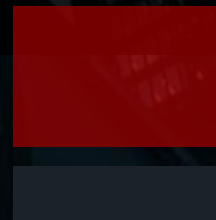




Introduction to Github Copilot

Making your code Intelligent





AGENDA

- What is Github Copilot?
- Features & Capabilities
- Setting up Github Copilot in VS Code Studio
- Basic usage & understanding of AI-suggested code



Evolution of Coding

From manual coding to AI-assisted Development

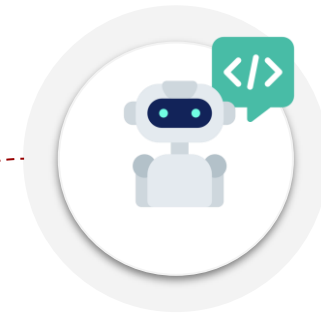
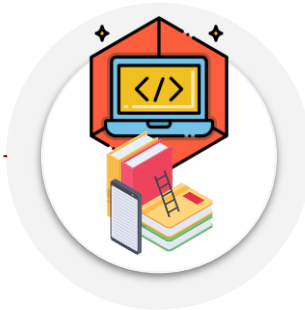


1970s

*Handwritten code and
punch cards*

*Integrated Development Environments
(IDEs) and code libraries*

1990s



2020s

*AI is revolutionizing the
way we write code*

Challenges Developers Face Today

The Coding Pain Points



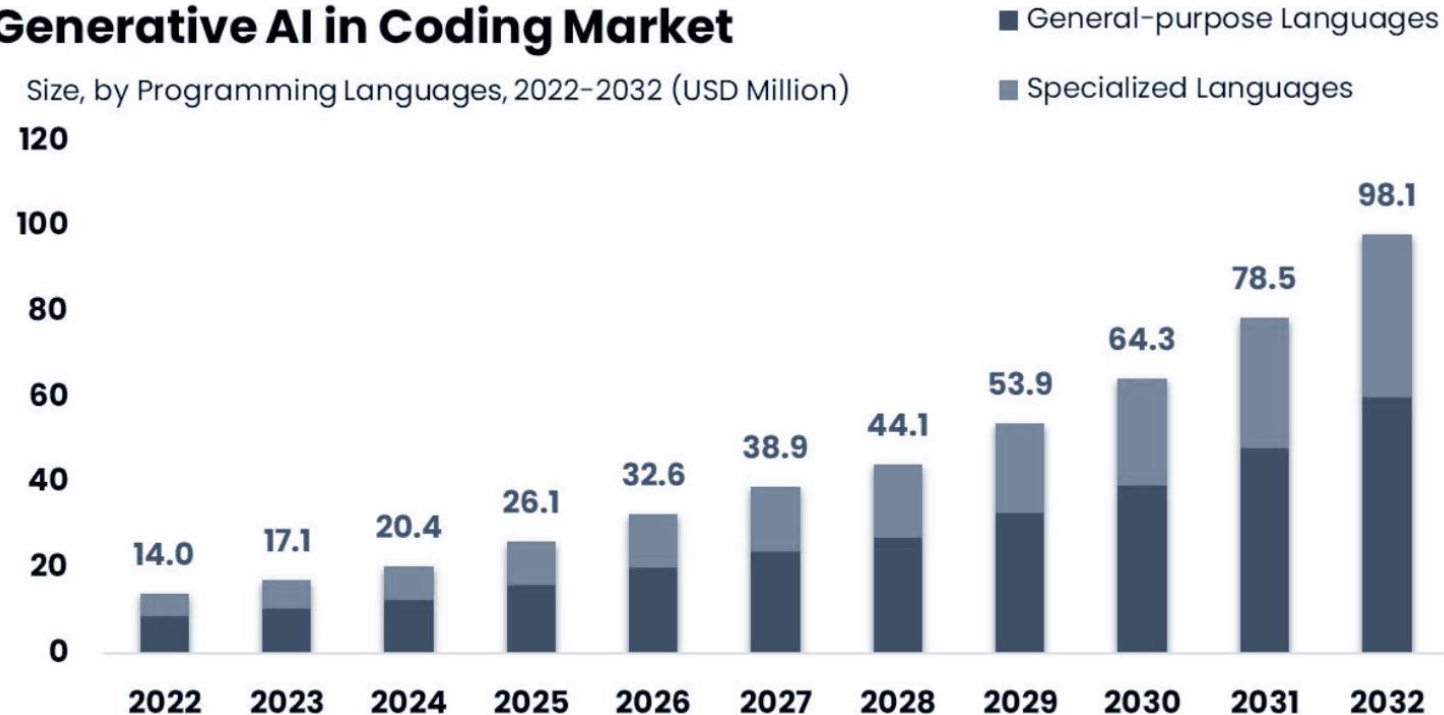
- *Tedious repetition of boilerplate code*
- *Keeping up with multiple programming languages*
- *Difficulty in debugging & understanding existing code*
- *Long development cycles and tight deadlines*

Need for Automation & Vision of AI in Coding

Can AI make developers more efficient?

Generative AI in Coding Market

Size, by Programming Languages, 2022-2032 (USD Million)



Source: [MarketResearch.Biz](https://www.marketresearch.biz)

GenAI in coding Market will
grow at the CAGR of

22.1%

As per BCG,

30-50%

productivity boost can be achieved in software development by integrating automation tools, leading to faster coding, testing, and deployment with fewer manual errors and higher quality results

Introducing Github Copilot

Meet Your AI Coding Partner: GitHub Copilot

- GitHub Copilot is the world's first at-scale AI developer tool that can help you write code faster with less work
- GitHub Copilot draws context from comments and code to suggest individual lines and whole functions instantly
- GitHub Copilot helps developers code faster, focus on solving bigger problems, stay in the flow longer, and feel more fulfilled with their work



Introducing Github Copilot

Meet Your AI Coding Partner: GitHub Copilot



The screenshot shows the GitHub Copilot interface. On the left is a chat window with the GitHub Copilot logo and the text "monalisa Write unit tests for this function". Below this, it shows a code snippet for a unit test. On the right is a code editor with three tabs: "parse_expenses.py", "addresses.rb", and "sentiments.ts". The "parse_expenses.py" tab is active, showing a Python function "def parse_expenses (expenses_string):" that parses a string of expenses into a list of tuples. The function includes comments and example input/output. The code editor also shows a list of expenses data.

```
import unittest
import datetime

class TestParseExpenses(unittest.TestCase):
    def test_parse_expenses_with_valid_input(self):
        expenses_string = '2023-01-02 -34.01 USD\n2023-01-03 2.59 DKK\n2023-01-03 -2.72 EUR'
        expected_output = [(datetime.datetime(2023, 1, 2), -34.01, 'USD'), (datetime.datetime(2023, 1, 3), 2.59, 'DKK'), (datetime.datetime(2023, 1, 3), -2.72, 'EUR')]
        self.assertEqual(parse_expenses(expenses_string), expected_output)

    def test_parse_expenses_with_empty_input(self):
        expenses_string = ''
        expected_output = []
        self.assertEqual(parse_expenses(expenses_string), expected_output)

if __name__ == '__main__':
    unittest.main()
```

The code assumes that the datetime module is imported. The test cases cover different scenarios such as valid input, empty input, input with comments, invalid date format, and invalid value format.

Ask a question or type '/' for commands

```
1 import datetime
2
3 def parse_expenses (expenses_string):
4     """Parse the list of expenses and return the list of triples (date, amount, currency)
5     Ignore lines starting with #.
6     Parse the date using datetime.
7     Example expenses_string:
8     2023-01-02 -34.01 USD
9     2023-01-03 2.59 DKK
10    2023-01-03 -2.72 EUR
11    """
12    expenses = []
13
14    for line in expenses_string.splitlines():
15        if line.startswith("#"):
16            continue
17        date, value, currency = line.split(" ")
18        expenses.append((datetime.datetime.strptime (date, "%Y-%m-%d"),
19                        float (value),
20                        currency))
21    return expenses
22
23 expenses_data = '''2023-01-02 -34.01 USD
24 2023-01-03 2.59 DKK
25 2023-01-03 -2.72 EUR'''
26
27
28
29
30
31
32
33
34
```



Introducing Github Copilot

Meet Your AI Coding Partner: GitHub Copilot



50,000+

*Businesses have
adopted GitHub
Copilot*

1 in 3

*Fortune 500
companies are using
Github copilot*

55%

*of developers
preferred Github
Copilot over others*

Duolingo empowers its engineers to be force multipliers for expertise with GitHub Copilot



Problem

Inconsistent standards and workflows limited developer mobility and efficiency, limiting Duolingo's ability to expand its content and deliver on its core mission



Solution

GitHub Copilot, Codespaces, and custom API integrations enforce code consistency, accelerate developer speed, and remove the barriers to using engineering as a force multiplier for expertise



Products



GitHub Enterprise



GitHub
Codespaces



GitHub Copilot

25%

increase in developer speed
with GitHub Copilot

67%

decrease in median code review
turnaround time

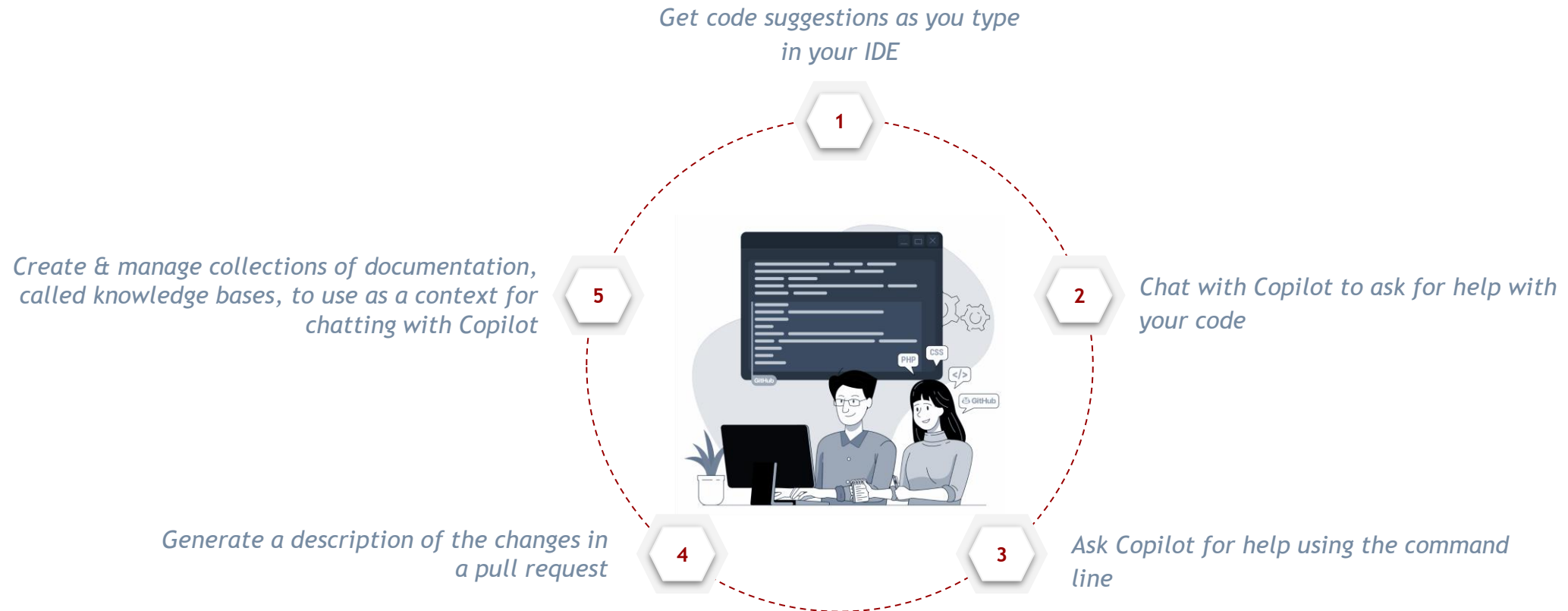
70%

increase in pull requests



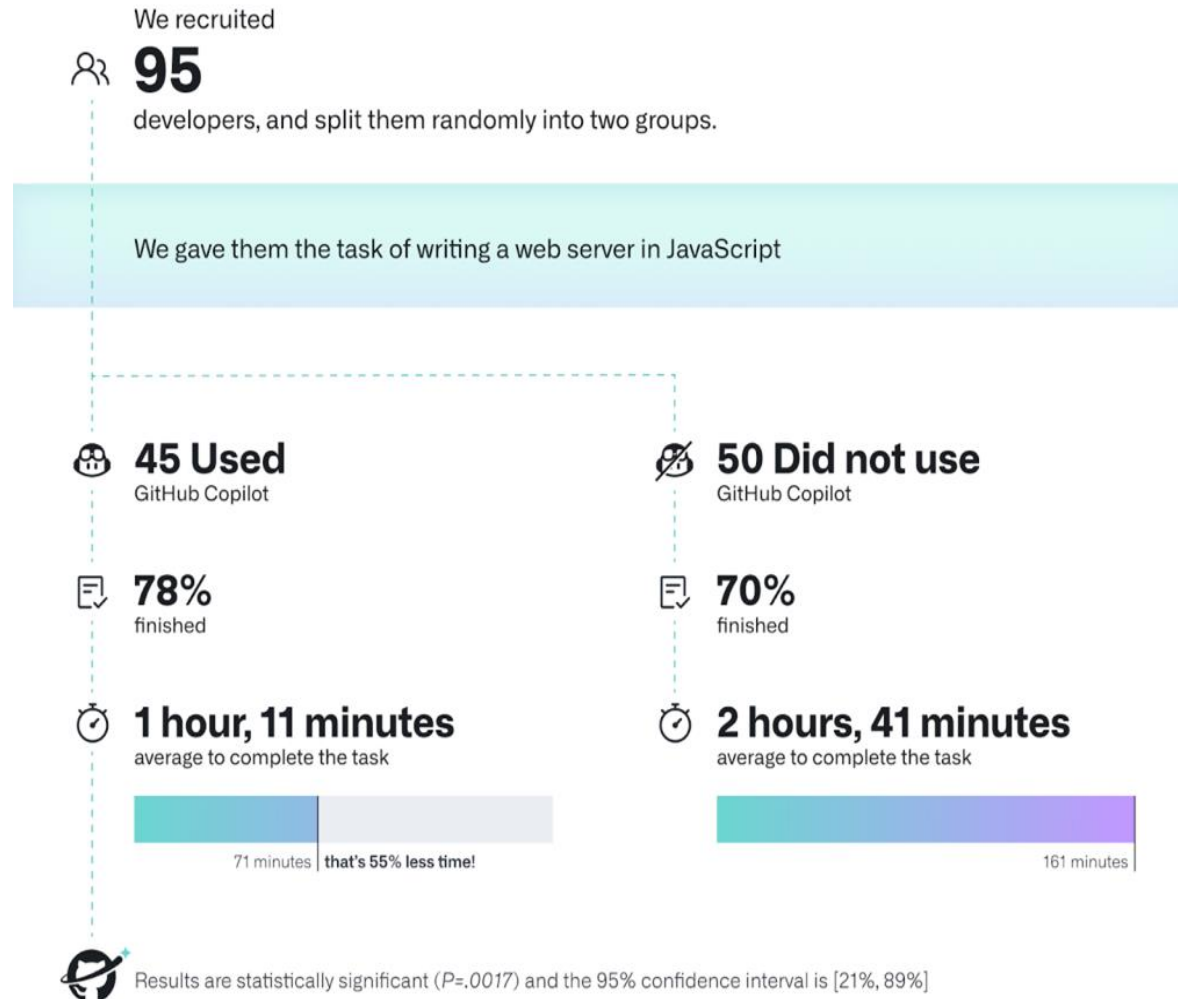
Features of Github Copilot

The AI coding assistant changing coding paradigm



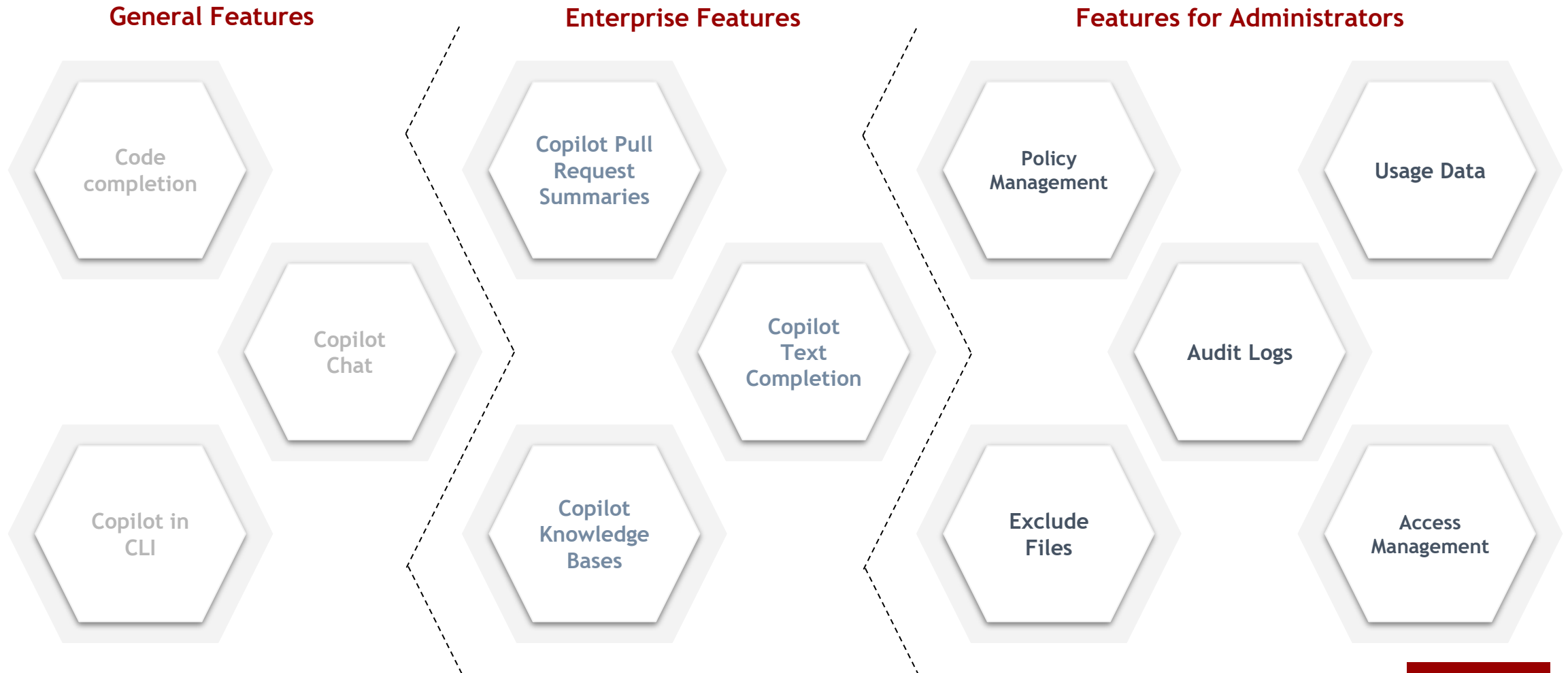
Features of Github Copilot

This AI coding assistant is changing the coding paradigm



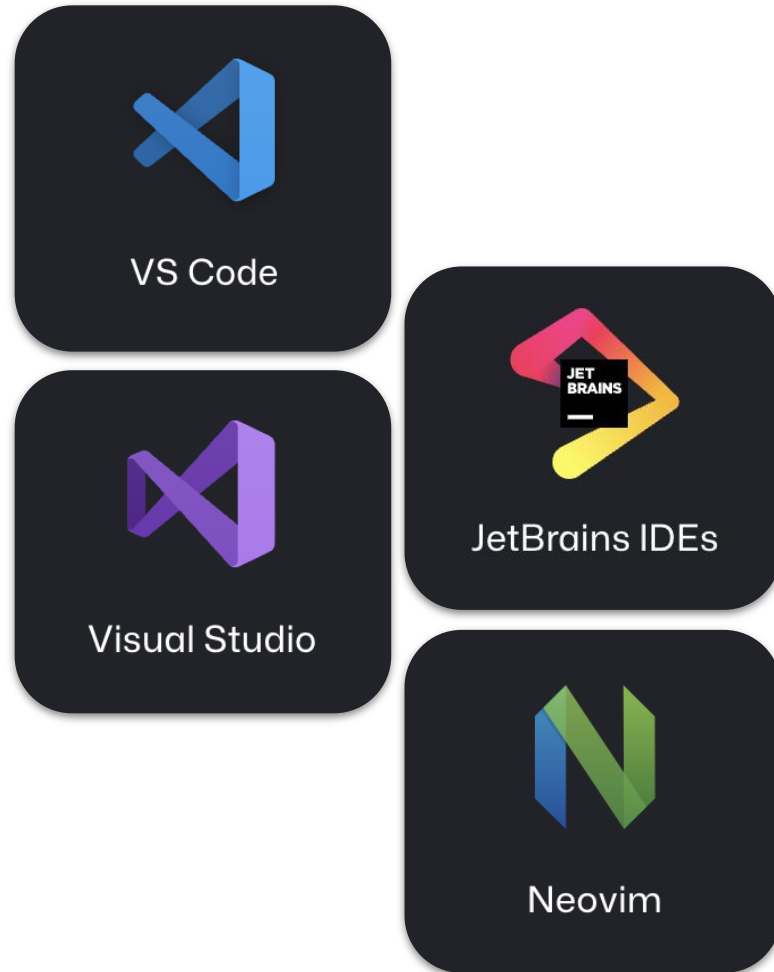
Features of Github Copilot

The AI coding assistant changing coding paradigm



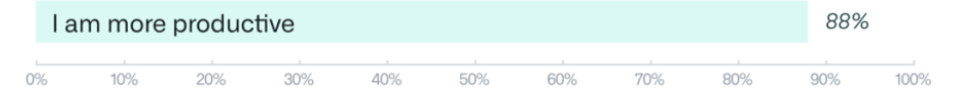
Keep flying with your favourite editor

Ask for assistance right in your IDE

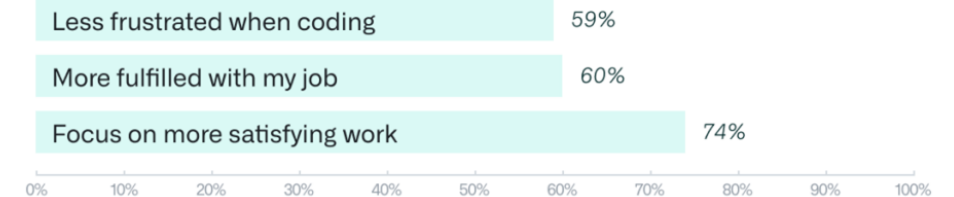


When using GitHub Copilot...

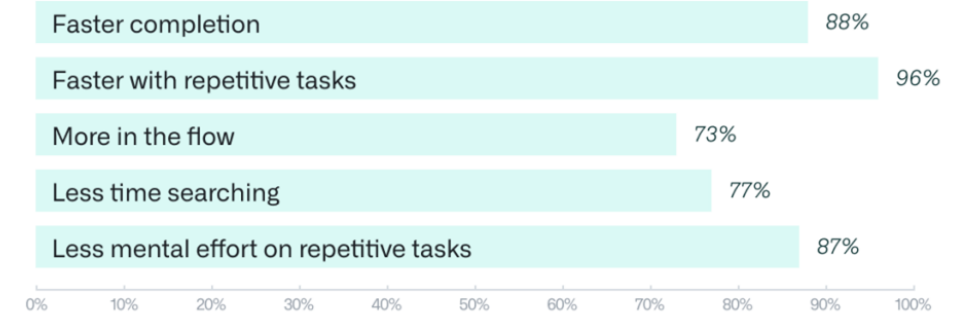
Perceived Productivity



Satisfaction and Well-being*



Efficiency and Flow*



Source:
https://visualstudiomagazine.com/articles/2022/09/13/~/_media/ECG/visualstudiomagazine/Images/2022/09/Copilot1.ashx

How does Github Copilot works?

Know how magic is happening!!



Subscription Plans for Github Copilot

Explore your Options!!

Copilot Individual

For individual developers, freelancers, students, and educators that want to code faster and happier

\$10 USD

Code completion in IDEs, Copilot Chat in IDEs, Copilot Chat in GitHub Mobile, Copilot in the CLI, Block suggestions matching public code



Copilot Business

For organizations ready to improve engineering velocity, code quality, and developer experience

\$19 USD

Copilot Individual + Exclude specific files from Copilot, Organization-wide policy management, Audit logs, Increased GitHub Models rate limits



Copilot Enterprise

For companies looking to customize GitHub Copilot to their organization and infuse AI across the developer workflow

\$39 USD

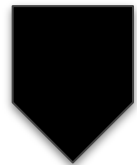
Copilot Business + Copilot Chat in GitHub.com, Copilot Chat skills in IDEs, Copilot pull request summaries, Copilot knowledge bases, Fine tuning a custom large language model



Strengths & Weaknesses

Understand your coding partner in depth!

Some of the things Copilot does best are



Writing tests and repetitive code



Debugging and correcting syntax



Explaining and commenting code



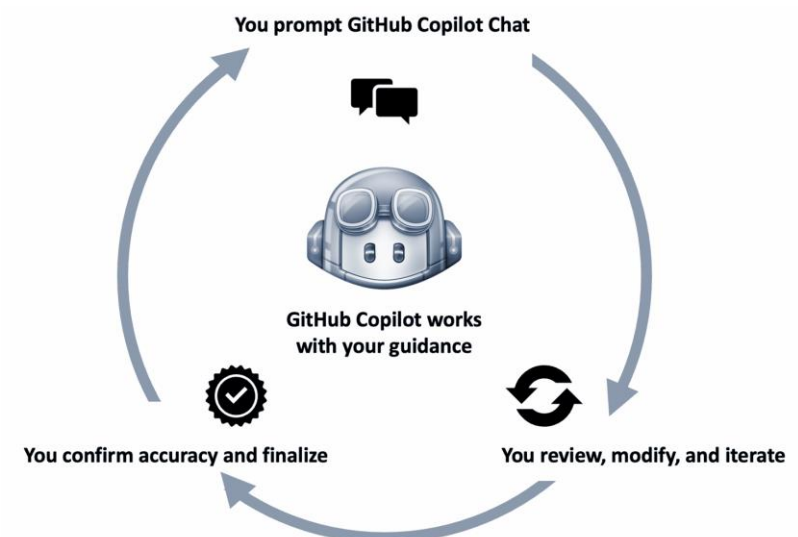
Generating regular expressions

Copilot is not designed to

*Respond to prompts unrelated to coding
and technology*



*Replace your expertise & skills. Remember that you are
in charge, & Copilot is a powerful tool at your service*



Set Up

Preparing your armor!!

1

Setup for Organization

2

Setup for Self

Set Up

Preparing your armor!!

Setup for Organization



Go to the [GitHub Copilot Business sign up page](#)



Enter organization details for which you want to purchase GitHub Copilot and click **Continue**



Confirm your **payment** details, then click **Save**



In the "**Public code suggestions**" dropdown, select **Allow** or **Block** to allow or block suggestions that match public code, and click **Save and continue**. You can change these preferences later by returning to your GitHub Copilot settings



Either grant access to GitHub Copilot for all current and future users in your organization, or for specific users in your organization



To finish setting up your GitHub Copilot Business subscription, click **Save and finish**. Organization members will receive an email with instructions on how to start using GitHub Copilot

Set Up

Preparing your armor!!

Setup for Self



Go to the [GitHub Copilot Individual sign up page](#)



Add your personal details and click **Continue**



Confirm your **payment** details, then click **Save**



In the "**Public code suggestions**" dropdown, select **Allow** or **Block** to allow or block suggestions that match public code, and click **Save and continue**. You can change these preferences later by returning to your GitHub Copilot setting



To finish setting up your GitHub Copilot Individual subscription, click **Save and finish**

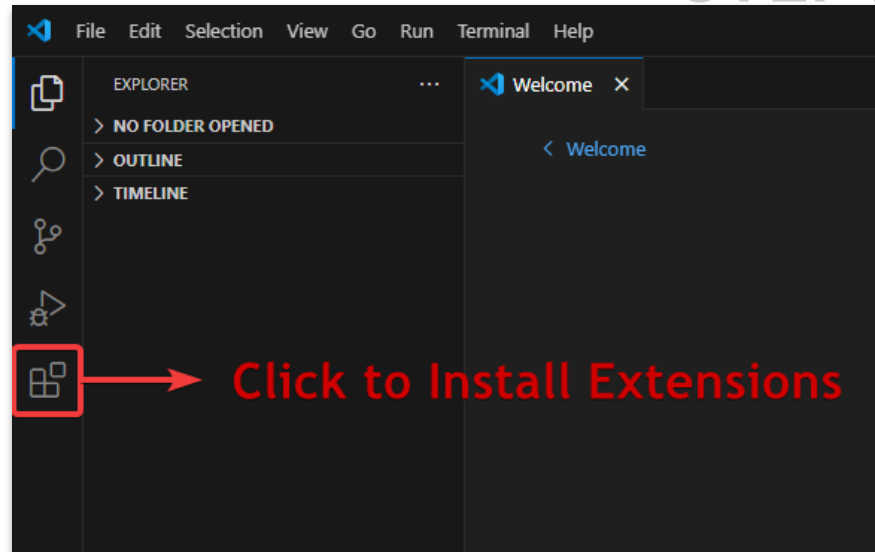


If you are a **verified student, teacher, or maintainer of a popular open source project**, GitHub Copilot is free to use. Visit [Getting free access to Copilot as a student, teacher, or maintainer](#)

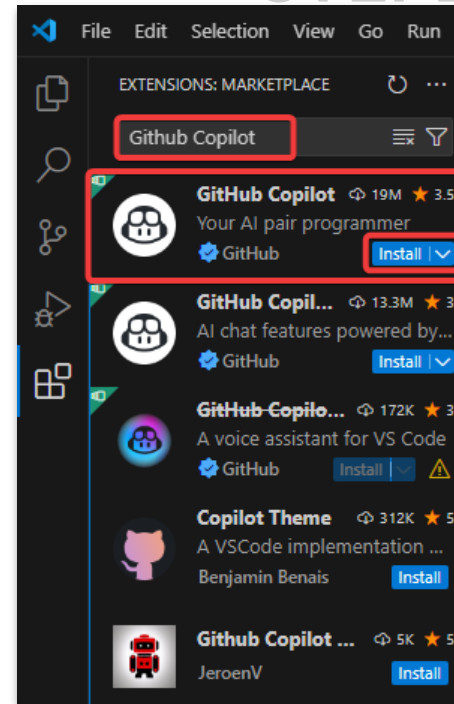
Installing Github Copilot on VS Code Studio

Preparing your armor!!

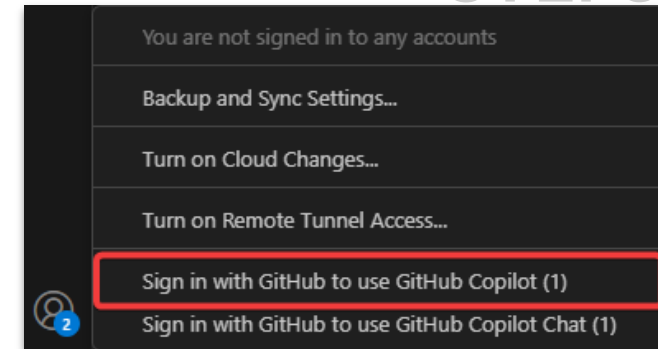
STEP 1



STEP 2

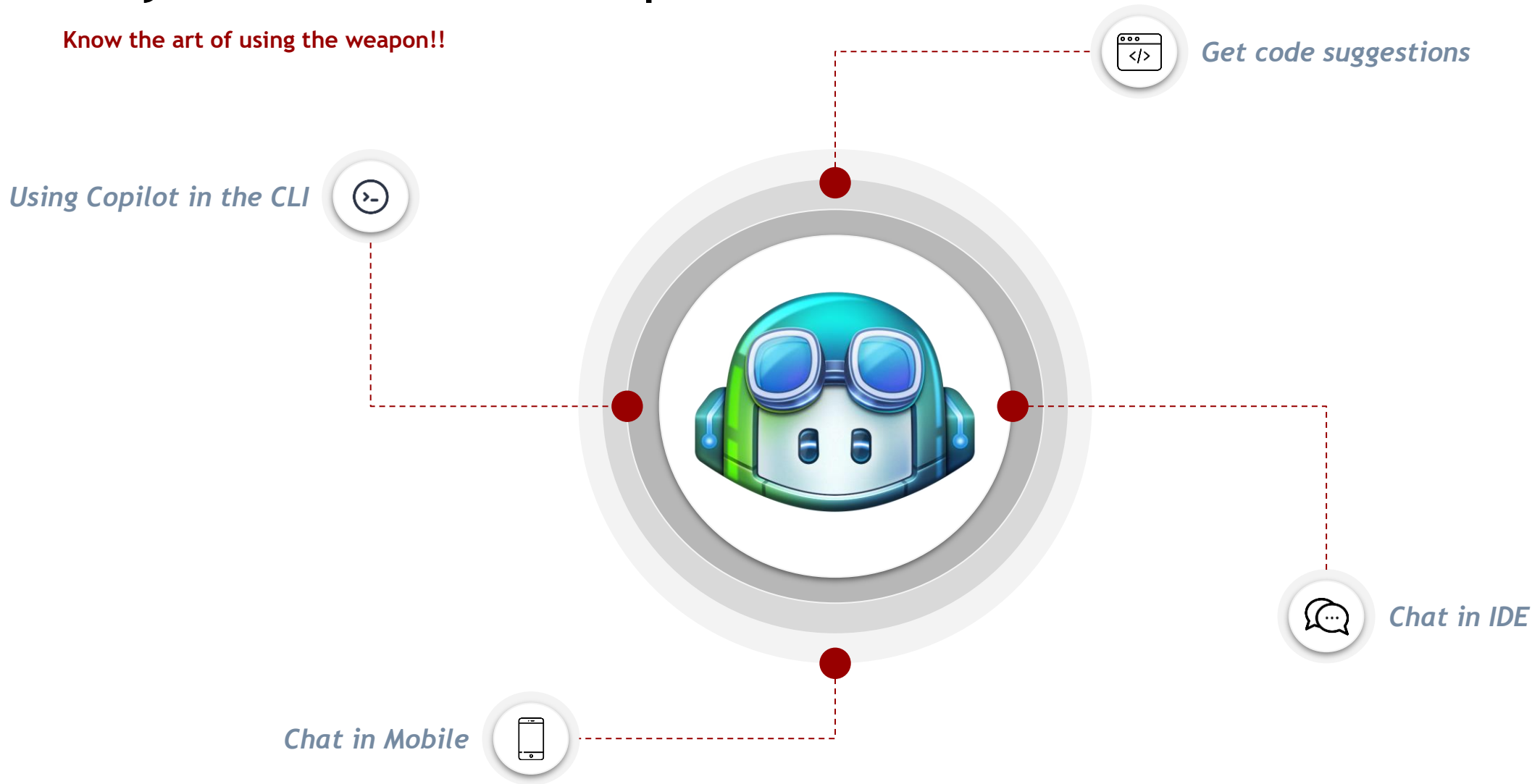


STEP 3



Ways to use Github Copilot

Know the art of using the weapon!!



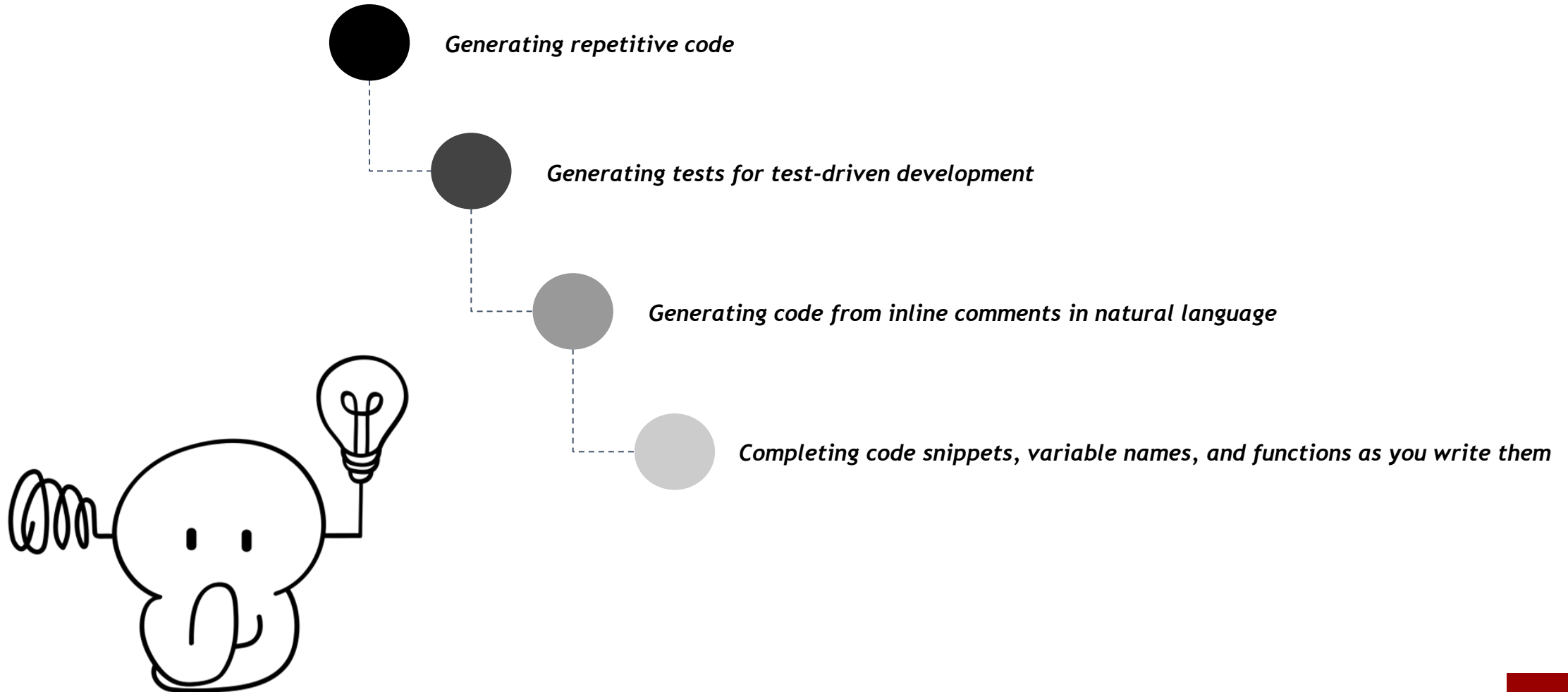
Get Code Suggestion

Use GitHub Copilot to get code suggestions in your editor

- **Real-time Suggestions:** GitHub Copilot provides code suggestions as you type in your IDE
- **Context-Aware:** The suggestions are based on the context of the code you're writing
- **AI-Powered:** The tool uses advanced AI models trained on vast amounts of code to predict and suggest relevant snippets
- **Code Completion:** It offers both simple code completions (e.g., finishing a line of code) & complex suggestions (e.g., entire functions or code blocks)
- **Multi-Language Support:** GitHub Copilot supports a wide range of programming languages and frameworks
- **Improves Efficiency:** It helps developers write code faster by reducing manual coding effort
- **Learning from Patterns:** The tool learns from common coding patterns and your personal style to refine suggestions

Get Code Suggestion

When does it work best?



Get Code Suggestion

Use GitHub Copilot to get code suggestions in your editor

>_

```
// write a function to  
// find all images without alternate text  
// and give them a red border
```

|..



GitHub Copilot will automatically suggest the rest of the function

Press **TAB** to accept the suggestion



>_

```
// write a function to  
// find all images without alternate text  
// and give them a red border
```

```
function findMyImage(begin, end) { ...
```

Get Code Suggestion

Show alternative suggestions for your code

Press **TAB** to accept the suggestion



```
>_  
  
import pandas as pd  
data = ...
```

```
>_  
  
import pandas as pd  
data = pd.read_csv(...)
```



Use the following keyboard shortcuts
for viewing suggestions

OS	See next suggestion	See previous suggestion
macOS	Option (⌘) +]	Option (⌘) + [
Windows or Linux	Alt+]	Alt+[

Get Code Suggestion

Didn't like the initial suggestions? You can even view multiple Suggestions as well

```
>_  
  
import pandas as pd  
|..
```



Press **CONTROL + ENTER** to see multiple suggestions in new tab

GitHub Copilot Suggestions

8 Suggestions

Suggestion 1

```
import numpy
```

Accept suggestion 1

Suggestion 2

```
import numpy as np  
import matplotlib
```

Accept suggestion 2

Suggestion 3

```
import numpy as np  
  
df = pd.DataFrame(np.random.randn(6, 4), columns=  
list('AB
```

Accept suggestion 3

Get Code Suggestion

Show alternative suggestions for your code

>_

```
import pandas as pd
```

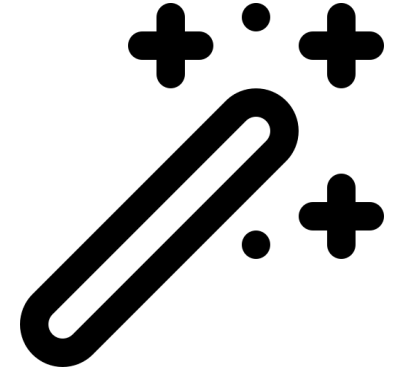
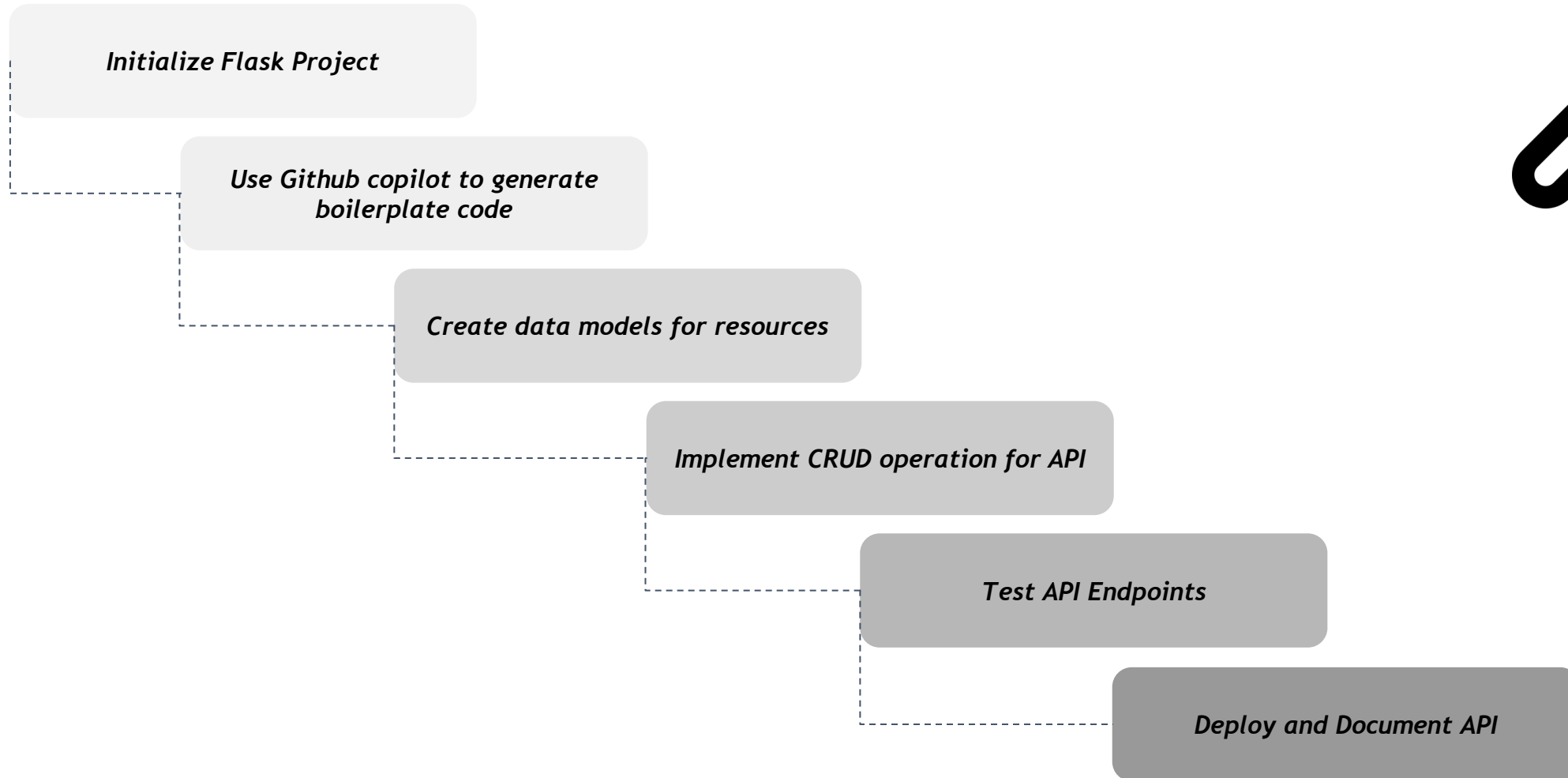
```
data = pd.read_csv(data.csv)
```



Press **CONTROL + →** to accept
the words out the complete suggestion

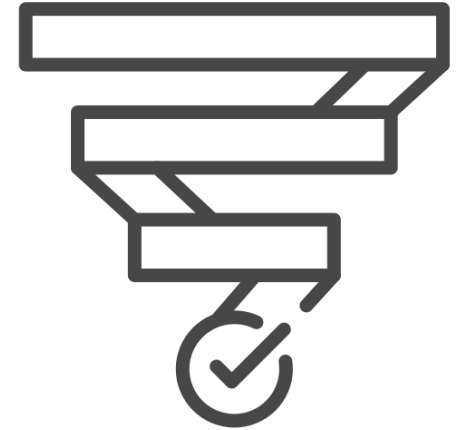
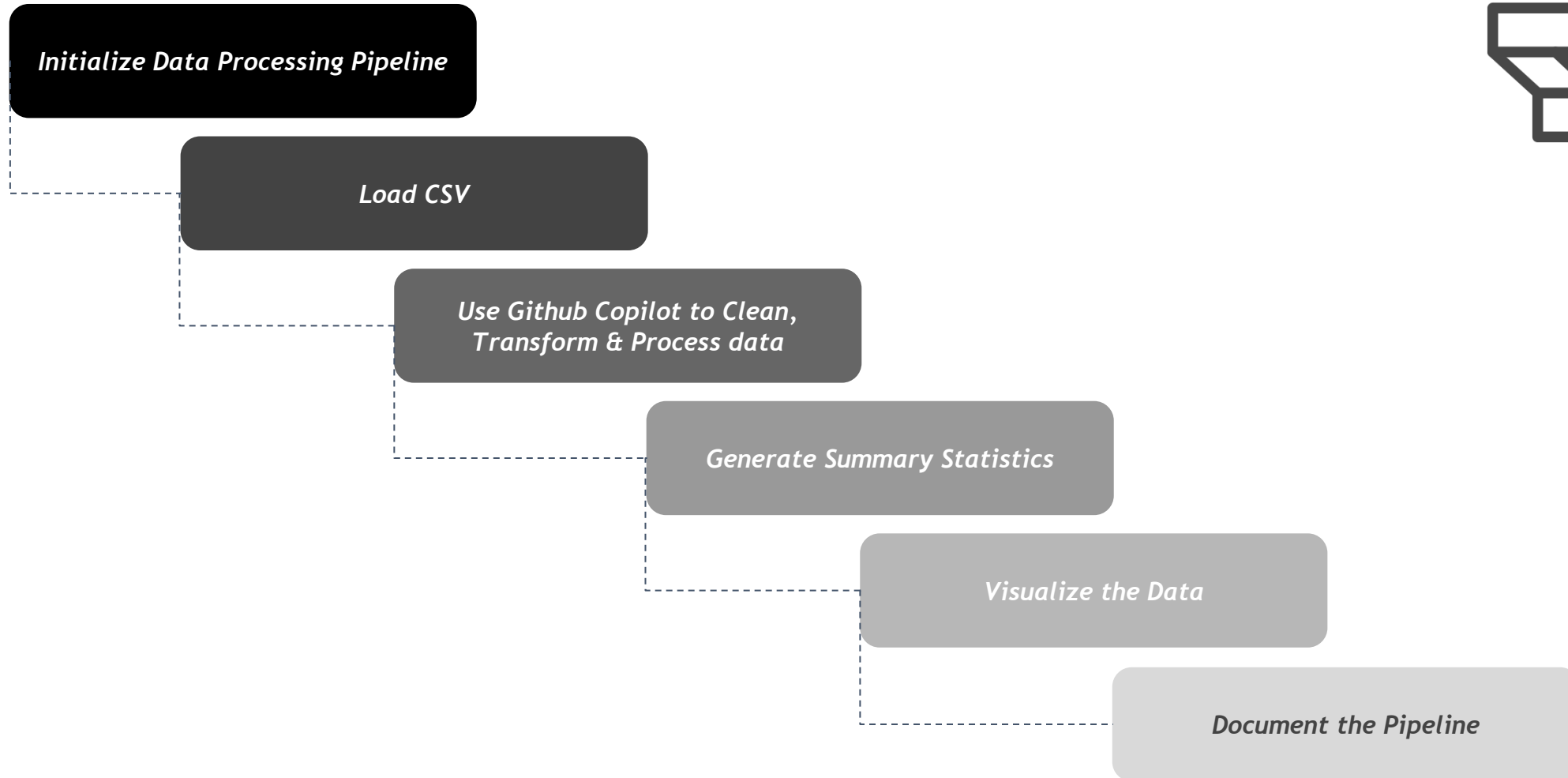
Common Use Cases of “Get Code Suggestion”

Building a Full CRUD API using Flask and Github Copilot



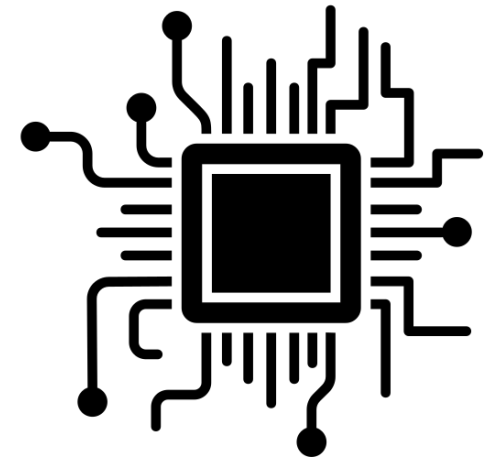
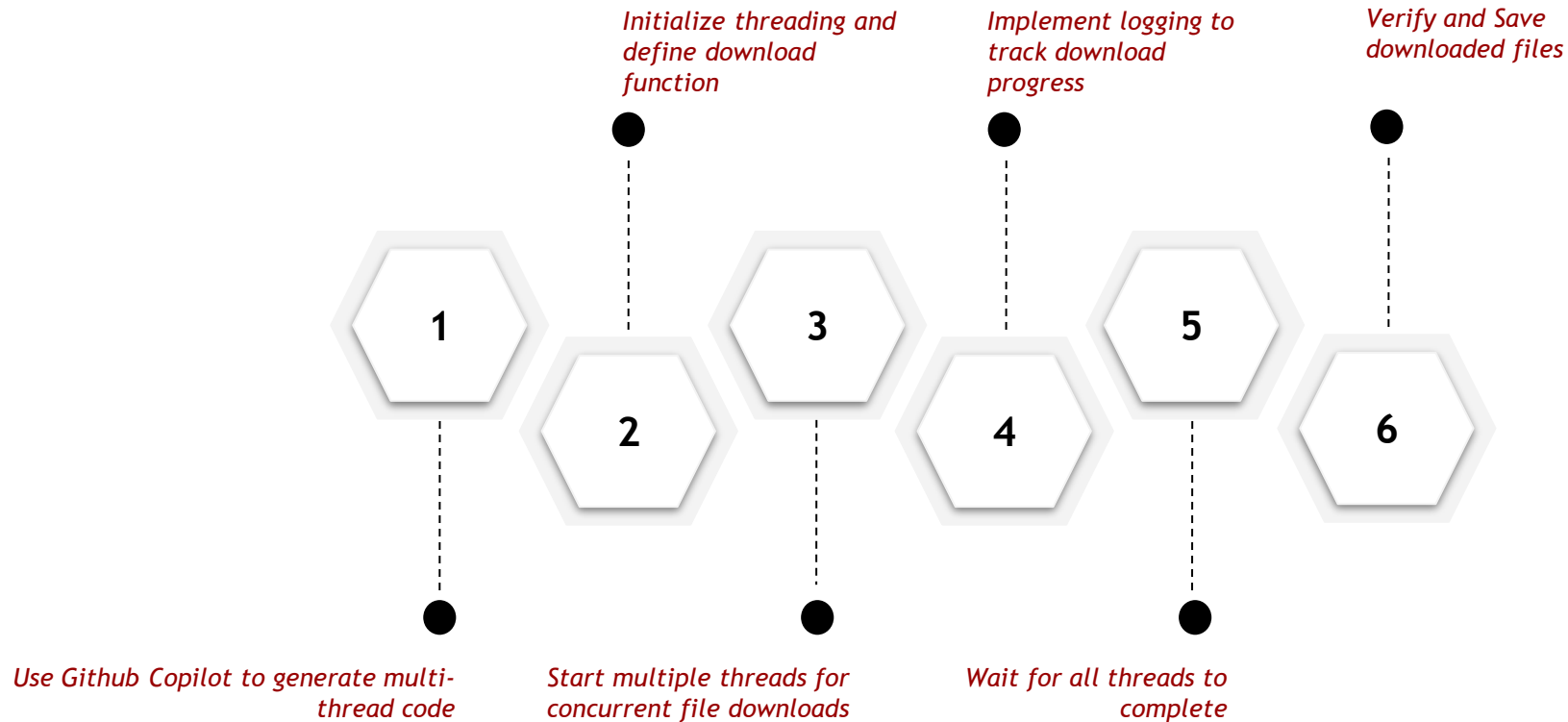
Common Use Cases of “Get Code Suggestion”

Making Data Processing Pipelines with Pandas and Github Copilot



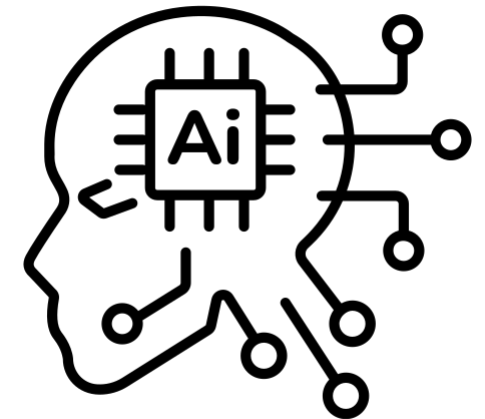
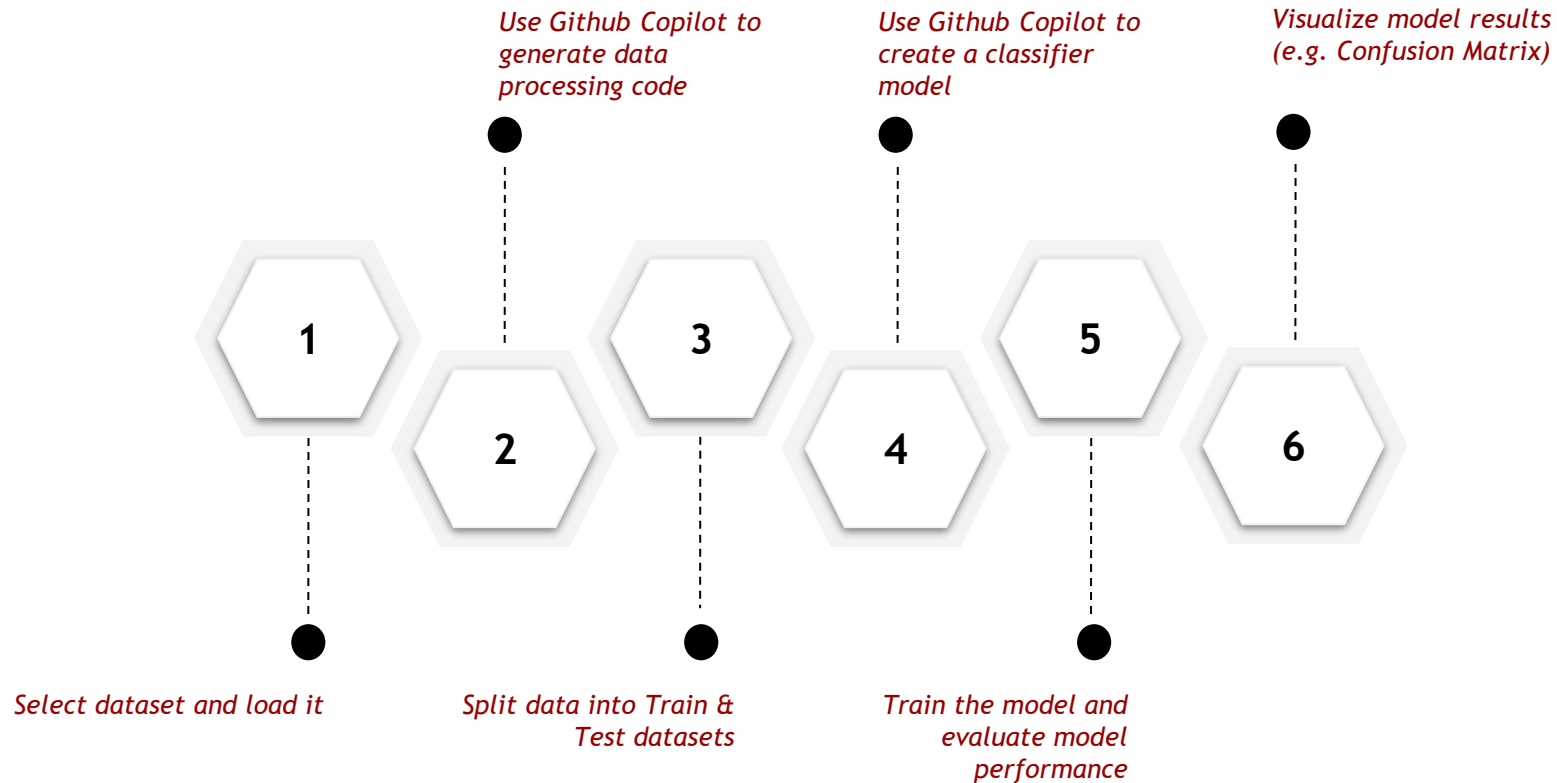
Common Use Cases of “Get Code Suggestion”

Develop Multi-Threaded Python Application



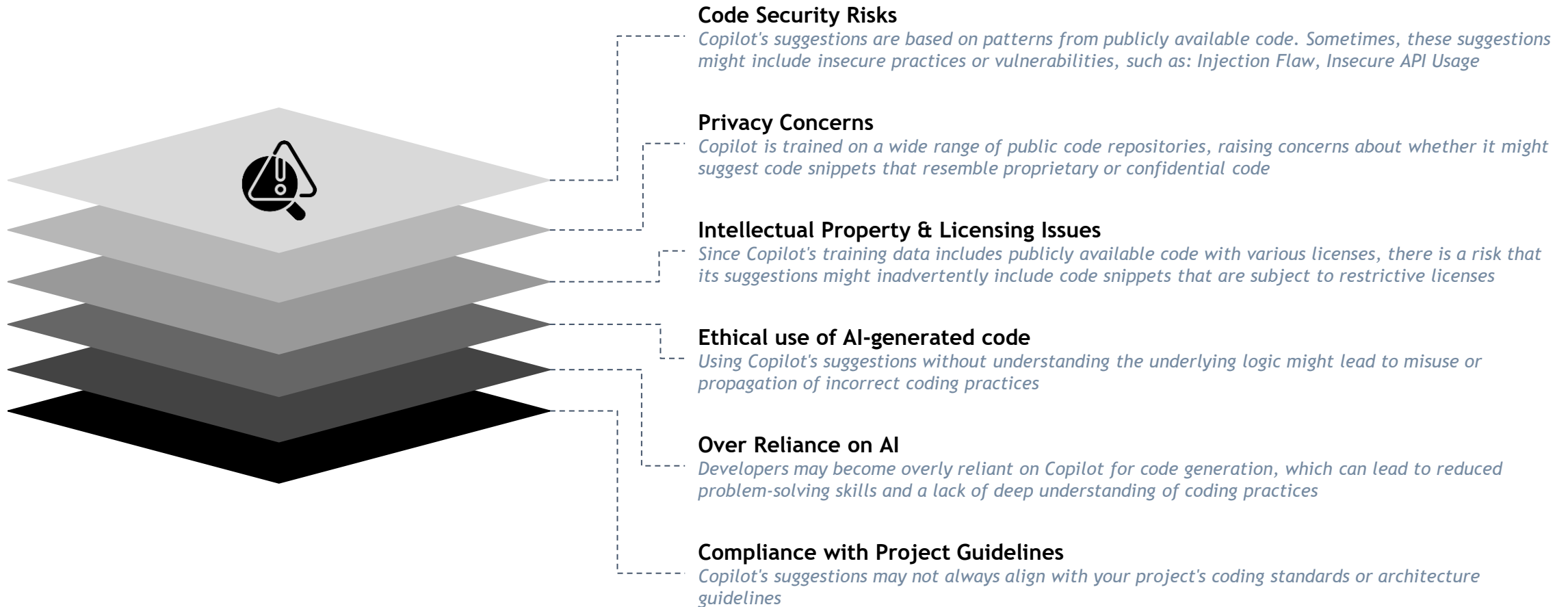
Common Use Cases of “Get Code Suggestion”

Implement Machine Learning Models using Scikit-learn



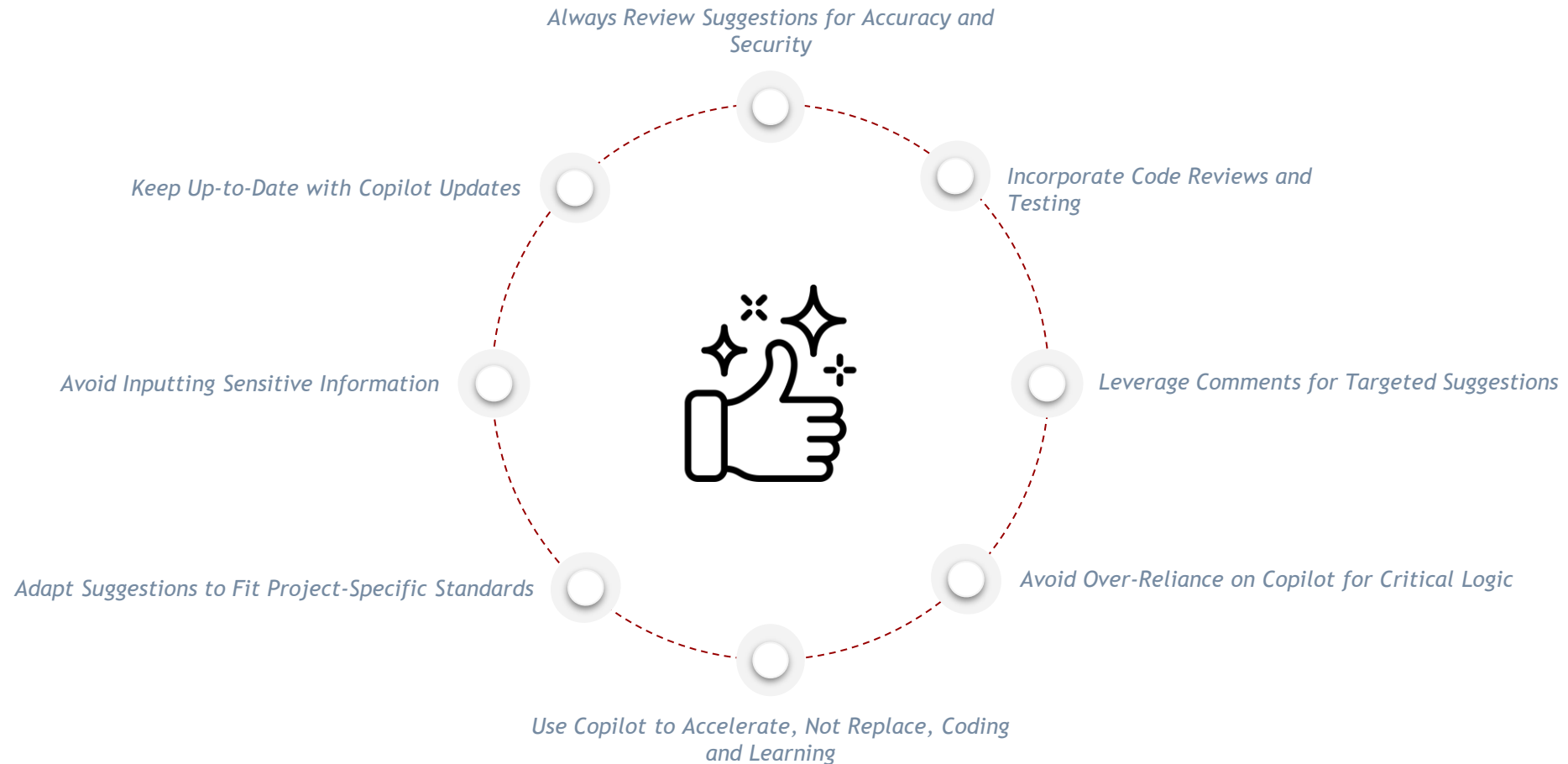
Get Code Suggestion

Security & Ethical Consideration



Best Practices for “Get Code Suggestion”

Optimizing GitHub Copilot Usage: Best Practices for Effective and Secure Code Generation

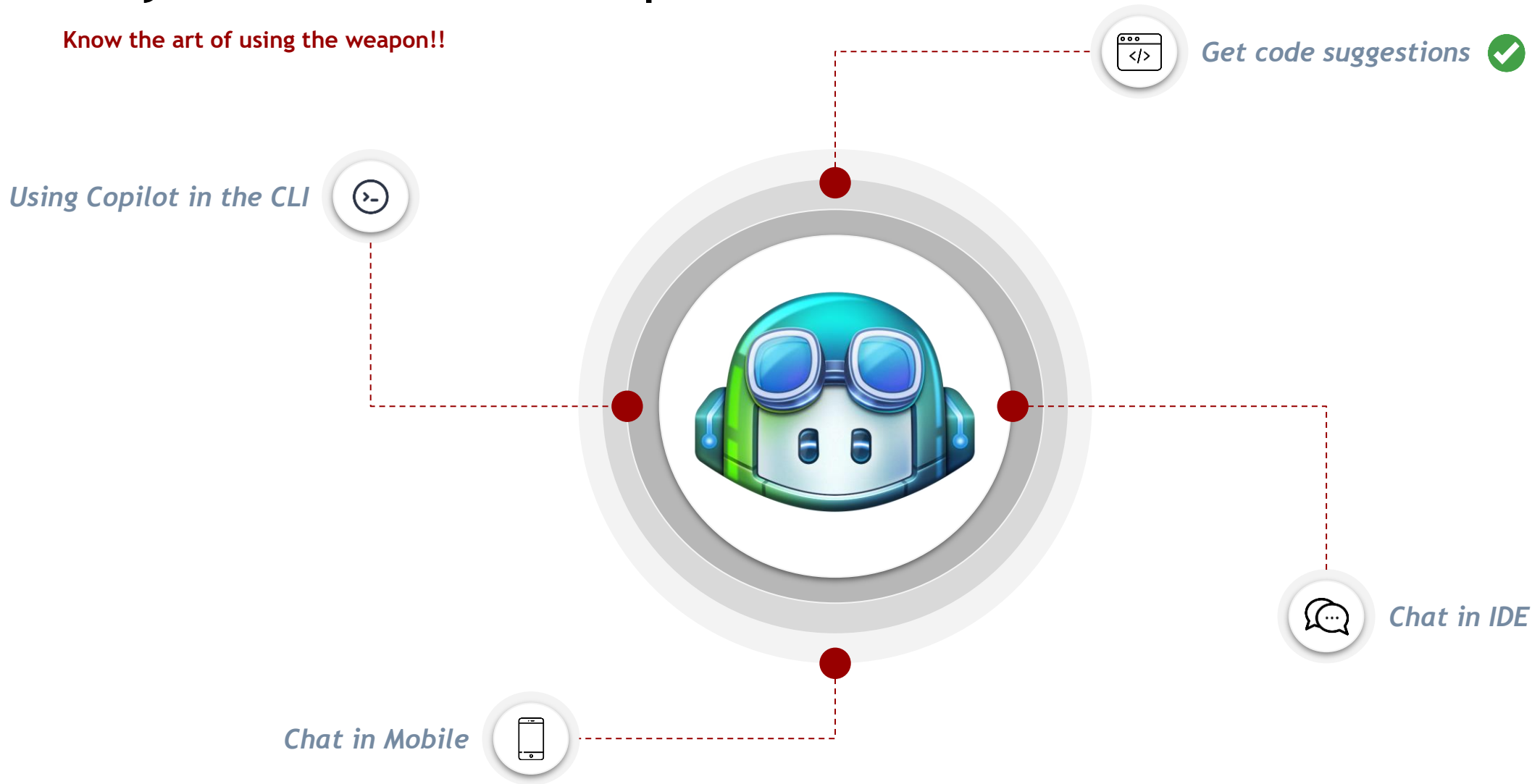


D E M O



Ways to use Github Copilot

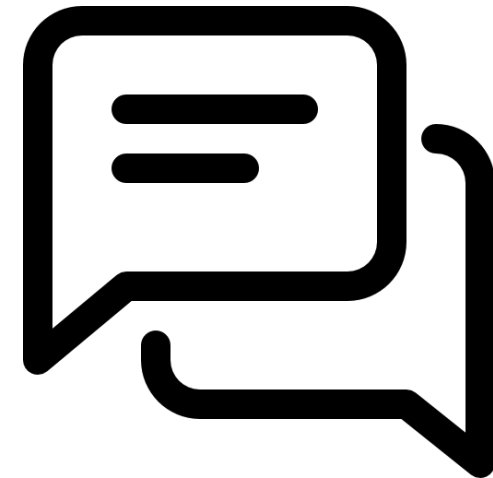
Know the art of using the weapon!!



Chat in IDE

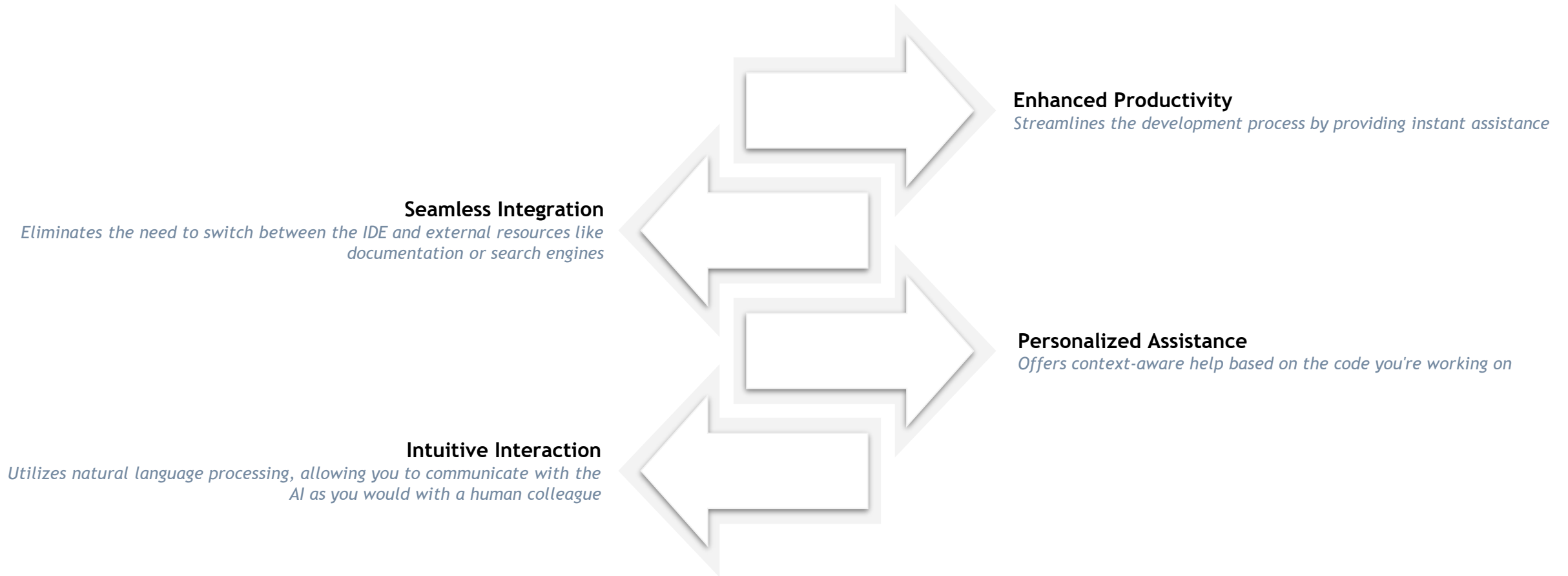
Use Copilot Chat in your editor to give code suggestions, explain code, generate unit tests, and suggest code fixes

- "Chat in IDE" refers to an interactive chat feature integrated within your Integrated Development Environment (IDE)
- Specifically, with tools like GitHub Copilot, it provides an AI-powered conversational assistant directly inside your coding environment
- This feature allows developers to engage in a dialogue with the AI to receive code suggestions, explanations, and assistance without leaving the IDE



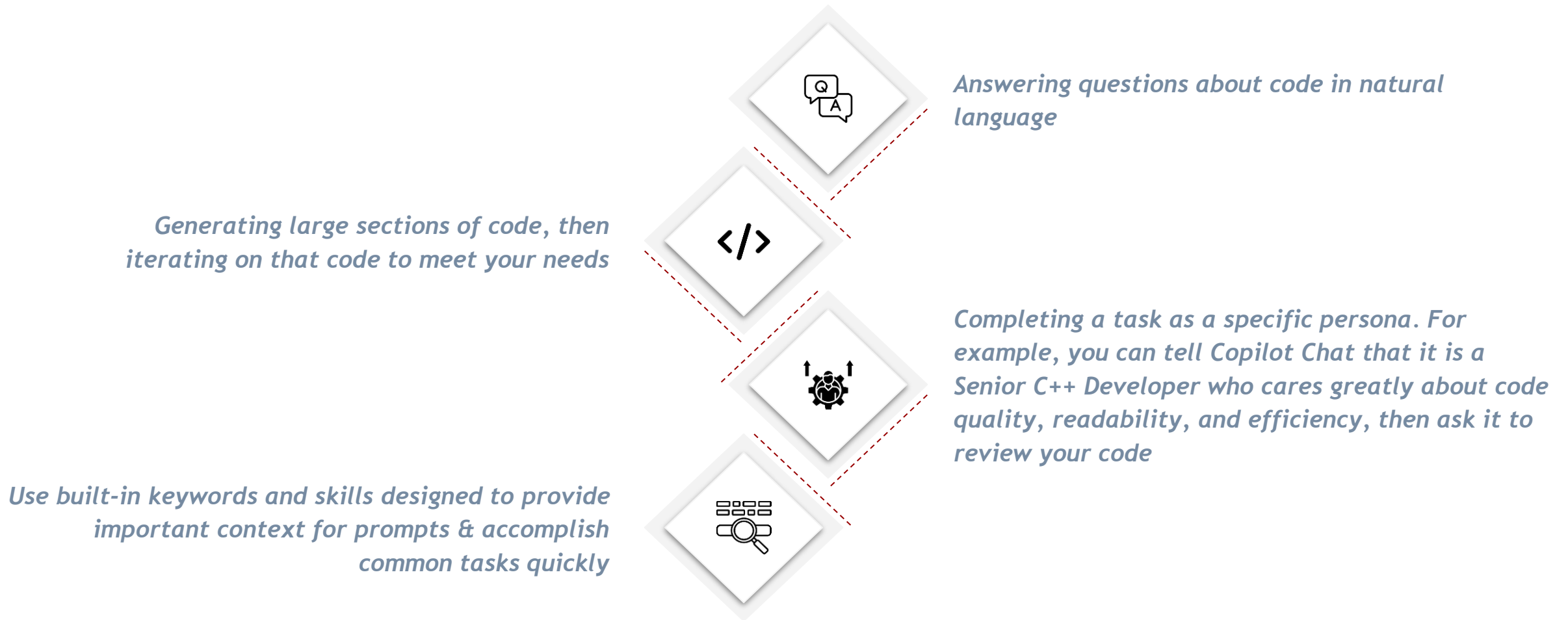
Why use “Chat in IDE”?

Know the benefits



“Chat in IDE” is best suited for

Know the benefits

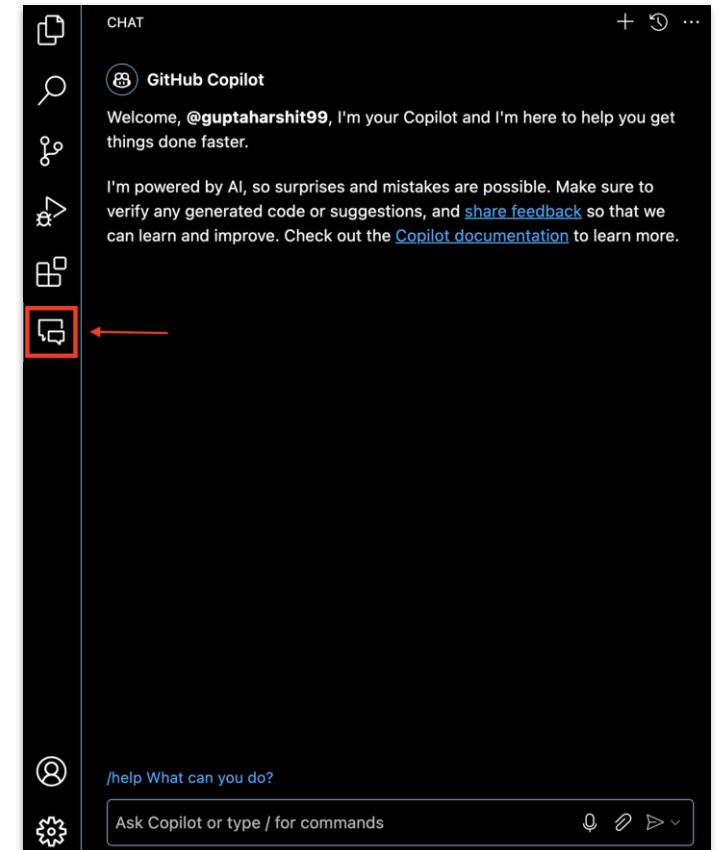


Using “Chat in IDE”

Submit Prompts

- Open the chat view by clicking the chat icon in the activity bar or by entering **Control+Command+i** (Mac) / **Ctrl+Alt+i** (Windows/Linux)
- Enter a prompt in the prompt box, or click one of the suggested prompts
- Evaluate Copilot's response, and make a follow up request if needed

The response may contain text, code blocks, buttons, images, URIs, and file trees. The response often includes interactive elements. For example, the response may include a menu to insert a code block, or a button to invoke a Visual Studio Code command



Using keywords in your prompt

Chat Participants

Use chat participants to scope your prompt to a specific domain. To use a chat participant, type @ in the chat prompt box, followed by a chat participant name

1

@workspace

Has context about the code in your workspace. Use @workspace when you want Copilot to consider the structure of your project, how different parts of your code interact, or design patterns in your project

2

@vscode

Has context about Visual Studio Code commands and features. Use it when you want help with Visual Studio Code

3

@terminal

Has context about the Visual Studio Code terminal shell and its contents. Use it when you want help creating or debugging terminal commands

Using keywords in your prompt

Chat Participants - Sample Questions

@terminal *how to update an npm package*

@workspace *how are notification scheduled*

@vscode *tell me how to debug a node.js app*

@terminal *find the largest file in the src directory*

@vscode *how do I change my Visual Studio Code Color*

@workspace *add form validation, similar to newsletter page*

Using keywords in your prompt

Slash Commands

Use slash commands to avoid writing complex prompts for common scenarios. To use a slash command, type / in the chat prompt box, followed by a command

1

/tests

Generate unit tests for the selected code

2

/fix

Propose a fix in problems in the selected code

3

/explain

Explain the selected Code

4

/start

Start a new chat



*To see all available slash commands, type
"/" in the chat prompt box*

Using keywords in your prompt

Slash Commands - Sample Questions

/new *python django web application*

/test *test using JTest Framework*

/newnotebook *retrieve the titanic dataset and use seaborn to plot the data*

/fix *optimize this function for better experience*

/explain *how does this function calculate fibonacci sequence?*

Using keywords in your prompt

Chat Variables

*Use chat variables to include specific context in your prompt. To use a chat variable, type **#** in the chat prompt box, followed by a chat variable*

1

#file

Include a specific file as context in the chat

2

#git

Include information about the current Git repository

3

#terminalLastCommand

Include the last run command in the active Visual Studio Code terminal



*To see all available chat variables, type **#** in the chat prompt box*

Using keywords in your prompt

Chat Participants - Sample Questions

`/explain #file: src/utils/helpers.js`

`/explain #git: status`

`/fix #git: merge -conflict in file "config/settings.yml"`

`/explain #git: diff HEAD~1`

`/explain #terminalLastCommand`

`/fix #terminalLastCommand` is causing an error: "command not found: deploy.sh"

Use Voice Interactions

Prompt with your voice

With the voice control capabilities in VS Code, provided by the [VS Code Speech](#) extension, you can initiate a chat conversation by using your voice



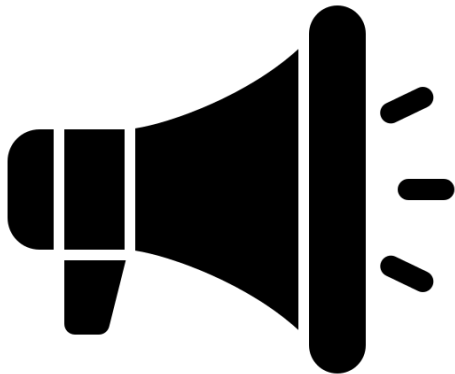
Use your voice to dictate your chat prompt



Use the "Hey Code" voice command to start a voice session with Copilot Chat



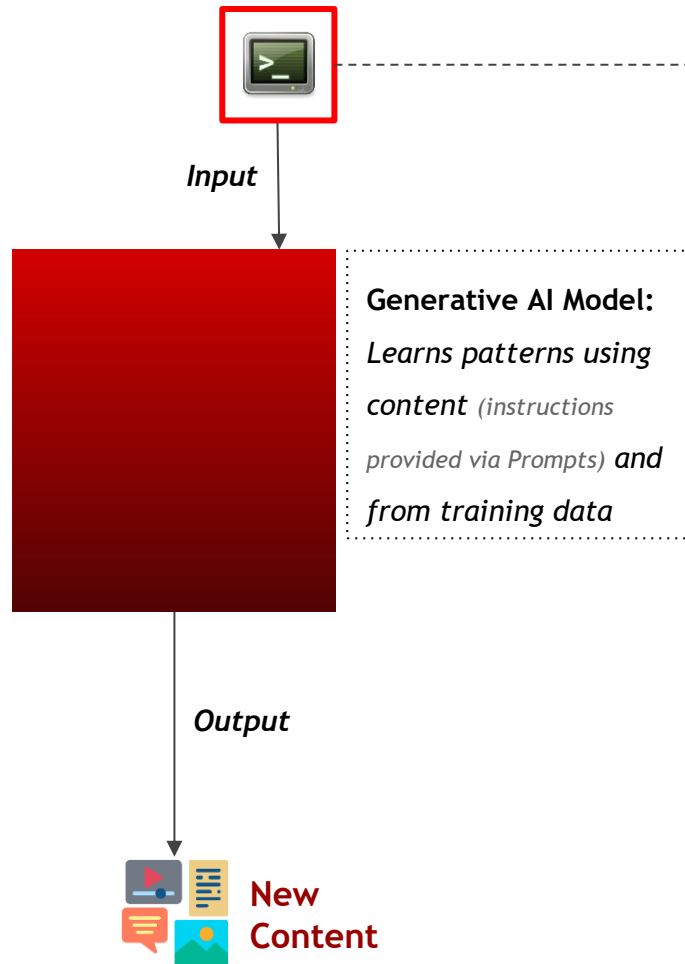
Accelerate voice input for chat by using the "hold to speak" mode



D E M O

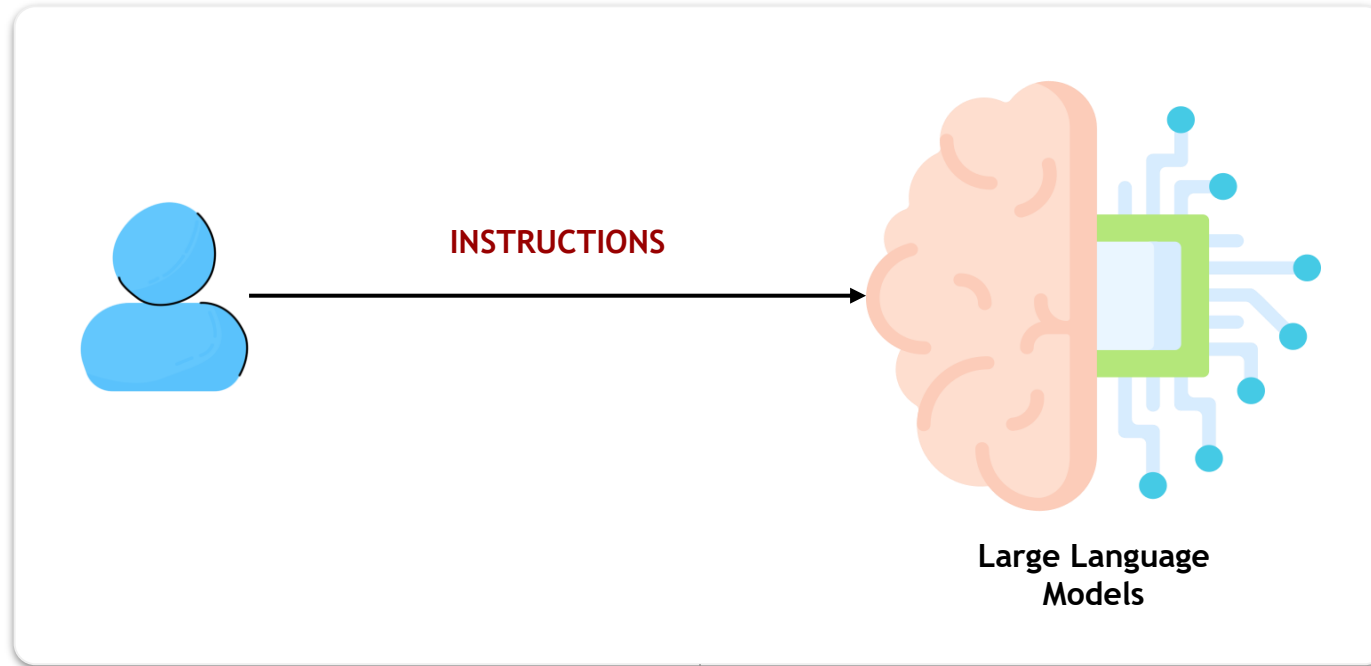


What is a Prompt?



- > *A prompt refers to a specific instruction, query, or input given to an AI model to generate a response or perform a task*
- > *It serves as cue, guidance, or a starting point for the AI system to produce an output based on its learned knowledge and language comprehension*
- > *The form and tone of a prompt may vary depending on the AI application and the desired outcome*
- > *Well-defined, context-driven, and unambiguous prompts bring AI output closer to the intended results*

What is Prompt Engineering?



Prompt Engineering is an art of asking the right question to get the best response from an LLM

What is Prompt Engineering?



Developer

As per me,

PROMPT

Code blocks, individual lines of code, or natural language comments a developer writes to generate a specific suggestion from GitHub Copilot

PROMPT ENGINEERING

Providing instructions or comments in the IDE to generate specific coding suggestions

CONTEXT

Details that are provided by a developer to specify the desired output from a generative AI coding tool

What is Prompt Engineering?



ML Researcher

As per me,

PROMPT

Compilation of IDE code & relevant context (IDE comments, code in open files, etc.) that is generated by algorithms & sent to the model of a Gen AI coding tool

PROMPT ENGINEERING

Creating algorithms that will generate prompts (compilations of IDE code and context) for a large language model

CONTEXT

Details (like data from your open files & code you've written before) that algorithms send to a large language model (LLM) as additional information about the code

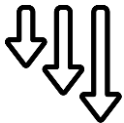
Why is Prompt Engineering Important?



Improve the Accuracy

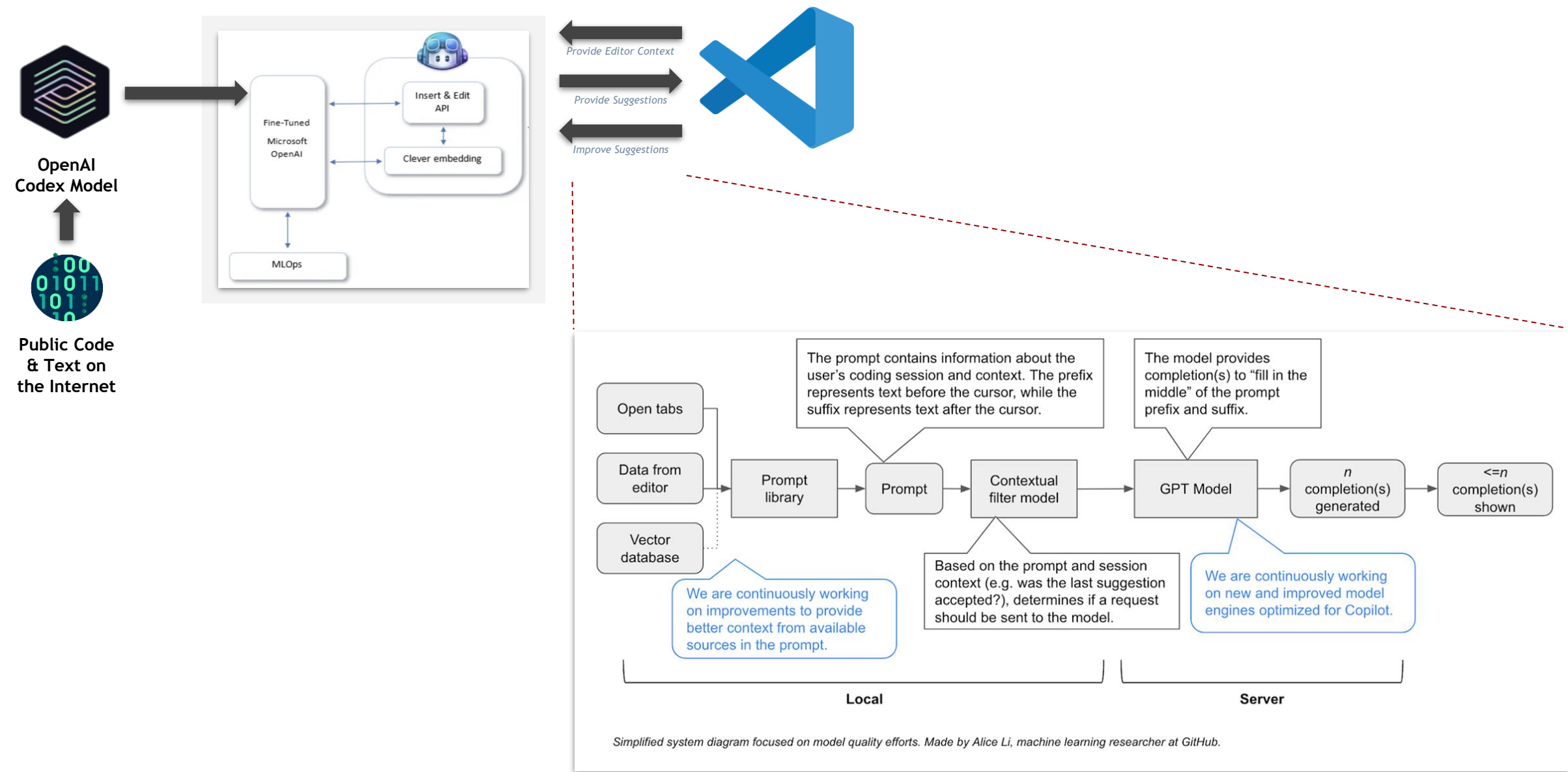


Improve the usefulness of AI generated content



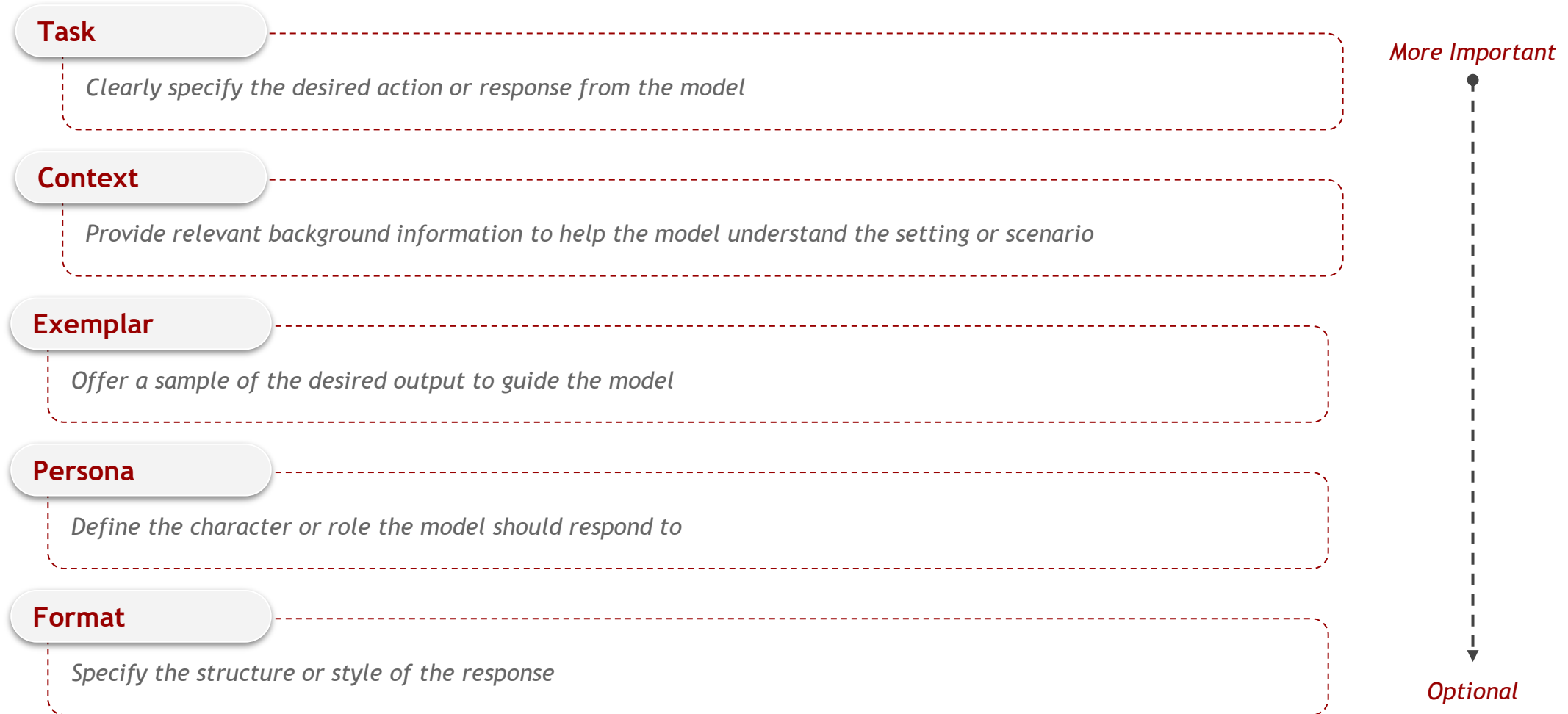
Reduce the risk of generating harmful and biased content

How Github Copilot understand your code?



Key elements of a Prompt

Let's look at aspects that make up a good prompt:



Key elements of a Prompt

Let's look at aspects that make up a good prompt:

Task

Always start the Task sentence with an Action Verb, e.g.

- **GENERATE**
- **GIVE**
- **WRITE**
- **ANALYZE**
- *and many more ...*

Key elements of a Prompt

Let's look at aspects that make up a good prompt:

Context

- What is the user's **background**?
- What does **success** look like?
- What **environment** are they in?

Create a Python script for an e-commerce website to automate customer order processing. **Keep in mind that the user is a mid-level software engineer familiar with web development and APIs but new to handling large-scale transaction data.** **Success** means the script is efficient, easy to maintain, and includes error handling for failed API calls. **The user is working in a fast-paced startup environment with limited resources and a tight deadline.**

Key elements of a Prompt

Let's look at aspects that make up a good prompt:

Exemplar

Exemplars are not necessary for every prompt but including them as relevant examples greatly improves the quality of your output

Develop a REST API using Flask that handles user authentication for a web application. The user is a backend developer new to Flask but familiar with Django. Success involves creating an API with endpoints for registration, login, and logout. Provide an example of an API response for a successful login, including the structure of the JSON data and the status codes. Additionally, include sample error responses to demonstrate proper error handling.

Example Pseudocode:

First, import Flask and the necessary modules. Initialize a Flask application. Create a simple data dictionary to store user credentials:

```
from flask import Flask, request, jsonify
```

```
app = Flask(name)
```

```
users = {"test_user": "password123"}
```

Define a '/login' endpoint using the **POST** method. Retrieve the username and password from the incoming JSON request. Check if the username exists and matches the password. If successful, return a JSON response with a success status, a message, and an example token with a 200 status code.

If the credentials do not match, return an error response with a status of "error" and an "Invalid credentials" message, using a 401 status code.

Finally, run the Flask application in debug mode.

Expected Successful Response:

```
{ "status": "success",  
  "message": "Login successful!",  
  "token": "example_token_123"  
}
```

Expected Error Response:

```
{ "status": "error",  
  "message": "Invalid credentials."  
}
```

Key elements of a Prompt

Let's look at aspects that make up a good prompt:

Persona

Who do you want the AI to be? Think of someone you wish you had instant access to with the task you're facing

IF YOU'RE A SENIOR BACKEND ENGINEER

Imagine you are a senior backend engineer with over 10 years of experience in building secure and scalable web applications.

You've worked extensively with Flask, Django, and REST API design. I'm a mid-level developer building a REST API for user authentication in Flask. Guide me as a mentor would, helping me design the login endpoint to handle user credentials securely, implement proper error handling, and return appropriate JSON responses. Break down complex concepts into simple steps, and provide best practices to ensure the API is robust and secure.

Key elements of a Prompt

Let's look at aspects that make up a good prompt:

Format

Set the structure of the output from GPT-Models

Create a detailed guide for building a user authentication REST API using Flask. Structure the response into the following sections:

- 1. **Introduction:** Briefly explain the purpose of user authentication and why Flask is a good choice.*
- 2. **Requirements:** List all the libraries and tools needed to build the API.*
- 3. **Step-by-Step Instructions:***
 - Include a clear breakdown of each step with explanations.*
 - Provide code snippets for each part, starting from setting up the Flask app, creating endpoints for registration, login, and logout.*
- 4. **Code Walkthrough:** Explain the key parts of the code, focusing on how authentication is handled.*
- 5. **Best Practices:** Highlight important considerations such as security measures, error handling, and scalability.*
- 6. **Sample API Responses:** Show example JSON responses for successful login, registration, and error scenarios.*
- 7. **Conclusion:** Summarize the key takeaways and next steps.*

Ensure each section is clearly labeled and presented in a concise manner. Use simple language and provide comments within code snippets for better understanding

Key elements of a Prompt

Let's look at aspects that make up a good prompt:

You are a senior software architect with over 15 years of experience in developing secure and scalable web applications. I am a mid-level developer, working in a fast-paced startup environment, and I need to create a REST API for user authentication using Flask. I have experience with Django but am new to Flask and handling large-scale authentication systems.

Your task is to guide me through building a simple but secure authentication API. Please provide clear, step-by-step instructions. The goal is to ensure the API has endpoints for registration, login, and logout, implements token-based authentication, and includes proper error handling.

Structure the response as follows:

Introduction: A brief overview of user authentication and why Flask is suitable for this task.

Requirements: List of libraries and tools needed.

Step-by-Step Implementation: Break down the process with code snippets and explanations. Include comments within the code for clarity.

Best Practices: Advice on security measures, error handling, and scalability.

Sample Code Response: Provide an example response for a successful login and an error case.

For the 'Step-by-Step Implementation' section, include a sample code snippet. For example, when implementing the login endpoint, show how to handle user input, validate credentials, and return a JSON response with a success message and token

Good Prompts vs. Bad Prompts

The quality of the prompt significantly influences the output generated by the model. Here are the traits of Good & Bad Prompts:



Tips for designing Prompts

Write Clear Instructions



Help me build an authentication system



Guide me through building a user authentication system in Flask. Include steps for creating endpoints for user registration, login, and logout. Use token-based authentication and include code examples for each step. Make sure to explain how to handle errors and security best practices

Change your prompt as per following to make it clearer:

- *Include details in your query to get more relevant answers*
- *Ask the model to adopt a persona*
- *Use delimiters to clearly indicate distinct parts of the input*
- *Specify the steps required to complete a task*
- *Provide examples*

Tips for designing Prompts

Ask to provide reference text when required



Explain data encryption



Explain data encryption as it relates to API security. Provide reference examples from standard security guidelines like OWASP if possible. Include practical examples for implementing encryption in a Python application

Change your prompt like this to make it clearer:

- *Instruct the model to answer using a reference text/website*
- *Instruct the model to answer with citations from a reference text*

Tips for designing Prompts

Split a complex task into simpler subtasks



Build a complete e-commerce website



Break down the process of building an e-commerce website into simpler steps. Start by outlining the initial setup, then move on to creating a product catalog, implementing user authentication, setting up a shopping cart, integrating a payment gateway, and adding order tracking features. Provide instructions and sample code snippets for each part.

Use the following tactics to improve the prompt:

- *Use intent classification to identify the most relevant instructions for a user query*
- *For dialog-driven applications that require very long conversations, summarize or filter previous dialog(s)*
- *Summarize long documents piecemeal and construct a complete summary, recursively*

Tips for designing Prompts

Test changes systematically



Test if my new login feature works

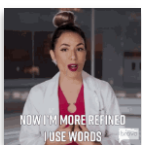
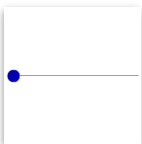
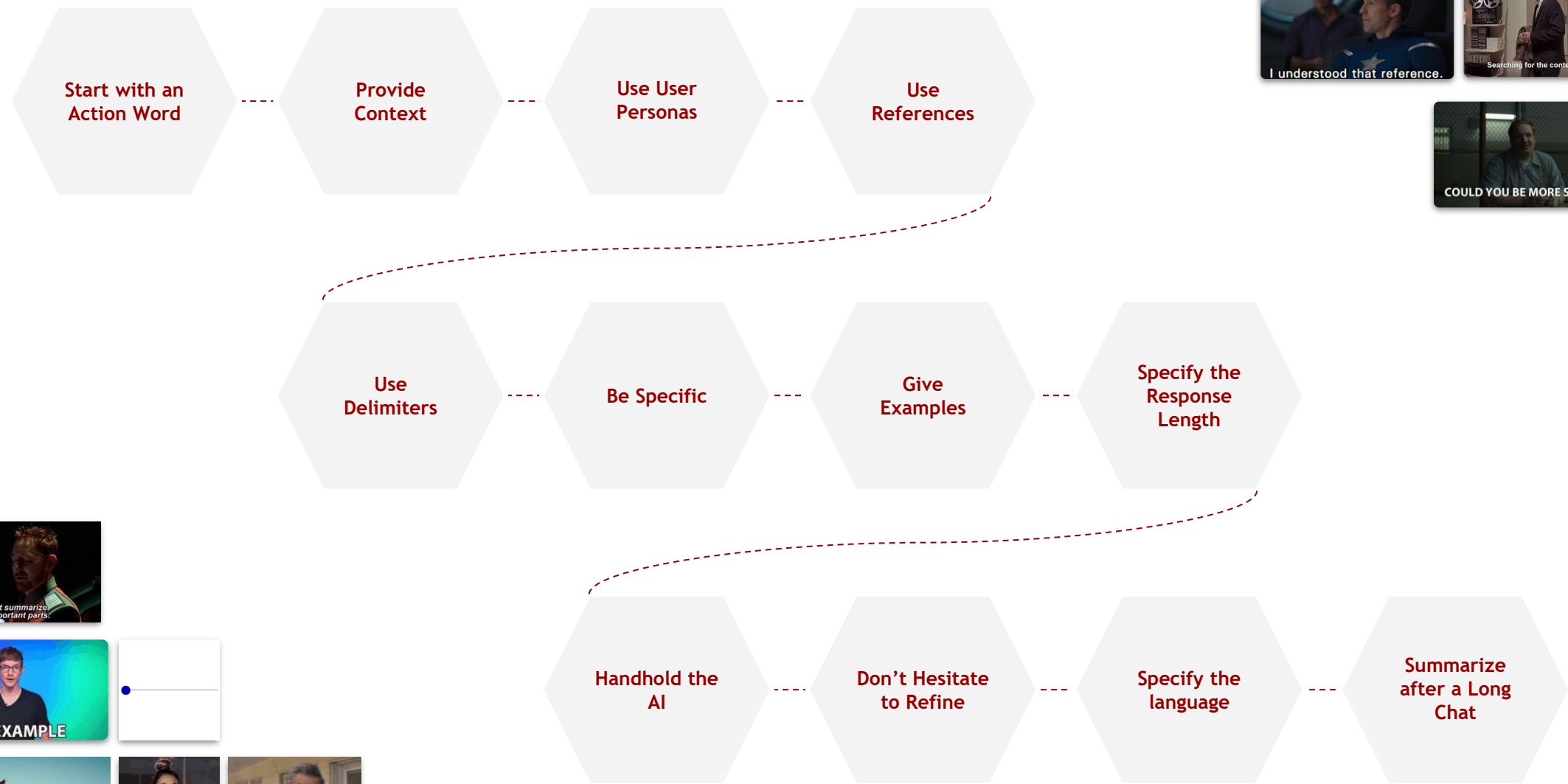


Test the new login feature systematically. Start by verifying the input validation for username and password fields. Next, test the API response for both successful and failed login attempts. Finally, check if the token is generated correctly and validate the security of stored user data. Provide detailed feedback on each test case

Use the following tactics to improve the prompt:

- *Provide a structured approach to testing*
- *Specify which parts to focus on and how to systematically go through the process*

Summarizing Prompt Engineering Best Practices



D E M O



A low-angle, upward-looking photograph of several modern skyscrapers with glass facades. The image is overlaid with a semi-transparent dark grey rectangle in the center, which contains the text 'THANK YOU'. Additionally, there are three solid red rectangular blocks: one at the top center, one at the bottom left, and one at the bottom right. A small black horizontal bar is positioned below the red block at the bottom left.

THANK YOU