Prompt engineering for Copilot Chat

Follow these strategies to improve your Copilot results.

In this article

A prompt is a request that you make to GitHub Copilot. For example, a question that you ask Copilot Chat, or a code snippet that you ask Copilot to complete. In addition to your prompt, Copilot uses additional context, like the code in your current file and the chat history, to generate a response.

Follow the tips in this article to write prompts that generate better responses from Copilot.

Start general, then get specific

When writing a prompt for Copilot, first give Copilot a broad description of the goal or scenario. Then list any specific requirements.

For example:

> Write a function that tells me if a number is prime
>
> The function should take an integer and return true if the integer is prime
>
> The function should error if the input is not a positive integer

Give examples

Use examples to help Copilot understand what you want. You can provide example input data, example outputs, and example implementations.

For example:

> Write a function that finds all dates in a string and returns them in an array. Dates can be formatted like:
>
> - 05/02/24
> - 05/02/2024
> - 5/2/24
> - 5/2/2024
> - 05-02-24
> - 05-02-2024
> - 5-2-24
> - 5-2-2024
>
> Example:
>
> findDates("I have a dentist appointment on 11/14/2023 and book club on 12-1-23")
>
> Returns: ["11/14/2023", "12-1-23"]

Unit tests can also serve as examples. Before writing your function, you can use Copilot to write unit tests for the function. Then, you can ask Copilot to write a function described by those unit tests.

Break complex tasks into simpler tasks

If you want Copilot to complete a complex or large task, break the task into multiple simple, small tasks.

For example, instead of asking Copilot to generate a word search puzzle, break the process down into smaller tasks, and ask Copilot to accomplish them one by one:

- Write a function to generate a 10 by 10 grid of letters.
- Write a function to find all words in a grid of letters, given a list of valid words.
- Write a function that uses the previous functions to generate a 10 by 10 grid of letters that contains at least 10 words.
- Update the previous function to print the grid of letters and 10 random words from the grid.

Avoid ambiguity

Avoid ambiguous terms. For example, don't ask "what does this do" if "this" could be the current file, the last Copilot response, or a specific code block. Instead, be specific:

- What does the `createUser` function do?
- What does the code in your last response do?

Ambiguity can also apply to libraries:

- If you are using an uncommon library, describe what the library does.
- If you want to use a specific library, set the import statements at the top of the file or specify what library you want to use.

Indicate relevant code

If you are using Copilot in your IDE to get suggestions as you code, open any relevant files and close irrelevant files. Copilot will use the open files to understand your request.

If you are using Copilot Chat in your IDE, open the file or highlight the code that you want Copilot to reference. You can also use keywords to manually supply context to Copilot Chat. For example, you can add the `@workspace` chat participant in VS Code, or `@project` in JetBrains IDEs. See GitHub Copilot Chat cheat sheet.

Experiment and iterate

If you don't get the result that you want, iterate on your prompt and try again.

If you are using Copilot to get suggestions as you code, you can delete the suggestion entirely and start over. Or you can keep the suggestion and request modifications.

If you are using Copilot Chat, you can reference the previous response in your next request. Or, you can delete the previous response and start over.

Keep history relevant

Copilot Chat uses the chat history to get context about your request. To give Copilot only the relevant history:

- Use threads to start a new conversation for a new task
- Delete requests that are no longer relevant or that didn't give you the desired result

Follow good coding practices

If you aren't getting the responses you want when you ask Copilot for suggestions or explanations in your codebase, make sure that your existing code follows best practices and is easy to read. For example:

- Use a consistent code style and patterns
- Use descriptive names for variables and functions
- Comment your code
- Structure your code into modular, scoped components
- Include unit tests

> **Tip**
>
> Use Copilot to help your code follow best practices. For example, ask Copilot to add comments or to break a large function into smaller functions.

Further reading

- How to use GitHub Copilot: Prompts, tips, and use cases in the GitHub blog
- Using GitHub Copilot in your IDE: Tips, tricks, and best practices in the GitHub blog
- A developer's guide to prompt engineering and LLMs in the GitHub blog
- Prompting GitHub Copilot Chat to become your personal AI assistant for accessibility in the GitHub blog

Legal