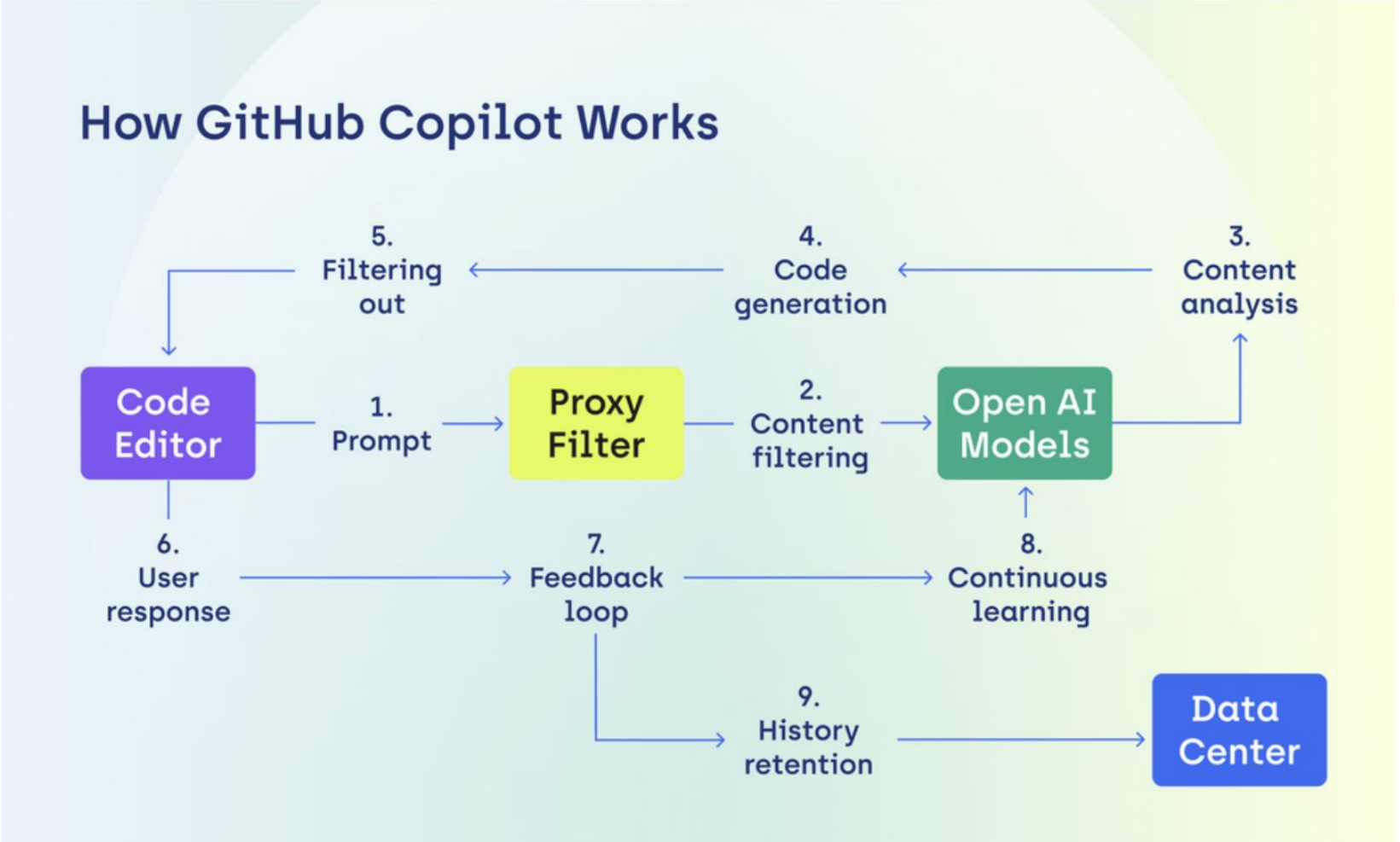
Github CoPilot

GitHub Copilot

AI pair programmer that helps you write code faster and with less work. It draws context from comments and code to suggest individual lines and whole functions instantly.

It has been trained on natural language text and source code from publicly available sources, including code in public repositories on GitHub.

How it works?


How GitHub Copilot Works
5. Filtering out
4. Code generation
3. Content analysis
Code Editor
1. Prompt
Proxy Filter
2. Content filtering
Open AI Models
6. User response
7. Feedback loop
8. Continuous learning
9. History retention
Data Center

How it works?

When you input a **prompt in the code editor** (either as comments or via the GitHub Copilot Chat), the information travels securely via HTTPS to **GitHub Copilot's servers**.

There, a **proxy filter sifts through the data**, removing any personal information and filtering out inappropriate content, like hate speech, offensive language, and other harmful material.

Following this, Copilot analyzes **keywords, phrases, and contextual clues** to ensure it fully understands your prompt. The AI then translates your natural language request into actionable coding tasks and **generates code** customized to the project's language, framework, and your preferences.

In the blink of an eye, GitHub Copilot's code suggestion appears in the editor! Now, it's your turn to respond: you can accept it, make adjustments, or reject it completely, triggering a **feedback loop**. With each interaction, Copilot learns and stores the **history of your prompts and responses** in its data center.

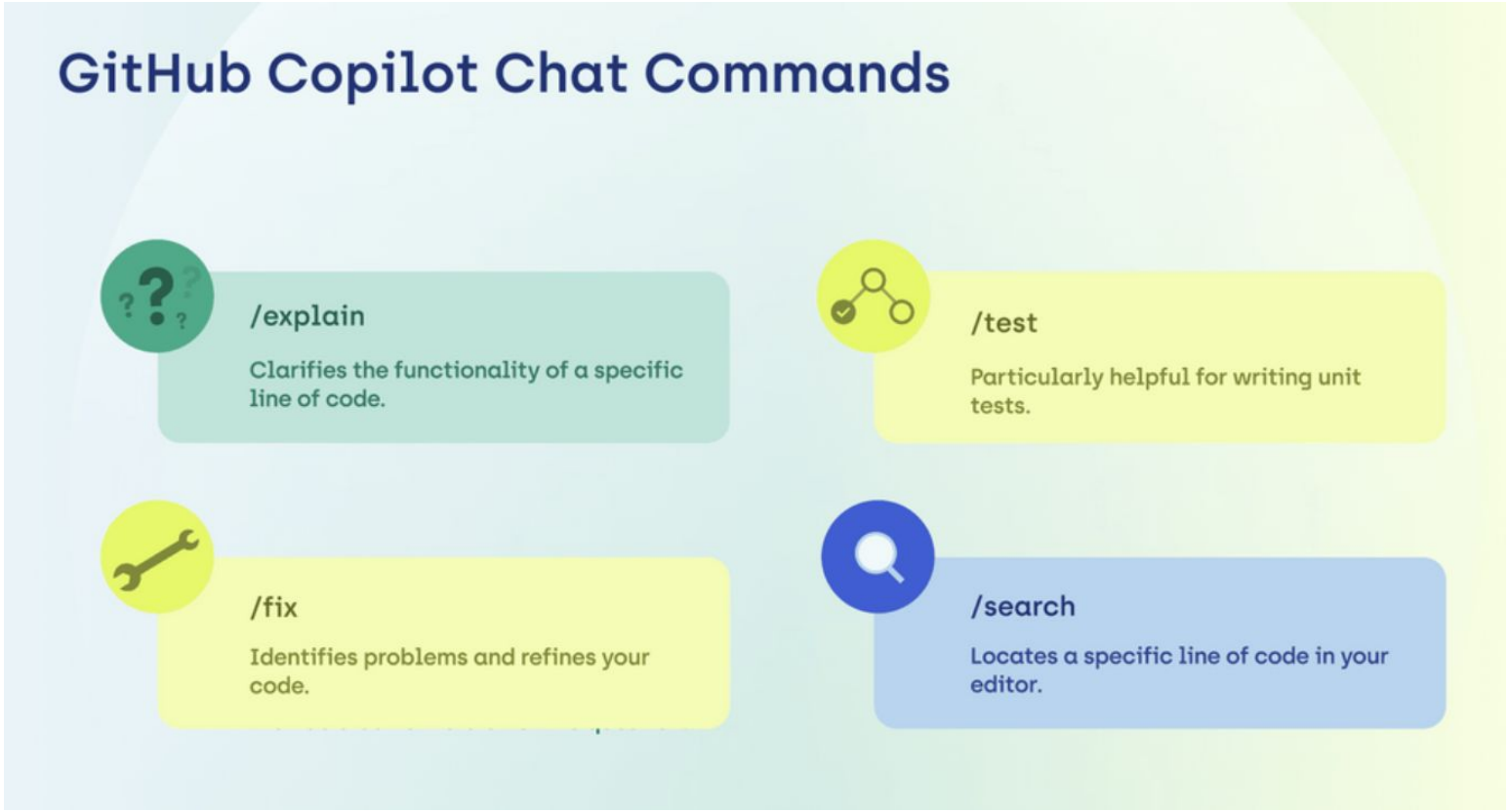
Modes

Ask Mode - Use it when you need help understanding or researching something. No code changes.

Edit Mode - Use it to apply small, specific changes while staying in control

Agent Mode - Agent mode uses tools to accomplish specialized tasks while processing a user request

Chat Commands



Context

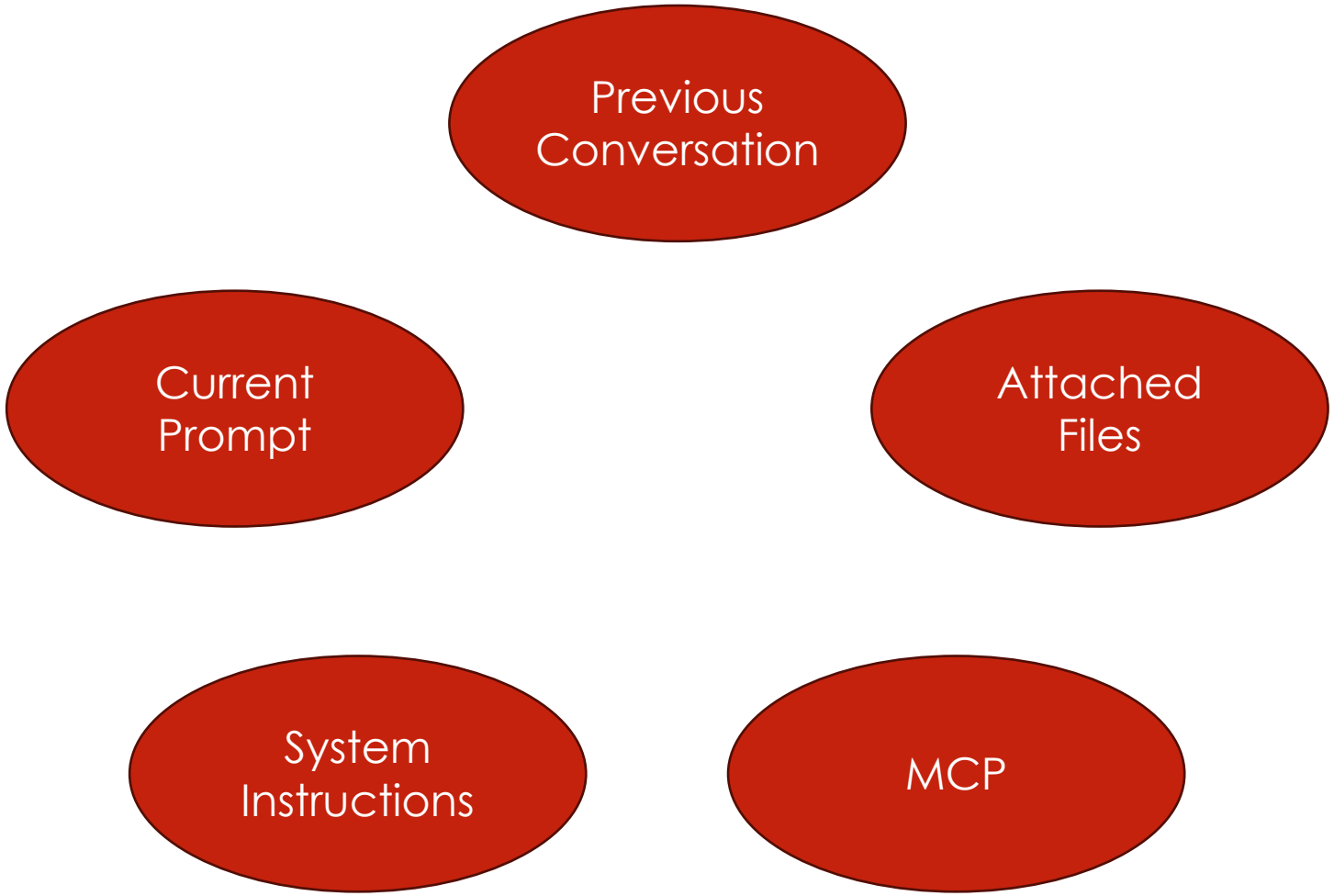
Copilot tries to determine the intent and scope of your question based on your natural language chat prompt.

Adding Context Helps GitHUb Copilot to give you the best and the most relevant

Context Types

- **Files** - include specific files from your workspace in the prompt
- **Folders** - add a folder to include the files in that folder in the prompt
- **Symbols** - add a symbol from your workspace to the prompt
- **Codebase** - let Copilot find the right files automatically
- **Editor or terminal selection** - include a selection of text from the editor or terminal in the prompt
- **Terminal** command output - include the output of the last command run in the terminal
- **Problems** - include a specific code issue from the Problems panel to the prompt
- **Test failures** - include details from test failures in the prompt
- **Search** results - include the results from the Search view in the prompt
- **Tools** - include tools in the prompt, for example the #fetch tool or tools from extensions or MCP servers

Context


Previous Conversation
Current Prompt
Attached Files
System Instructions
MCP

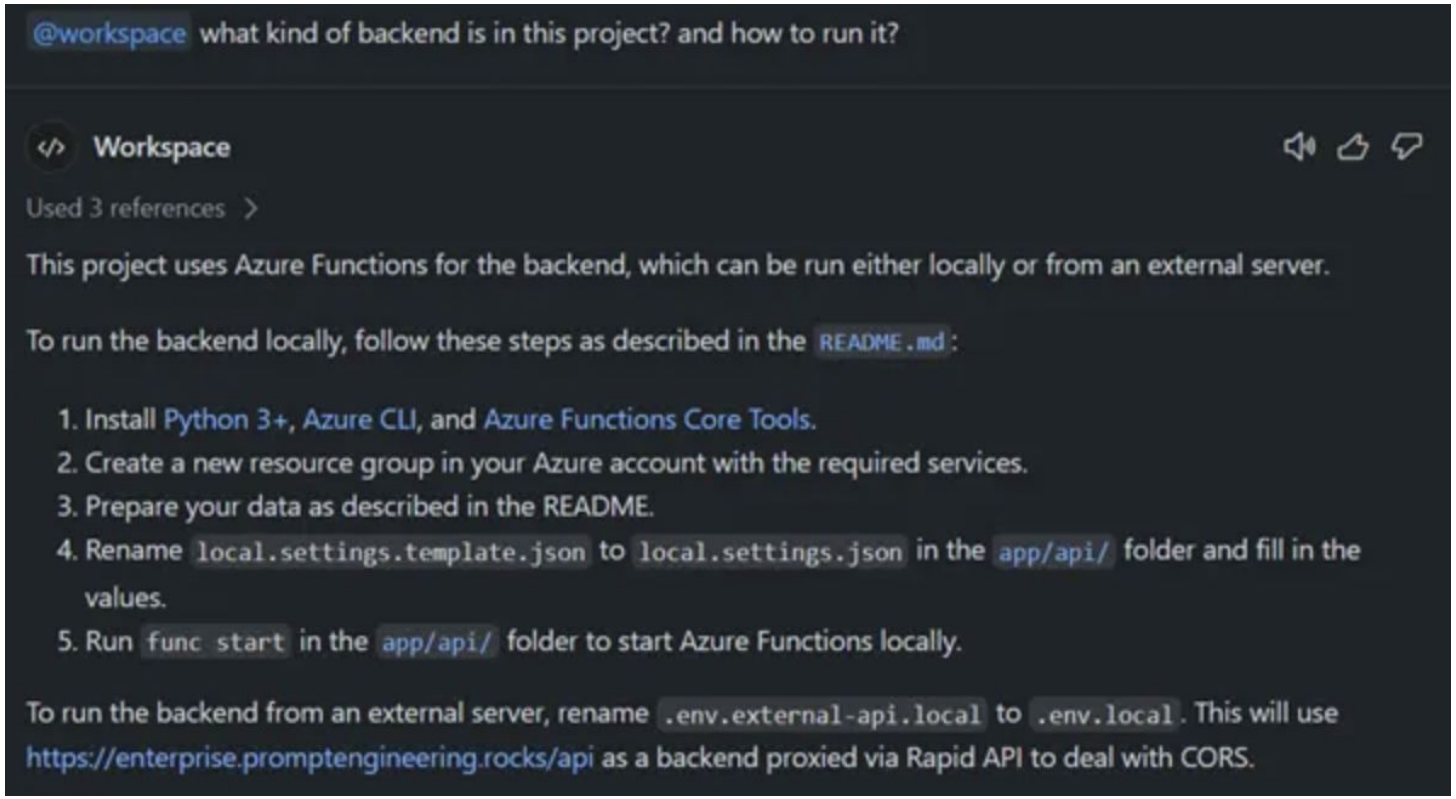
Chat Participants

In GitHub Copilot Chat, you can reference one of the AI-powered “domain experts” using conventional chat syntax — by prepending @ to the participant name.

- @workspace: Knows everything about the code in your currently open workspace. This is the chat participant you will most likely communicate with frequently.
- @terminal: Knows all about the integrated terminal shell, its contents, and its buffer.
- @vscode: Knows about the VS Code editor, its commands, and features.

Chat Participants - Example

Let's get information about the backend part of the project we've just been assigned to by asking the @workspace *chat participant right after we open the project folder in VS Code.*



Chat Variables

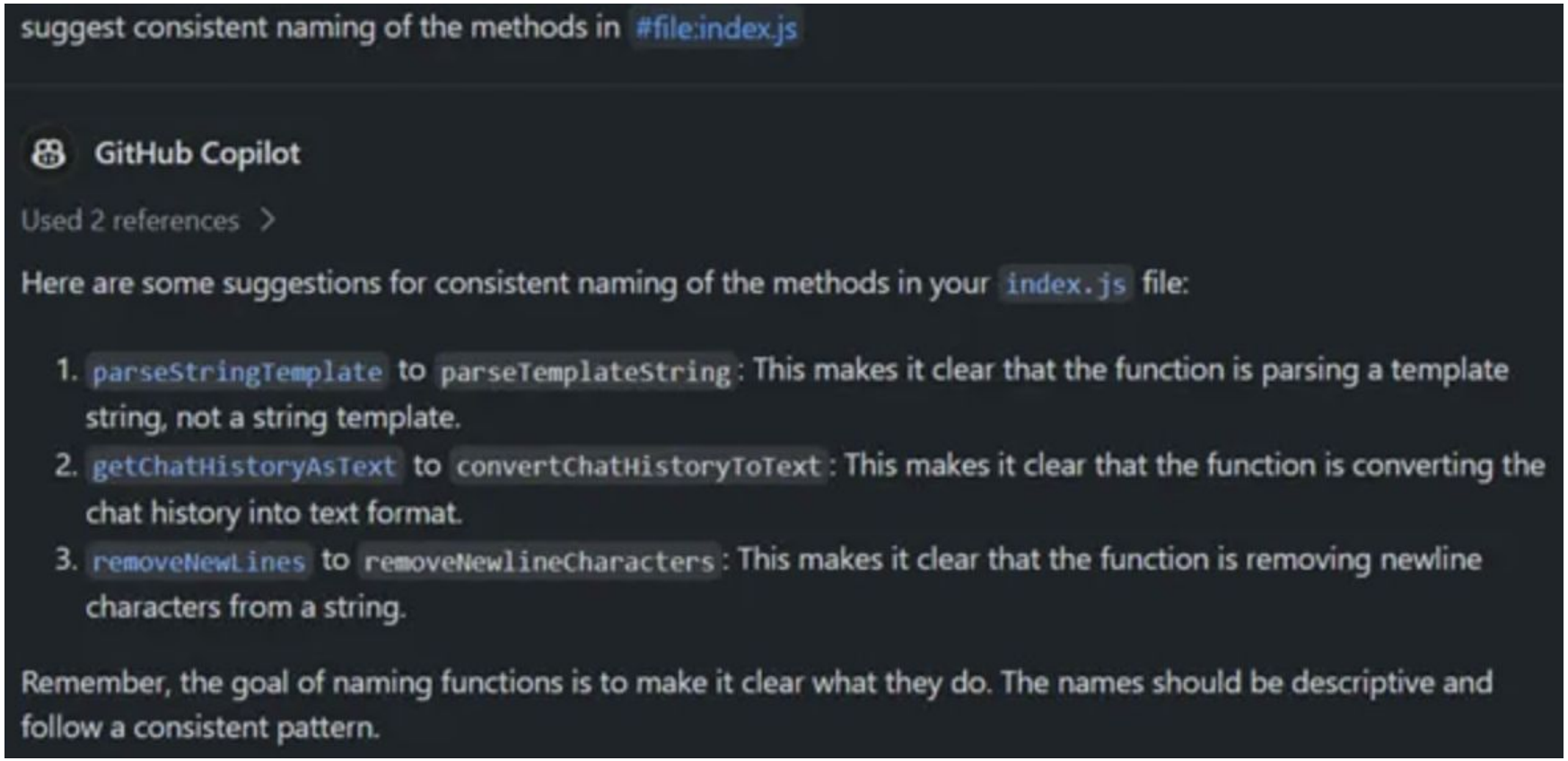
In many cases, considering the full solution as the context for your question or request (by using @workspace) is overkill.

You might want to point to specific files or even parts of the files in your message. This is what Chat variables is aimed for.

- #file: Points to a specific file in your workspace.
- #codebase: All content of the open workspace. It’s similar to using @workspace and might be useful when you chat with another agent (like @terminal) but still want to reference the full solution.
- #editor: Source code in the editor’s viewport (visible part).
- #git: Current git repository: branch, remotes, path, etc.
- #selection: The currently selected code.
- #terminalLastCommand: Last run command in the editor’s terminal.
- #terminalSelection: Selection in the editor's terminal.

Chat Variables - Example

Let’s get help on improving method names in a specific file (and we want to ensure that the whole content of the file is taken into consideration).



Instructions

The ` .github/copilot-instructions.md` file allows you to define custom coding standards, rules, and best practices for GitHub Copilot Chat to follow within a specific repository or workspace.

It essentially provides a set of instructions that Copilot Chat will use as a reference when generating code or providing explanations.

MCP Servers

Full Integration with MCP Servers