dismiss.
```

The status bar at the bottom indicates: Ln 1, Col 1 Spaces: 4 UTF-8 CRLF C#

## Using GitHub Copilot method 5 -

**Using Copilot edits** - Copilot Edits proposes code changes across multiple files in your workspace. These edits are applied directly in the editor, so you can quickly review them in-place, with the full context of the surrounding code.



## Using GitHub Copilot method 5 -

**Using Copilot edits** - Copilot Edits proposes code changes across multiple files in your workspace. These edits are applied directly in the editor, so you can quickly review them in-place, with the full context of the surrounding code.

The screenshot shows a VS Code workspace for a "flask-chatbot" project. The Explorer sidebar lists files like `__init__.py`, `routes.py`, `index.html`, `style.css`, and `requirements.txt`. The `index.html` file is open in the editor, showing HTML code for a chatbot interface. A floating panel titled "Copilot Edits (Ctrl+Shift+I)" displays suggestions for modifying the `style.css` file:

- Copilot Edits (Ctrl+Shift+I)**: Suggests making the chatbot interface more beautiful by fitting it to the whole screen, adding icons for sender and receiver, and making the send button green.
- GitHub Copilot**: Suggests updating the CSS to fit the whole screen, add icons, and make the send button green. It also lists changes for `style.css` (+41 -36) and `routes.py`.
- No changes needed in this file for the requested modifications.**

The bottom right of the panel shows a "Working Set (5 files)" list with items: `# style.css`, `__init__.py`, `index.html`, `__init__.py`, and `routes.py`. Buttons for "Accept" and "Discard" are also present.

The terminal at the bottom shows the command prompt: `(myenv) C:\Users\Charu\Desktop\Projects\flask-chatbot>`. The status bar at the bottom right indicates "GPT 4o" and a "Next" button.

# Added Extensions - Vision for Copilot



**Vision for Copilot** is an extension that enhances chat by allowing users to attach images as contextual input, enabling dynamic, visually-supported responses for richer interactions.

The screenshot shows the VS Code interface with the "Vision for Copilot" extension installed. In the top right corner, there is a blue eye icon with a white "C" inside, which is the extension's logo. The main area displays a chat window where a user (@justschen) has asked for HTML and CSS for a landing page. Below the message, a screenshot of a landing page is attached. The response from the extension is a snippet of HTML and CSS code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Furniture Landing Page</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;
      background-color: #f7ece2;
      color: #6c4f3d;
    }

    .container {
      width: 90%;
      max-width: 1200px;
      margin: 0 auto;
    }

    .header {
      padding: 20px;
      text-align: left;
    }

    .title {
      font-size: 48px;
      font-weight: bold;
      color: #8c7a5b;
    }

    .subtitle {
      font-size: 24px;
      margin-bottom: 20px;
    }

    .description {
      max-width: 500px;
      font-size: 16px;
      line-height: 1.5;
    }

    .links {
      margin-top: 20px;
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="header">
      <h1>Furniture Landing Page</h1>
    </div>
    <div class="title">
      <h2>Welcome to Our Furniture Store</h2>
    </div>
    <div class="subtitle">
      <p>We offer a wide range of high-quality furniture pieces at competitive prices. Explore our collection today!</p>
    </div>
    <div class="description">
      <p>Our store features a variety of furniture styles, including modern, rustic, and traditional. We pride ourselves on providing excellent customer service and fast shipping. Visit us online or in-store to see our full selection.</p>
    </div>
    <div class="links">
      <a href="#">Shop Now</a>
      <a href="#">About Us</a>
      <a href="#">Contact Us</a>
    </div>
  </div>
</body>

```

# Application 1 - Adding Comments

The screenshot shows a dark-themed instance of Visual Studio Code (VS Code) with the GitHub Copilot extension installed. The main area displays a Python script named `model.py` from a repository titled "FACE\_RECOGNITION". The code implements a SiameseNetwork module using PyTorch, specifically utilizing ResNet18 as the base model and adding a new fully connected layer for distance calculation.

In the bottom right corner of the editor, the GitHub Copilot interface is visible. It shows a suggestion card for the user "charurathour" with the title "Add comments" and a snippet of code for lines 1-26 of `model.py`. This card is highlighted with a red rectangle. Below the card, the GitHub Copilot logo and the text "Used 1 reference" are shown. A preview of the suggested code is displayed, which includes imports and the `__init__` method of the `SiameseNetwork` class, along with a note to "Apply in Editor".

At the bottom of the interface, there is a section titled "What comments should I add?" with a "Ask Copilot" button and a dropdown menu set to "model.py Current file". The status bar at the bottom of the screen shows the current file is "main\*", the line number is 22, the column is 43, and the encoding is UTF-8.

```
import torch
import torch.nn as nn
import torchvision.models as models

class SiameseNetwork(nn.Module):
    def __init__(self):
        super(SiameseNetwork, self).__init__()
        self.base_model = models.resnet18(pretrained=True)
        self.base_model = nn.Sequential(*list(self.base_model.children())[:-1])
        self.fc = nn.Sequential(
            nn.Linear(512, 1),
            nn.Sigmoid()
        )

    def forward_one(self, x):
        x = self.base_model(x)
        x = x.view(x.size(0), -1)
        return x

    def forward(self, input1, input2):
        output1 = self.forward_one(input1)
        output2 = self.forward_one(input2)
        l1_distance = torch.abs(output1 - output2)
        output = self.fc(l1_distance)
        return output
```

# Application 2 - Generating Explanation of Existing Code

The screenshot shows a VS Code interface with the following elements:

- EXPLORER** sidebar: Shows a project named "FACE\_RECOGNITION" with files: Data, model.py, pairs.py (selected), README.md, test.py, and train.py.
- Code Editor**: The active file is pairs.py, which contains Python code for face recognition. The code defines functions to load images from a folder, create pairs of images, and split them into training, validation, and test sets.
- GitHub Copilot Panel**: A right-hand panel titled "charurathour" displays a prompt: "Explain what this code does" with a red box around it. Below the prompt, it says "pairs.py:45-68". It also lists "Used 1 reference" and provides a detailed explanation: "This code defines functions to preprocess and split image pairs for training, validation, and testing in a face recognition project." It highlights the "Functions:" section and lists the "split\_and\_preprocess\_pairs(pairs, labels)" function with its purpose: "Splits the input pairs and labels into training, validation, and test sets." and "Preprocesses the image pairs by applying the preprocess\_image function to each image in the pairs."
- Bottom Status Bar**: Shows file information like "main\* 1 0 0", line and column counts (Ln 51, Col 33), and file encoding (UTF-8).

# Application 3 - Code Review

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with a dark theme. The left sidebar displays a project structure under 'PROJECTS' for 'flask-chatbot > app > routes.py'. The main editor window shows Python code for a Flask application. A red box highlights a comment from 'charurathour': 'review this code according to PEP 8' with a link to 'routes.py:1-16'. The bottom right corner features the GitHub Copilot interface, which includes a list of PEP 8 guidelines, a command palette, and a text input field for asking questions.

EXPLORER test.py routes.py

flask-chatbot > app > routes.py > ...

```
1  from flask import render_template, request, jsonify
2  from app import app
3  from transformers import AutoModelForCausalLM, AutoTokenizer
4  import torch
5
6  # Load the quantized model and tokenizer from Hugging Face
7  model_name = "microsoft/DialoGPT-medium"
8  tokenizer = AutoTokenizer.from_pretrained(model_name)
9  model = AutoModelForCausalLM.from_pretrained(model_name, torch_dtype=_
10
11 @app.route('/')
12 def index():
13     return render_template('index.html') # Render the main HTML tem
14
15 @app.route('/chat', methods=['POST'])
16 def chat():
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS + v ... ^ X

(new\_env) C:\Users\Charu\Desktop\Projects>

charurathour  
review this code according to PEP 8  
routes.py:1-16

GitHub Copilot  
Used 1 reference  
Here is a review of the code according to PEP 8 guidelines:

- Imports:**
  - Group imports into standard library imports, related third-party imports, and local application/library-specific imports.
  - Add a blank line between each group of imports.
- Line Length:**
  - Ensure lines do not exceed 79 characters.
- Blank Lines:**
  - Surround top-level function and class definitions with two blank lines.
- Comments:**
  - Ensure comments are complete sentences and have a space after the `#` symbol

Ask Copilot  
routes.py Current file  
@ 0 0 GPT 4o

# Application 4 - Debugging

The screenshot shows a VS Code interface with the following details:

- EXPLORER**: Shows a project structure with files like `test.py`, `requirements.txt`, `flask-chatbot`, `app`, `_pycache_`, `static`, `templates`, `__init__.py`, `routes.py`, `README.md`, and `requirements.txt`.
- TERMINAL**: Displays the following error message:

```
cli.py", line 260, in locate_app
    __import__(module_name)
File "C:\Users\Charu\Desktop\Projects\flask-chatbot\app\__init__.py", line 7, in <module>
    from app import routes
File "C:\Users\Charu\Desktop\Projects\flask-chatbot\app\routes.py", line 3, in <module>
    from transformers import AutoModelForCausalLM, AutoTokenizer
ModuleNotFoundError: No module named 'transformers'
```
- OUTPUT**: Shows a dropdown menu with options: powershell, cmd, and powershell.
- DEBUG CONSOLE**: Not visible in the screenshot.
- PROBLEMS**: Shows the same error message as the terminal.
- COMMENTS**: Not visible in the screenshot.
- GitHub Copilot**: A sidebar suggestion box for the error message:

The error `ModuleNotFoundError: No module named 'transformers'` indicates that the `transformers` package is not installed in your current Python environment. This package is required for your application. You can install it using pip:

```
pip install transformers
```

Ask Copilot

requirements.txt Current file

GPT 4o
- STATUS BAR**: Shows the path `(chatbot_env) C:\Users\Charu\Desktop\Projects\flask-chatbot>`, line 5, col 6, spaces: 4, UTF-8, LF, pip requirements, and icons for main, outline, timeline, and search.

# Application 5 - Language Conversion

The screenshot shows the Visual Studio Code interface with a dark theme. On the left is the Explorer sidebar with project files like 'flask-calculator-app' and 'flask-chatbot'. The main area shows 'test.py' with the following code:

```
1 class AddTwoNumbers:
2
3     # Function to add two numbers
4     @staticmethod
5     def add_numbers(num1, num2):
6         return num1 + num2
7
8     # Main method to test the function
9     if __name__ == "__main__":
10        number1 = 10 # First number
11        number2 = 20 # Second number
12
13    # Call the add_numbers function and print the result
14    sum = AddTwoNumbers.add_numbers(number1, number2)
15    print(f"The sum of {number1} and {number2} is: {sum}")
16
```

The terminal at the bottom shows:

```
(chatbot_env) C:\Users\Charu\Desktop\Projects\flask-chatbot>
```

A red box highlights the Java code completion suggestion in the right-hand panel:

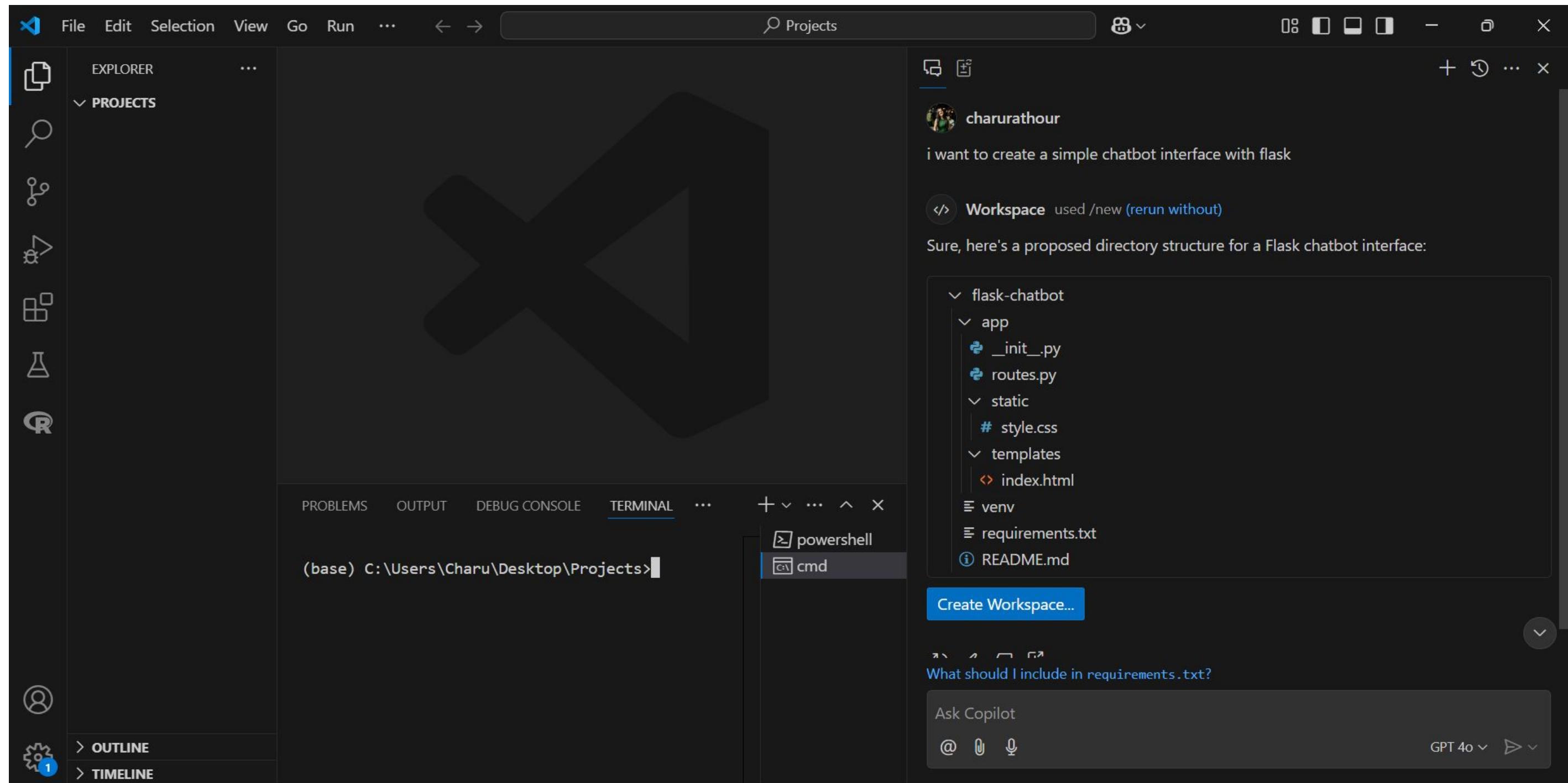
```
public class AddTwoNumbers {
    // Function to add two numbers
    public static int addNumbers(int num1, int num2) {
        return num1 + num2;
    }

    // Main method to test the function
    public static void main(String[] args) {
        int number1 = 10; // First number
        int number2 = 20; // Second number

        // Call the addNumbers function and print the result
        int sum = addNumbers(number1, number2);
        System.out.println("The sum of " + number1 + " and " + number2);
    }
}
```

Below the Java code, there's a note: "convert this function in python". The status bar at the bottom indicates "test.py Current file" and "GPT 4o".

# Application 6 - Creation of Workspace with Copilot



# Use Case - Building a Chatbot Application with GitHub Copilot

## Step 1: Setting Goals and Structuring Your Workflow

### Set Your Goal

- Define what you want to build.
- Example: "I want to create a chatbot for answering basic user queries."
- Be specific about the problem your application will solve.

### Identify Requirements

- What does your application need?
- Input: What will the user provide?
- Example: Text queries in a chat window.
- Output: What will the app deliver?
- Example: AI-generated text responses.

### Decide on the Tech Stack

- **Language:** Choose the programming language.  
Example: Python for backend and AI models.
- **Libraries and Frameworks:** List what you'll need.  
Example: Flask for backend, Transformers from Hugging Face for chatbot logic, HTML/JavaScript for frontend.
- **Backend:** Will you use Flask/Django/Node.js, or something else?
- **Frontend:** Web-based interface (HTML/JS) or other frameworks like React.

You can use Copilot chat to research and ideate for setting up goal

# Step 1: Setting Goals and Structuring Your Workflow

The screenshot shows a dark-themed code editor interface, likely Visual Studio Code. In the top left, the navigation bar includes File, Edit, Selection, View, Go, Run, and a three-dot menu. The top right features a search bar labeled 'Projects' and various window control icons. On the far left is a vertical sidebar with icons for Explorer, Search, Issues, Pull Requests, and Outline/Timeline.

The main area displays a terminal window titled '(base) C:\Users\Charu\Desktop\Projects>'. The terminal content is currently empty, showing only the command prompt.

In the center, there's a large dark overlay with a large white 'X' and some geometric shapes, possibly indicating a modal or a temporary UI element.

The bottom right corner contains a GitHub Copilot interface. It shows a suggestion from 'charurathour' for creating a simple chatbot, followed by a section titled 'GitHub Copilot' with instructions for creating a chatbot. A numbered list provides steps for building a chatbot:

- 1. Programming Language:** Choose a programming language like Python, JavaScript, or any other language you are comfortable with.
- 2. Environment Setup:** Set up your development environment with the necessary libraries and tools.
- 3. Chatbot Logic:** Implement the logic for your chatbot, including how it will process user input and generate responses.
- 4. Natural Language Processing (NLP):** Use NLP libraries or APIs to understand and process user input. Popular libraries include NLTK, spaCy, or APIs like Dialogflow.
- 5. User Interface:** Create a user interface for interacting with the chatbot. This could be a command-line interface, a web interface, or integration with messaging platforms like Slack or Facebook Messenger.
- 6. Testing:** Test your chatbot to ensure it responds correctly to various inputs.

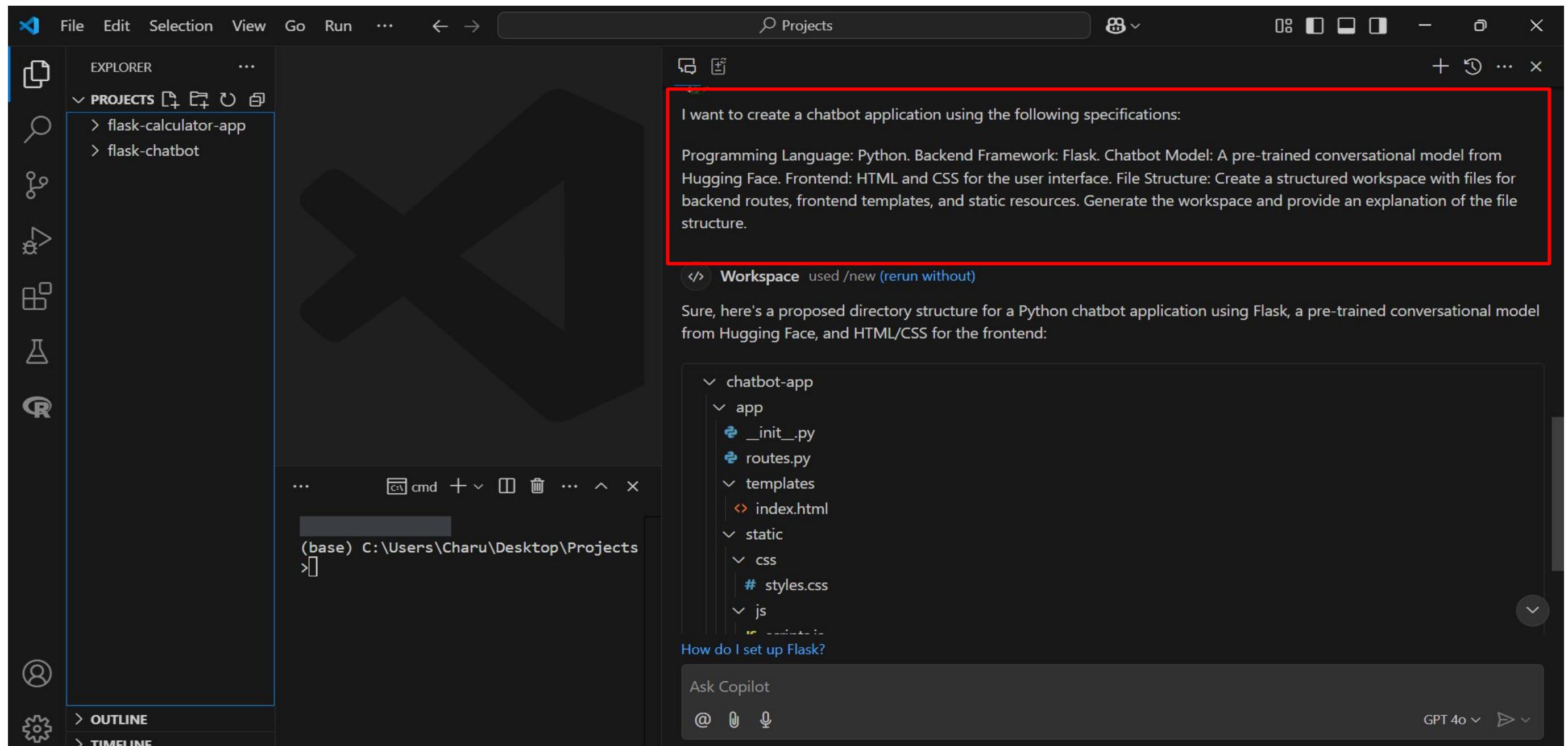
Below the list, a snippet of Python code is shown as an example:

```
def respond(user_input):  
    if "hello" in user_input:  
        return "Hello!"  
    elif "time" in user_input:  
        return str(datetime.now())  
    else:  
        return "I'm sorry, I didn't understand that."  
  
if __name__ == "__main__":  
    while True:  
        user_input = input("User: ")  
        response = respond(user_input)  
        print("Bot:", response)
```

A tooltip from GitHub Copilot asks, 'What libraries should I use?' with options to 'Ask Copilot' or type '@' to search for specific libraries.

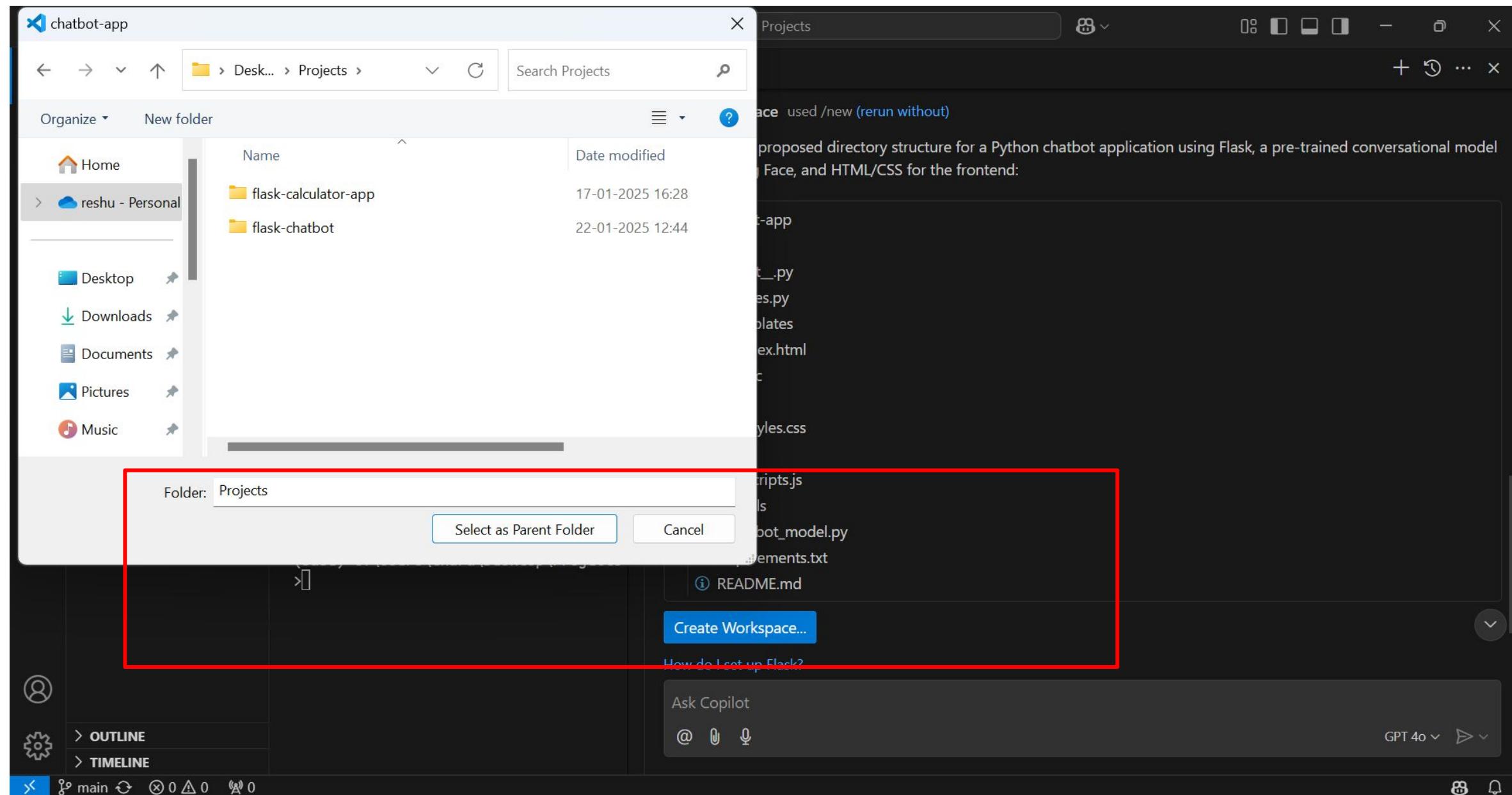
# Use Case - Building a Chatbot Application with GitHub Copilot

Step 2 : Use Copilot to create workspace by defining your goal and file structure in copilot chat area



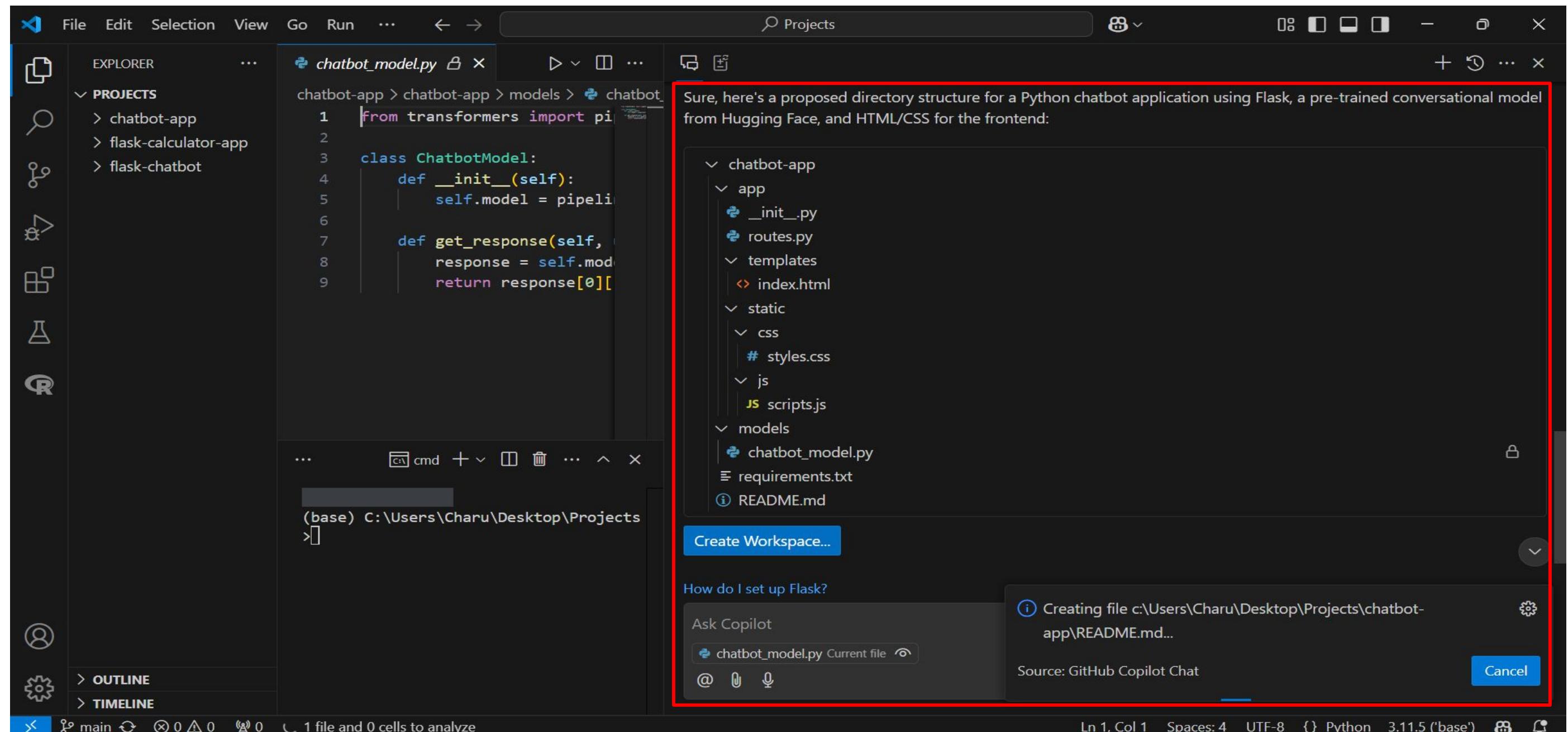
# Use Case - Building a Chatbot Application with GitHub Copilot

Step 2 : Use Copilot to create workspace by defining your goal and file structure in copilot chat area



# Use Case - Building a Chatbot Application with GitHub Copilot

Step 2 : Use Copilot to create workspace by defining your goal and file structure in copilot chat area



# Use Case - Building a Chatbot Application with GitHub Copilot

Step 3 : Create Environment and install dependencies from requirements file generated by copilot

The screenshot shows a Visual Studio Code (VS Code) interface with the following details:

- File Explorer (Left):** Shows a project structure under "chatbot-app". The "requirements.txt" file is selected.
- Terminal (Bottom):** Displays the command-line output of the pip installation process. A red box highlights the terminal output area.
- Output (Bottom):** Shows the status "In 1 Col 1 Spaces: 4 UTE-8 LF pip requirements".
- Right Side Panel:** Shows the contents of the "requirements.txt" file and the project's directory structure.
- Bottom Right:** Includes a "Create Workspace..." button and a GitHub Copilot sidebar with the text "How do I set up Flask?" and an "Ask Copilot" input field.

```
(base) C:\Users\Charu\Desktop\Projects\chatbot-app>conda activate chatbot_demo_env
(chatbot_demo_env) C:\Users\Charu\Desktop\Projects\chatbot-app>pip install -r requirements.txt
Collecting Flask==2.2.2 (from -r requirements.txt (line 1))
  Using cached Flask-2.2.2-py3-none-any.whl.metadata (3.9 kB)
Collecting transformers==4.21.1 (from -r requirements.txt (line 2))
  Using cached transformers-4.21.1-py3-none-any.whl.metadata (81 kB)
Collecting torch==1.12.1 (from -r requirements.txt (line 3))
  Downloading torch-1.12.1-cp39-cp39-win_amd64.whl.metadata (22 kB)
Collecting gunicorn==20.1.0 (from -r requirements.txt (line 4))
  Downloading gunicorn-20.1.0-py3-none-any.whl.metadata (3.8 kB)
Collecting requests==2.26.0 (from -r requirements.txt (line 5))
  Using cached requests-2.26.0-py2.py3-none-any.whl.metadata (4.8 kB)
Collecting Werkzeug>=2.2.2 (from Flask==2.2.2->-r requirements.txt (line 1))
  Using cached werkzeug-3.1.3-py3-none-any.whl.metadata (3.7 kB)
Collecting Jinja2>=3.0 (from Flask==2.2.2->-r requirements.txt (line 1))
  Using cached jinja2-3.1.5-py3-none-any.whl.metadata (2.6 kB)
Collecting itsdangerous>=2.0 (from Flask==2.2.2->-r requirements.txt (line 1))
  Using cached itsdangerous-2.2.0-py3-none-any.whl.metadata (1.9 kB)
Collecting click>=8.0 (from Flask==2.2.2->-r requirements.txt (line 1))
```

# Use Case - Building a Chatbot Application with GitHub Copilot

## Step 4 : Review code and make required changes

The screenshot shows a Visual Studio Code (VS Code) interface with a dark theme. On the left is the Explorer sidebar showing project files like `chatbot-app`, `app`, `models`, and `run.py`. The main editor area displays the `routes.py` file:

```
from flask import Flask, request, jsonify, render_template, Blueprint
from transformers import pipeline

app = Flask(__name__)
main = Blueprint('main', __name__)

# Load the pre-trained conversational model
chatbot = pipeline('conversational', model='microsoft/DialoGPT-D
```

A red box highlights the first few lines of the code. To the right of the editor, a sidebar titled "2. Set up routes" provides instructions and a preview of the code with suggested changes:

```
# filepath: /C:/Users/Charu/Desktop/Projects/chatbot-app/routes.py
from flask import Blueprint, render_template, request
from transformers import pipeline

main = Blueprint('main', __name__)

# Load the pre-trained conversational model
chatbot = pipeline('conversational', model='microsoft/DialoGPT-D
```

The bottom right corner shows the GitHub Copilot interface with fields for "Ask Copilot" and "GPT 4o".

# Use Case - Building a Chatbot Application with GitHub Copilot

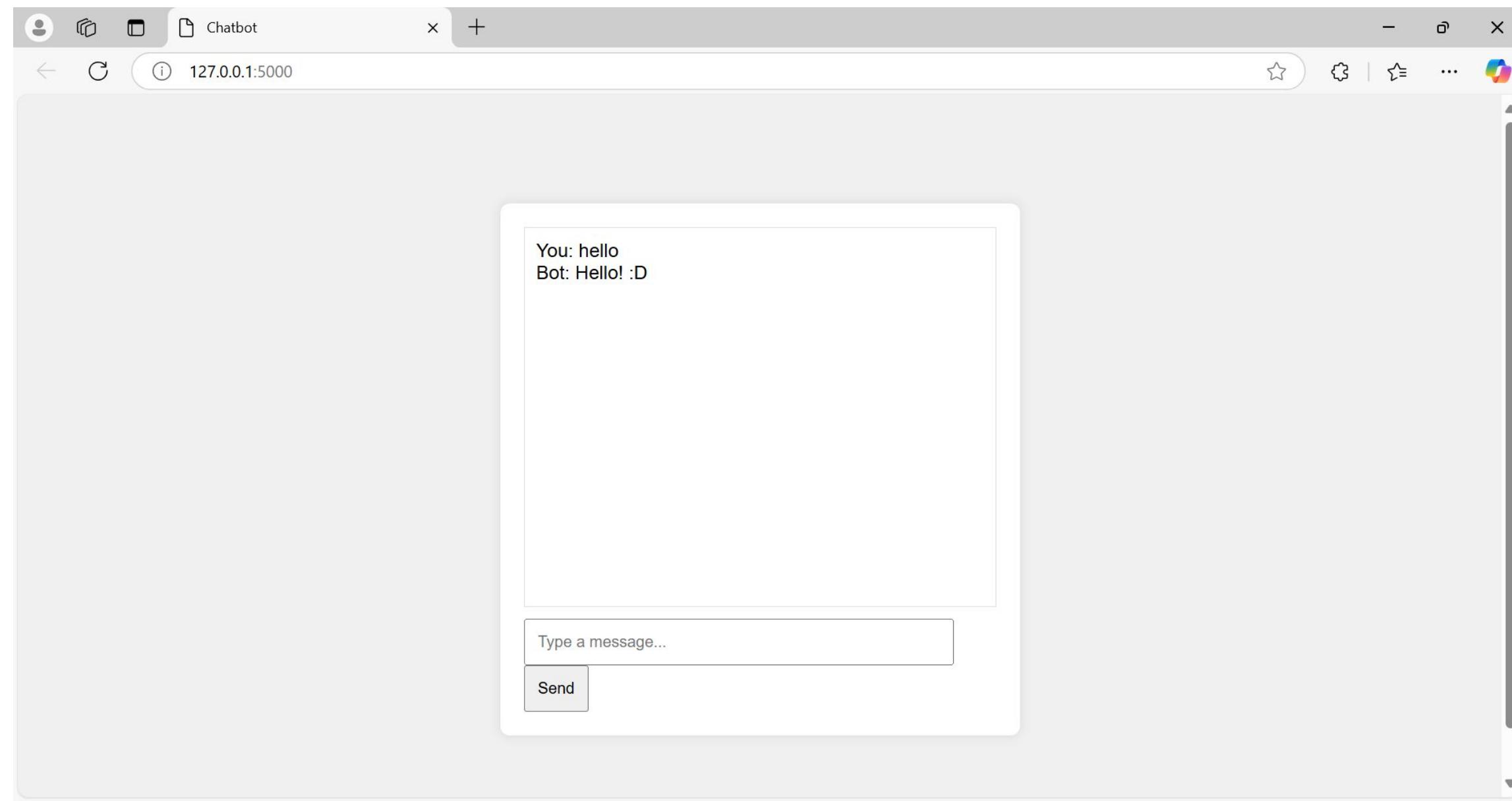
Step 5 : Run code and Debug with help of copilot if any errors come

The screenshot shows a VS Code interface with the following details:

- File Explorer:** Shows a project structure with files like `__init__.py`, `index.html`, `run.py`, and `styles.css`.
- Editor:** The `run.py` file is open, containing Python code for a Flask application.
- Terminal:** Shows the output of a command-line session with an `ImportError` related to the `url_quote` function.
- GitHub Copilot Panel:** A red box highlights the right-hand panel where GitHub Copilot provides a detailed error message and solutions.
  - Error Message:** An `ImportError` occurs due to an incompatibility between Flask and Werkzeug versions.
  - Solution 1: Downgrade Werkzeug** suggests downgrading Werkzeug to version 0.16.1.
  - Update Requirements:** It suggests updating `requirements.txt` to specify the compatible Werkzeug version.
- Status Bar:** Shows file statistics like "Ln 7, Col 24" and encoding "UTF-8".

# Use Case - Building a Chatbot Application with GitHub Copilot

Step 6 : Your basic chatbot is ready now you can iterate and make changes and update with copilot



# Use Case - Building a Chatbot Application with GitHub Copilot

Step 6 : Iterate and make changes and upgrade your application with copilot

The screenshot shows a VS Code interface with a dark theme. On the left is the Explorer sidebar showing project files like `chatbot-app`, `app`, `models`, and `run.py`. The `routes.py` file is open in the main editor area, containing Python code for a Flask application. A tooltip from GitHub Copilot suggests adding a header to `index.html`. Another tooltip suggests adding styles to `styles.css`. The bottom right shows a GitHub Copilot panel with a list of files and their edits, and a text input field for editing files.

```
from flask import Flask, request, jsonify, render_template
from models.chatbot_model import ChatbotModel

app = Flask(__name__)
main = Blueprint('main', __name__)

# Load the pre-trained conversational model
chatbot = ChatbotModel()

@main.route('/')
def home():
    return render_template('index.html')

@main.route('/chat', methods=['POST'])
def chat():
    user_input = request.json.get('message')
    response = chatbot.get_
    return jsonify({'response': response})
```

update interface by adding header of green color and heading Chatbot

GitHub Copilot

index.html

Add a header with a green background color and a heading "Chatbot".

index.html +3 -0

styles.css

Add styles for the header and heading.

# styles.css +13 -1

Working Set (7 files)

index.html chatbot-app\app\templa Accept All Edits (Ctrl+Enter)

\_init\_.py chatbot-app\app

routes.py chatbot-app\app

scripts.js chatbot-app\app\static\js

run.py chatbot-app

chatbot\_model.py chatbot-app\models

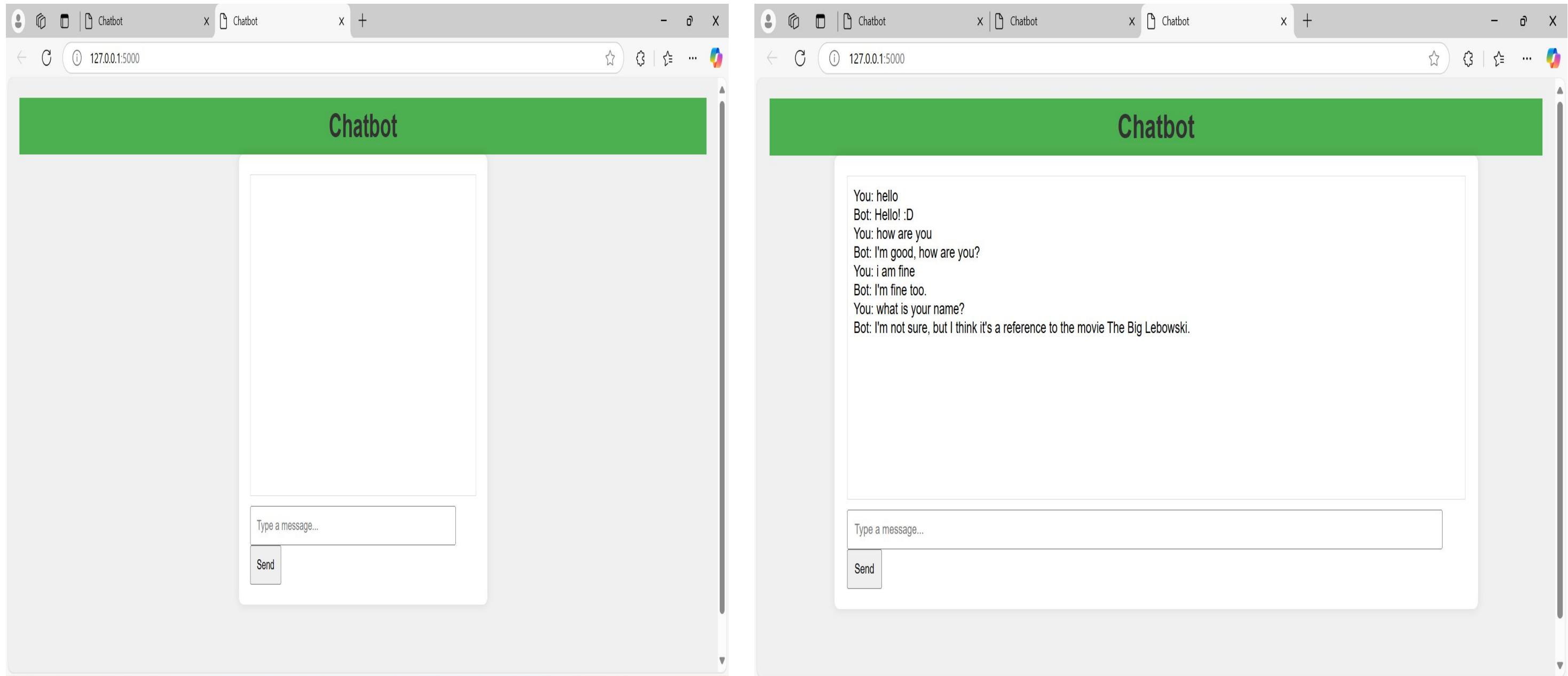
+ Add Files...

Edit files in your workspace

GPT 4o

# Use Case - Building a Chatbot Application with GitHub Copilot

Step 6 : Iterate and make changes and update with copilot and update your code to Github side by side

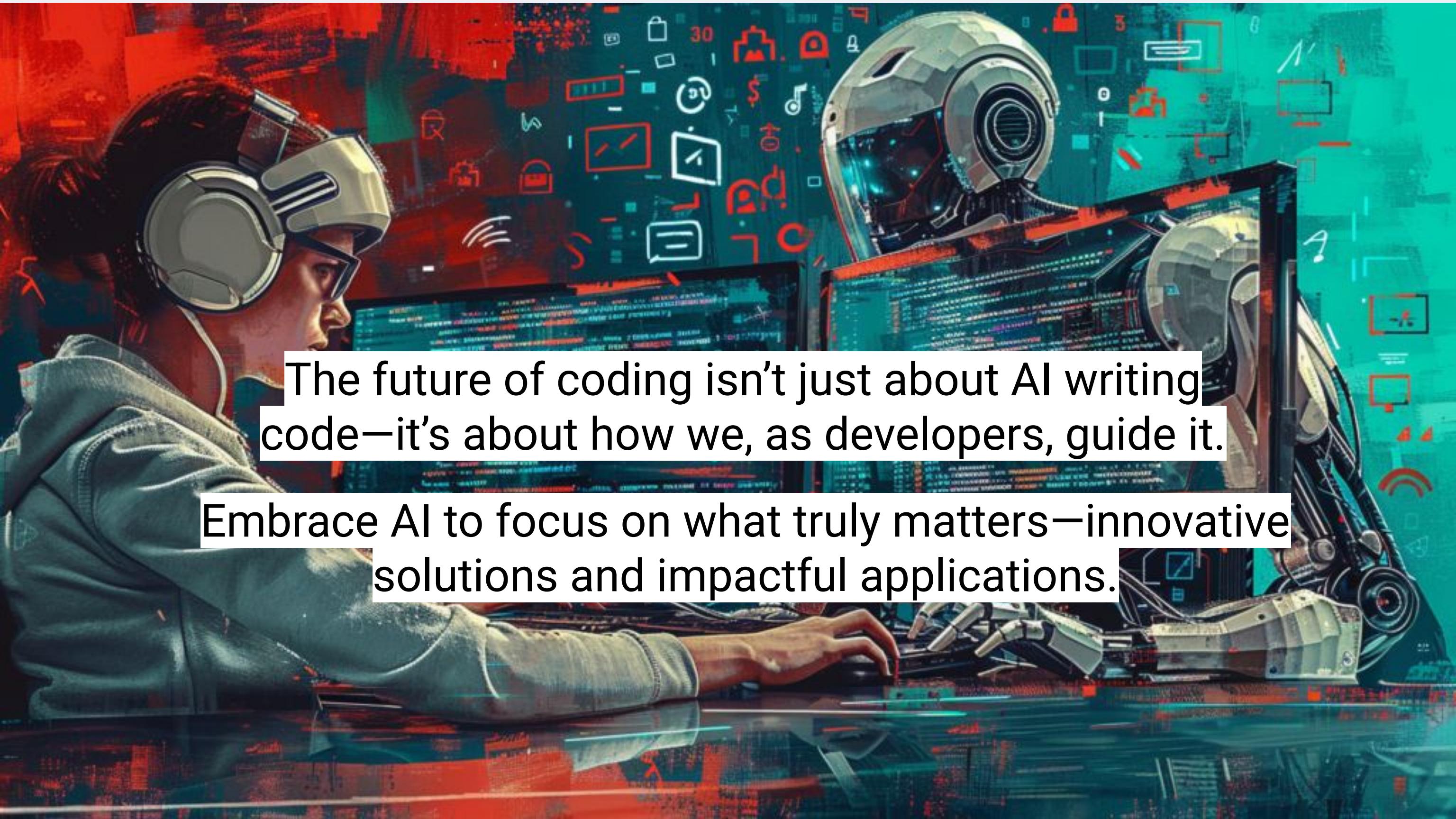


An application which might have taken 1-2 Hrs for development can be developed in half an hour with help of AI

# Conclusion: AI Empowering Developers

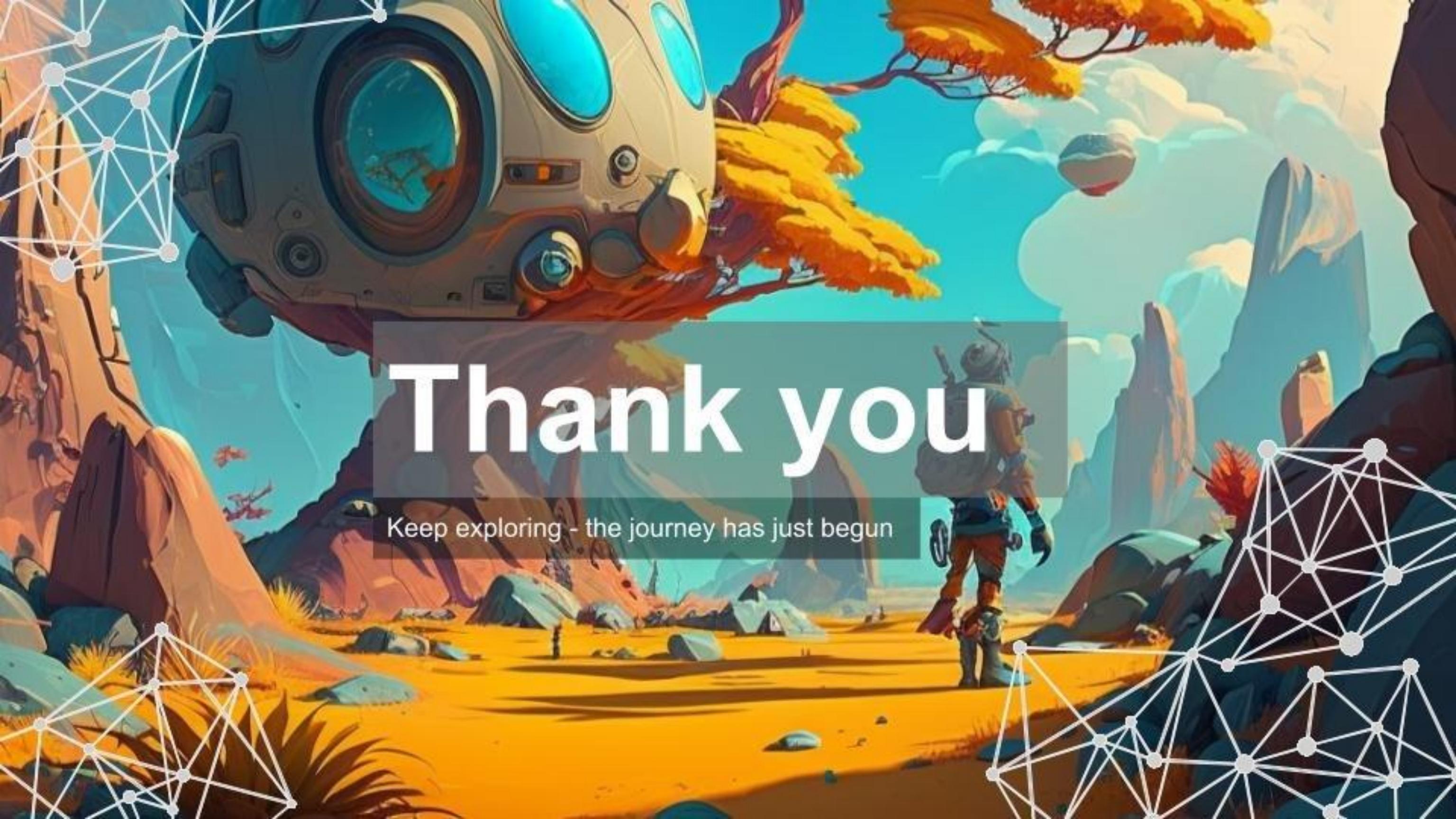
- **Time-Saving:** AI automates repetitive tasks like code generation, debugging, and documentation, reducing development time significantly.
- **Focus on Creativity:** With routine tasks handled by AI, developers can dedicate more time to innovative and complex problem-solving.
- **Enhanced Productivity:** By streamlining workflows, AI helps developers build faster, more reliable, and scalable software solutions.
- **Future of Coding:** AI tools are transforming development into a more creative and efficient process.



A futuristic, high-tech collage. On the left, a person wearing a VR headset is shown in profile, looking at a screen displaying code. In the center, a large, metallic, articulated robotic arm extends from the right side of the frame. The background is a dense, colorful digital space filled with floating icons and symbols related to technology, data, and communication, such as a shopping cart, a lock, a dollar sign, a speech bubble, and a Wi-Fi signal.

The future of coding isn't just about AI writing code—it's about how we, as developers, guide it.

Embrace AI to focus on what truly matters—innovative solutions and impactful applications.



# Thank you

Keep exploring - the journey has just begun