

# Customizing Copilot

SOFTWARE DEVELOPMENT WITH GITHUB COPILOT



Thalia Barrera

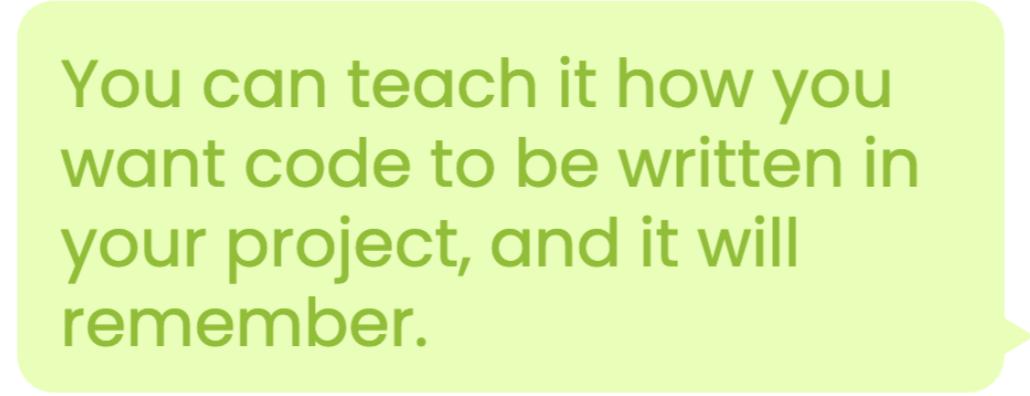
AI Engineering Curriculum Manager,  
DataCamp

How can I configure  
GitHub Copilot?





How can I configure  
GitHub Copilot?



You can teach it how you  
want code to be written in  
your project, and it will  
remember.





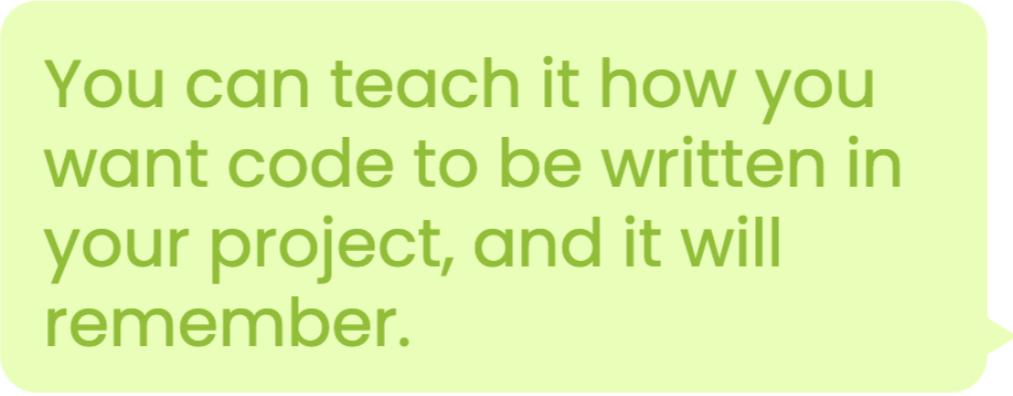
How can I configure  
GitHub Copilot?

You can teach it how you  
want code to be written in  
your project, and it will  
remember.

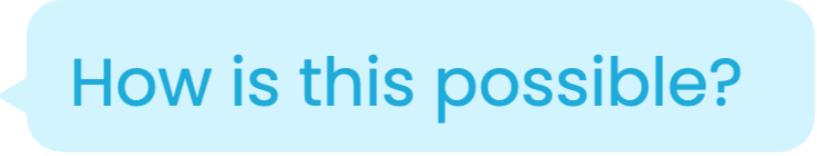
How is this possible?



How can I configure  
GitHub Copilot?



You can teach it how you  
want code to be written in  
your project, and it will  
remember.



How is this possible?



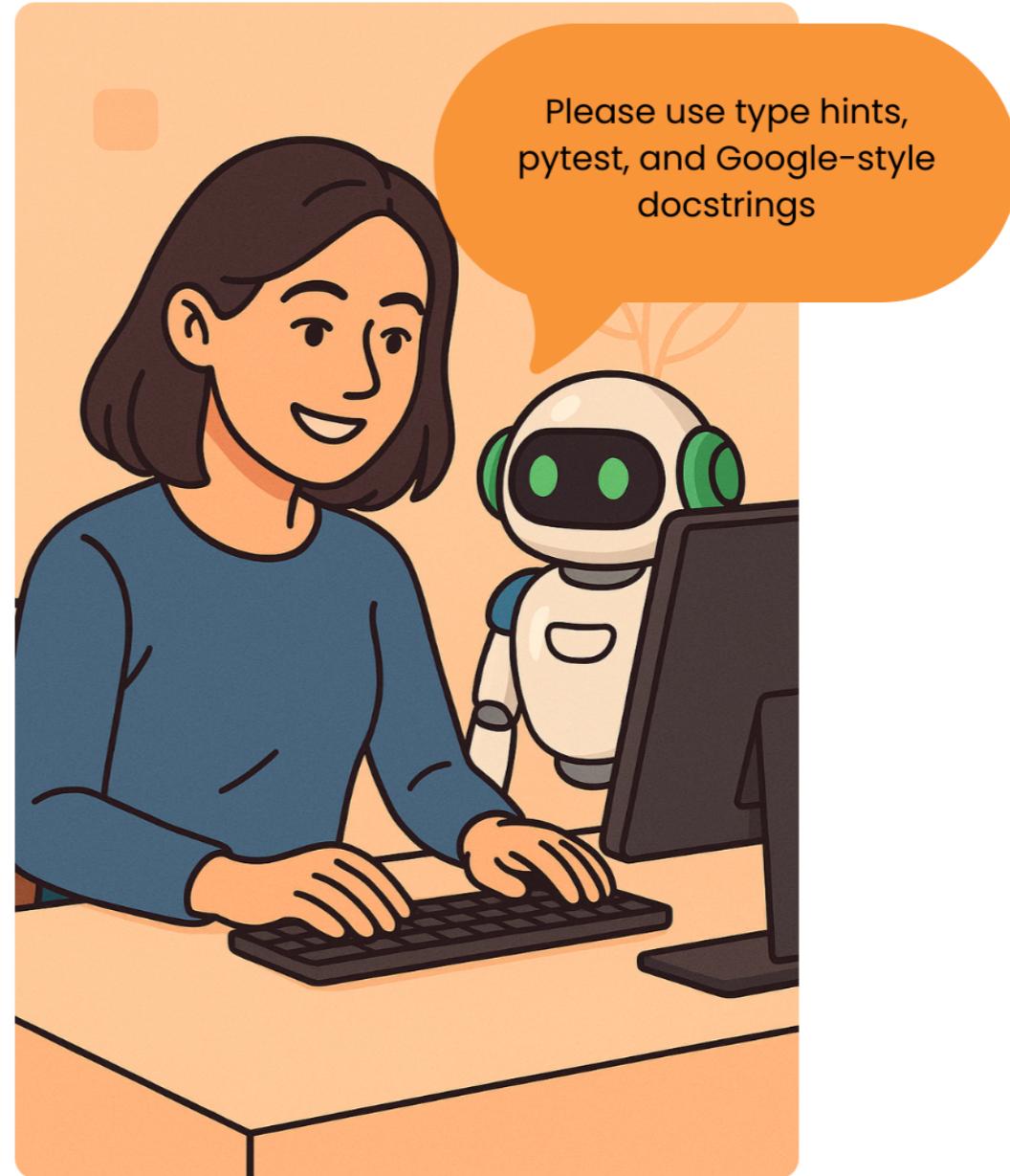
Through custom  
instructions.



# How custom instructions work

Define:

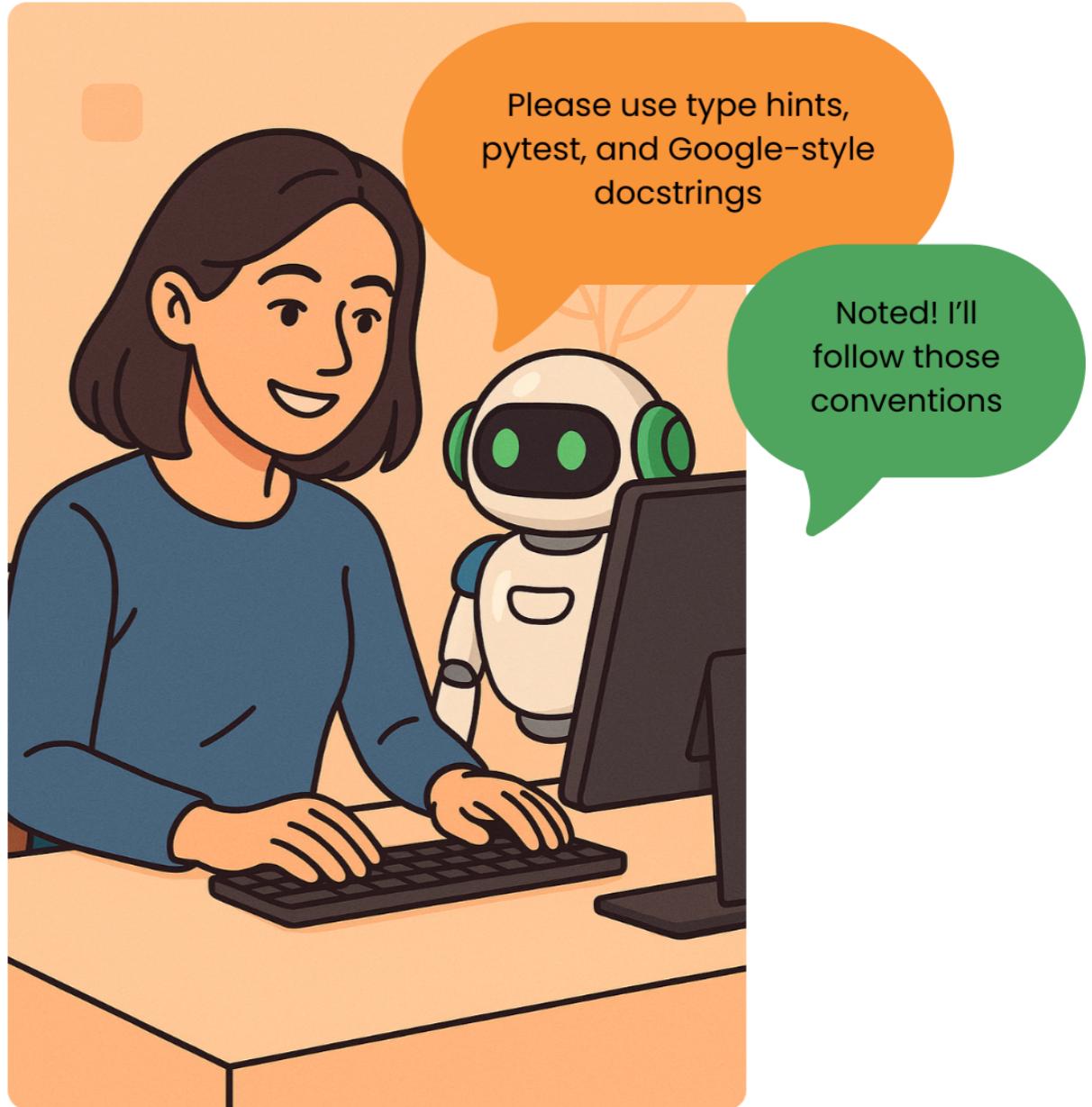
- Coding conventions
- Tool preferences
- Project context



# How custom instructions work

Define:

- Coding conventions
- Tool preferences
- Project context



# How to define custom instructions

Include them in: `.github/copilot-instructions.md`

```
## Project context
Python project for processing data pipelines with Airflow.

## Coding style
- Use type hints for all functions
- Follow PEP8
- Use snake_case naming
- Write Google-style docstrings

## Testing
- Use pytest
- Include one test per function

## More instructions...
```

# What makes an effective instruction

## Effective instructions:

- Short and specific
- Self-contained
- Add context or relevant information

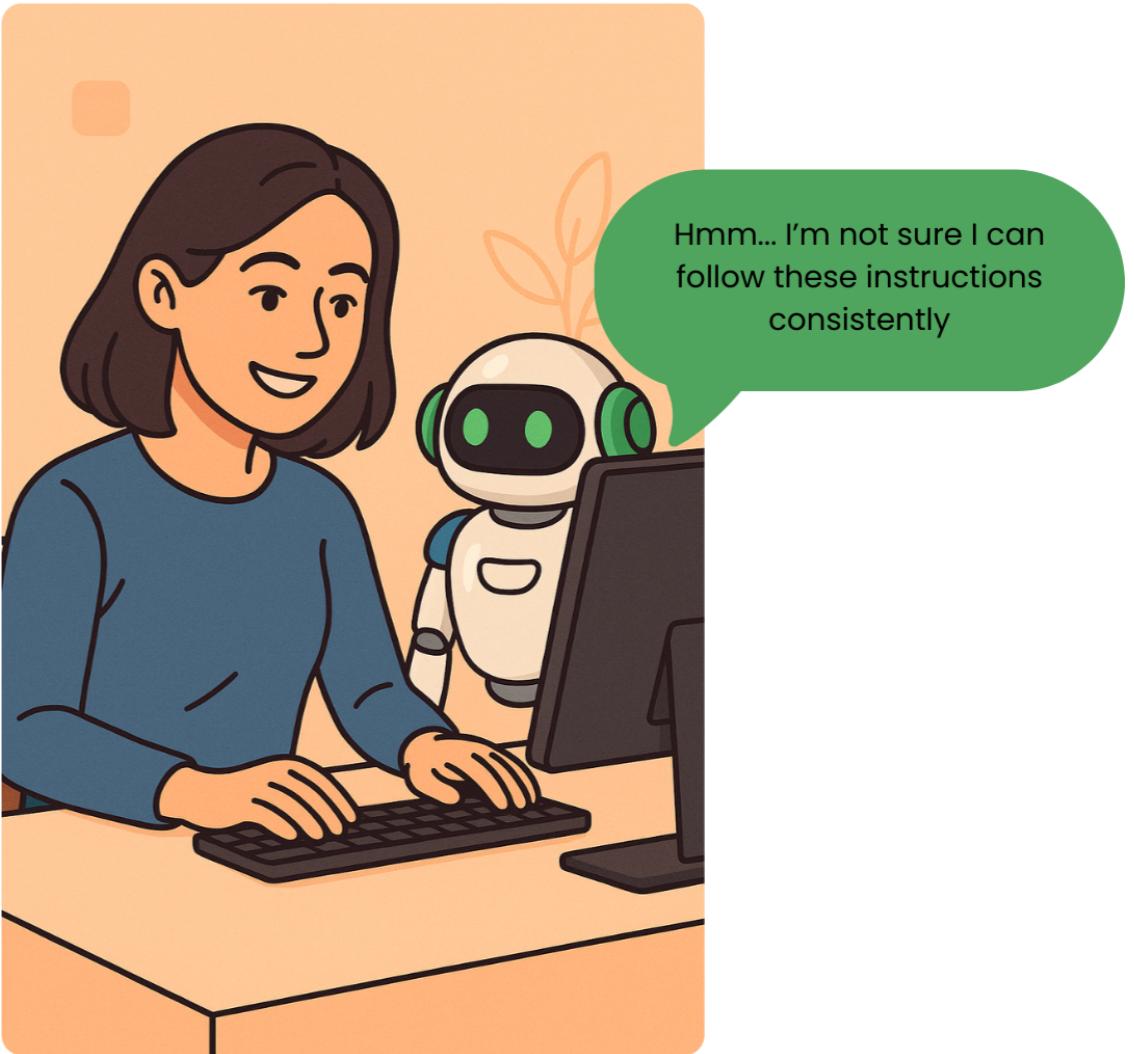
## Ineffective instructions:

- External documents or repositories
- Specific tone or writing style
- Limit on detail or formatting

<sup>1</sup> <https://docs.github.com/en/copilot/how-tos/configure-custom-instructions/add-repository-instructions>

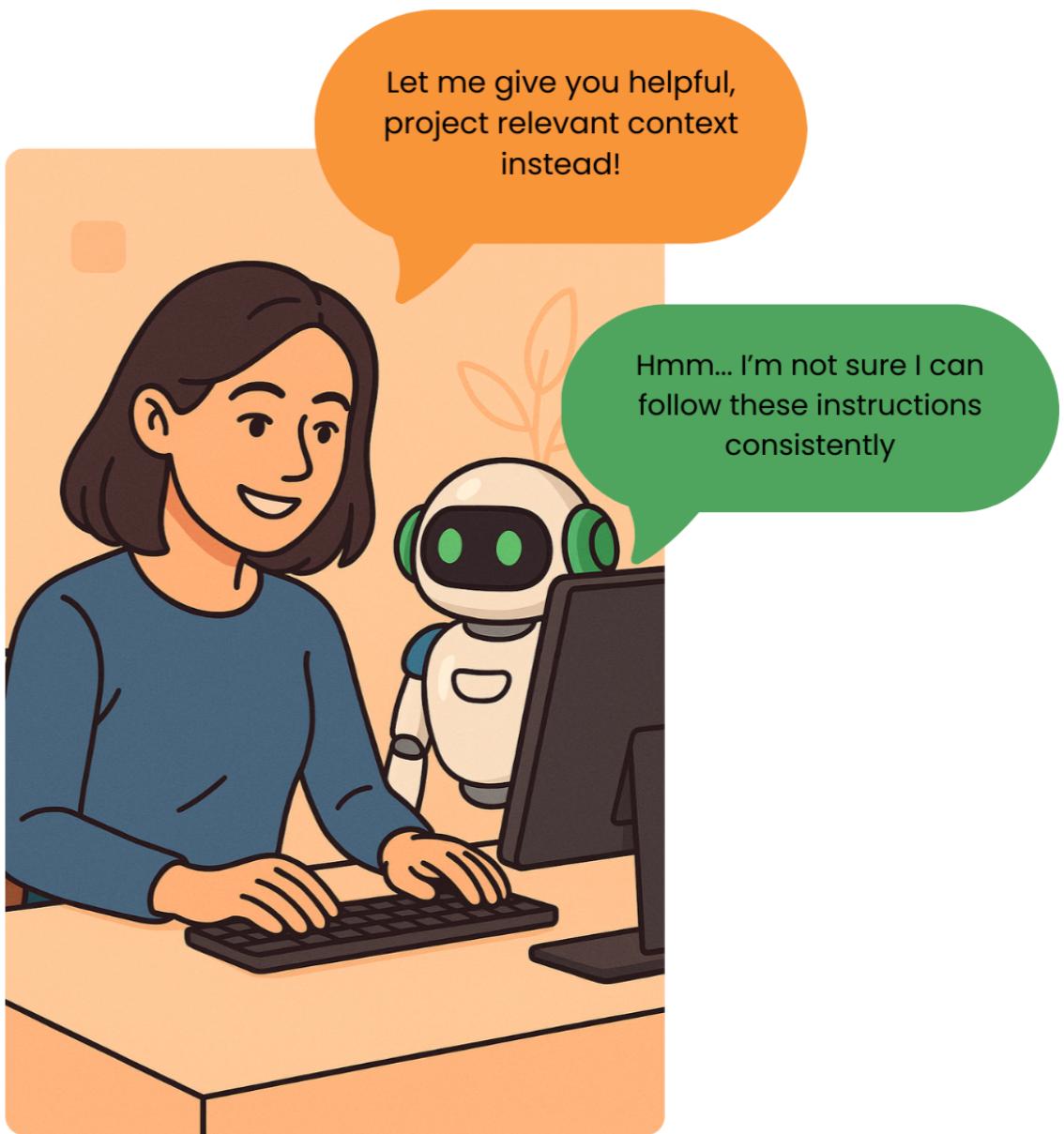
## ## Style guidelines

- Conform to the coding styles defined in `styleguide.md` in repo `my-org/my-repo`
- Answer in the style of a friendly colleague, using informal language
- Answer in fewer than 1,000 characters



## ## Style guidelines

- Conform to the coding styles defined in `styleguide.md` in repo `my-org/my-repo`
- Answer in the style of a friendly colleague, using informal language
- Answer in fewer than 1,000 characters



# How to enable custom instructions

- Save the `copilot-instructions.md` file
- It will be automatically included in your prompts

The screenshot shows the GitHub Copilot settings interface. On the left, there's a sidebar with 'User' and 'Workspace' tabs, and a list of commonly used features like Text Editor, Workbench, Window, Features, Application, Security, Extensions (.ipynb Support), CSS Language Features, Dev Containers, Emmet, GHE.com & GitHub E..., Git, GitHub, GitHub Copilot, and GitHub Copilot Chat. 'GitHub Copilot Chat' is currently selected, indicated by a blue border. The main area has a dark background with white text. It displays sections for 'Extensions', 'GitHub Copilot', and 'GitHub Copilot Chat'. Under 'GitHub Copilot Chat', there are three configuration items:

- Copilot > Chat > Agent: Auto Fix**: A checked checkbox labeled 'Automatically fix diagnostics for edited files.'
- Copilot > Chat > Code Generation: Use Instruction Files**: A checked checkbox labeled 'Controls whether code instructions from `.github/copilot-instructions.md` are added to Copilot requests.' Below it is a note: 'Note: Keep your instructions short and precise. Poor instructions can degrade Copilot's quality and performance. [Learn more about customizing Copilot.](#)'
- Copilot > Chat: Custom Instructions In System Message**: A checked checkbox labeled 'When enabled, custom instructions and mode instructions will be appended to the system message instead of a user message.'

# How to enable custom instructions

- Save the `copilot-instructions.md` file
- It will be automatically included in your prompts

The screenshot shows the GitHub Copilot settings interface. On the left, there's a sidebar with 'User' and 'Workspace' tabs, and a list of 'Commonly Used' features like Text Editor, Workbench, Window, Features, Application, Security, Extensions (.ipynb Support), CSS Language Features, Dev Containers, Emmet, GHE.com & GitHub E..., Git, GitHub, GitHub Copilot, and GitHub Copilot Chat. 'GitHub Copilot Chat' is currently selected. The main area has sections for 'Extensions', 'GitHub Copilot', and 'GitHub Copilot Chat'. Under 'GitHub Copilot Chat', there are three sections: 'Copilot > Chat > Agent: Auto Fix' (with a checked checkbox for 'Automatically fix diagnostics for edited files.'), 'Copilot > Chat > Code Generation: Use Instruction Files' (with a checked checkbox for 'Controls whether code instructions from `.github/copilot-instructions.md` are added to Copilot requests.'), and 'Copilot > Chat: Custom Instructions In System Message' (with a checked checkbox for 'When enabled, custom instructions and mode instructions will be appended to the system message instead of a user message.'). A note below the first section says: 'Note: Keep your instructions short and precise. Poor instructions can degrade Copilot's quality and performance. [Learn more about customizing Copilot.](#)'

# Using multiple instruction files

```
myproject/
└── api/
    ├── handlers.py
    └── utils.py
└── tests/
    ├── test_handlers.py
    └── test_utils.py
└── core/
    ├── models.py
    └── service.py
└── .github/
    └── instructions/
        ├── python_core.instructions.md
        ├── api_instructions.instructions.md
        └── tests_instructions.instructions.md
```

# Using multiple instruction files

```
myproject/
└── api/
    ├── handlers.py
    └── utils.py
└── tests/
    ├── test_handlers.py
    └── test_utils.py
└── core/
    ├── models.py
    └── service.py
└── .github/
    └── instructions/
        ├── python_core.instructions.md
        ├── api_instructions.instructions.md
        └── tests_instructions.instructions.md
```

# Using multiple instruction files

```
myproject/
└── api/
    ├── handlers.py
    └── utils.py
└── tests/
    ├── test_handlers.py
    └── test_utils.py
└── core/
    ├── models.py
    └── service.py
└── .github/
    └── instructions/
        ├── python_core.instructions.md
        ├── api_instructions.instructions.md
        └── tests_instructions.instructions.md
```

```
api.instructions.md U X
.github > api.instructions.md
1  ---
2  description: "API layer rules"
3  applyTo: "api/**/*.py"
4  ---
5
6  # API Layer Guidelines
7
8  - Validate inputs early.
9  - Return JSON consistently.
10 - Don't include heavy business logic here – call into core / services.
```

```
tests.instructions.md U X
.github > tests.instructions.md
1  ---
2  description: "Testing rules"
3  applyTo: "tests/**/*.py"
4  ---
5
6  # Test Guidelines
7
8  - Use `pytest` style.
9  - Mock external dependencies.
10 - Name tests clearly, e.g. `test_xxx_returns_yyy`.
```

# Setting organization-level instructions

The screenshot shows the GitHub Copilot settings interface under the 'Customization' section. On the left, a sidebar lists various settings categories: Repository, Codespaces, Planning, Copilot (selected), Access, Policies, Models, Custom instructions (selected), Knowledge bases, Custom model (with a 'Preview' button), Content exclusion, Actions (with a dropdown arrow), Webhooks, and Discussions. The 'Copilot' and 'Custom instructions' items are highlighted with blue vertical bars. The main content area has a heading 'Customization' and a sub-section 'Preferences and instructions' containing a bulleted list of preferences. Below the list is a text input field with the character count '214 / 600 characters'. At the bottom are 'Discard changes' and 'Save changes' buttons.

Code, planning, and automation

Customization

Teach Copilot your coding standards, languages, and frameworks. Keep instructions high-level, not about specific workflows. Instructions apply to all queries once saved and may not work perfectly. Press the test customization button to test.

Preferences and instructions

- Prefer writing <language> if no language is specified.
- Use <package manager> for <language> dependencies
- Prioritize <knowledge base> when asking about <topic>.
- Respond with <bullet points/minimal preamble>.

214 / 600 characters

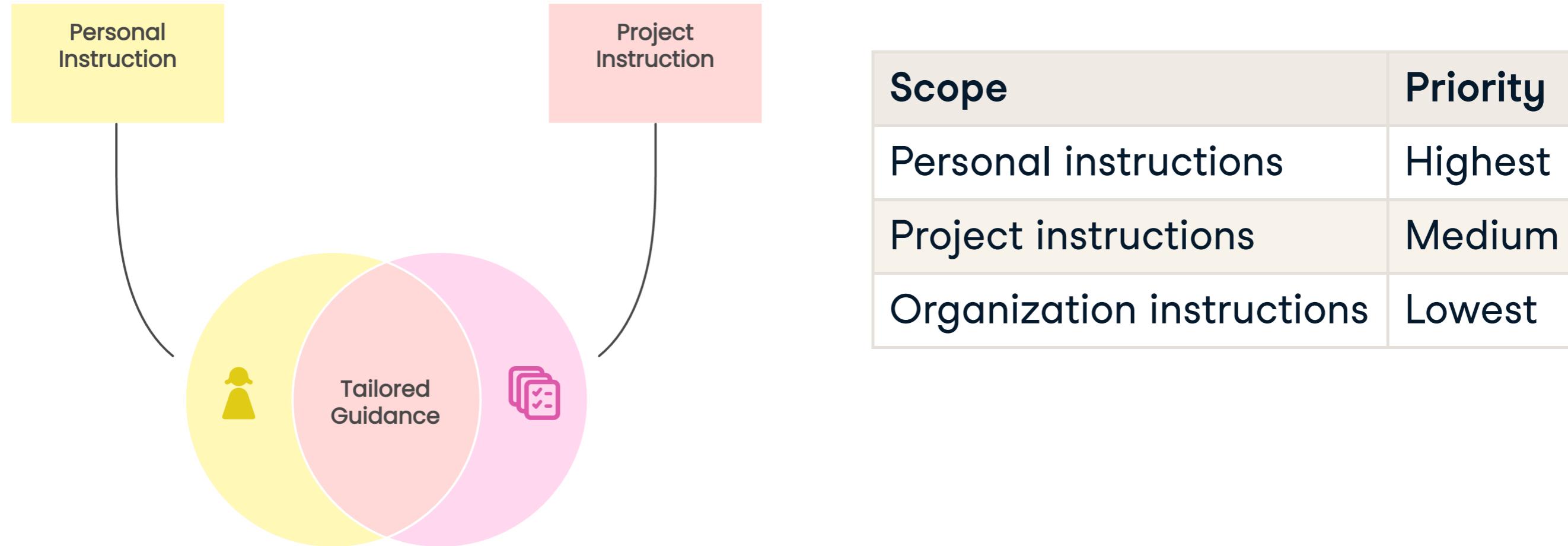
Discard changes Save changes

# Setting organization-level instructions

Examples of company-wide rules:

- Use internal tools
- Enforce specific libraries
- Apply consistent style
- Warn on deprecated patterns

# Multi-level instructions priority

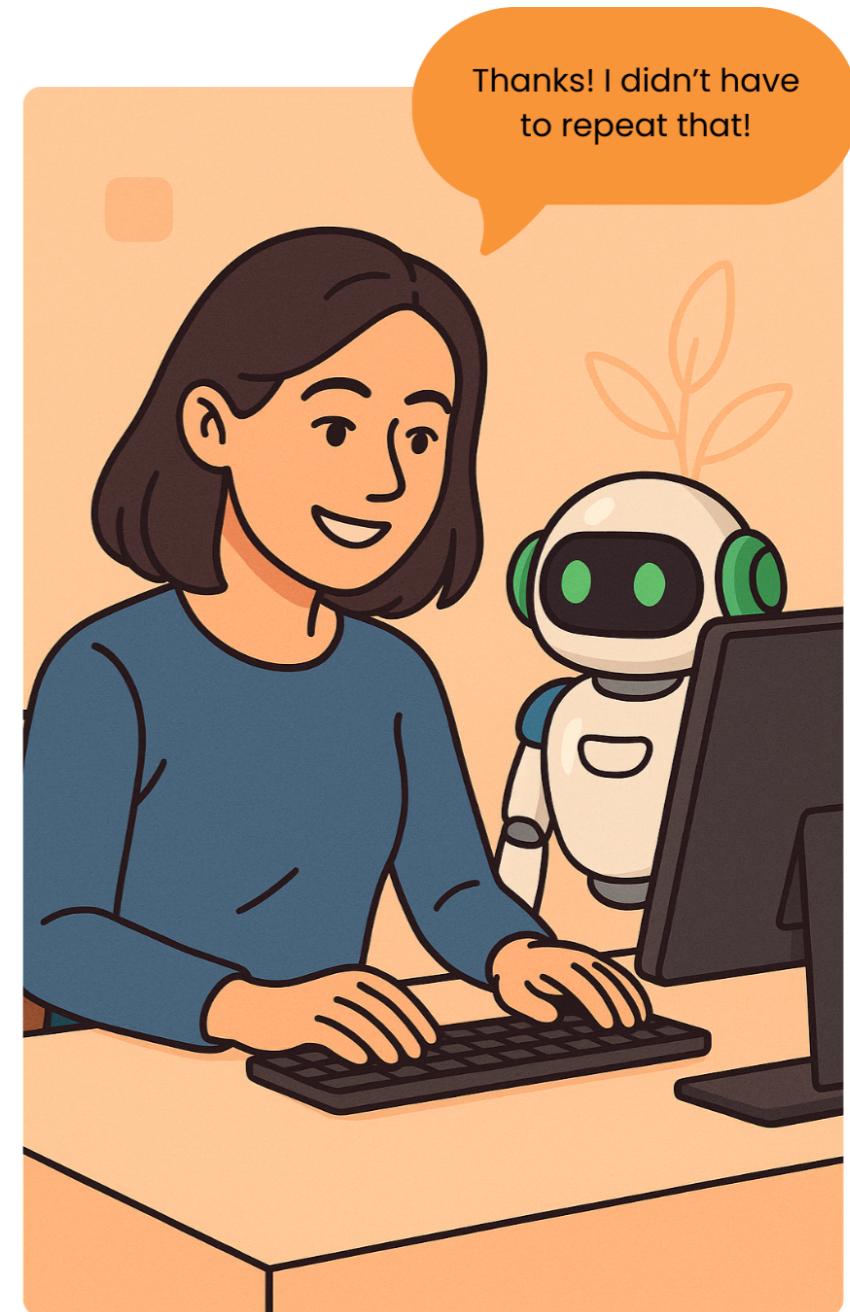


**Example prompt:** "Write a function that calculates the average of a list of numbers."

```
def calculate_average(numbers: List[float]) -> float:  
    """Calculate the average of a list of numbers.  
  
    Args:  
        numbers (List[float]): A list of numbers to average.  
  
    Returns:  
        float: The average value of the numbers.  
  
    Raises:  
        ValueError: If the list is empty.  
    """  
  
    if not numbers:  
        raise ValueError("The list of numbers cannot be empty.  
    return sum(numbers) / len(numbers)
```

**Example prompt:** "Write a function that calculates the average of a list of numbers."

```
def calculate_average(numbers: List[float]) -> float:  
    """Calculate the average of a list of numbers.  
  
    Args:  
        numbers (List[float]): A list of numbers to average.  
  
    Returns:  
        float: The average value of the numbers.  
  
    Raises:  
        ValueError: If the list is empty.  
    """  
  
    if not numbers:  
        raise ValueError("The numbers list cannot be empty.")  
    return sum(numbers) / len(numbers)
```



# **Let's practice!**

**SOFTWARE DEVELOPMENT WITH GITHUB COPILOT**

# Selecting and switching models

SOFTWARE DEVELOPMENT WITH GITHUB COPILOT



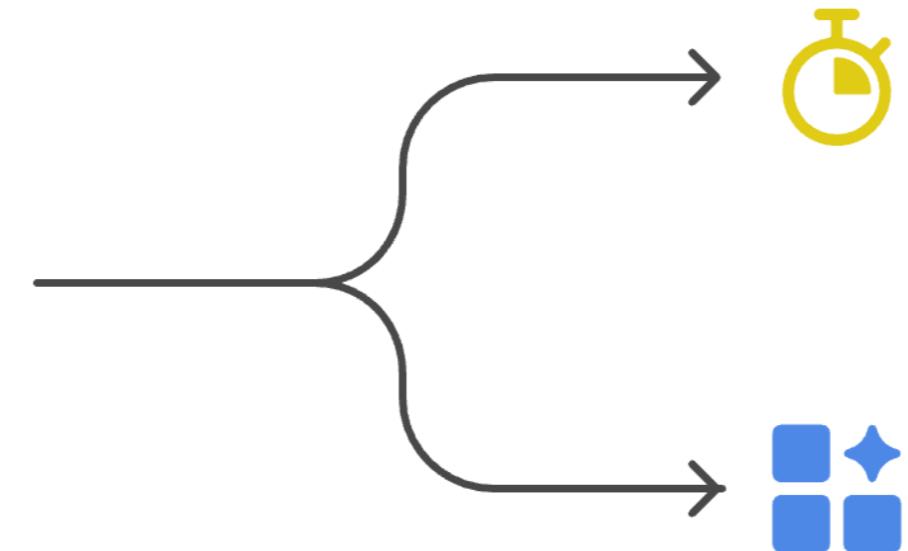
Thalia Barrera

AI Engineering Curriculum Manager,  
DataCamp

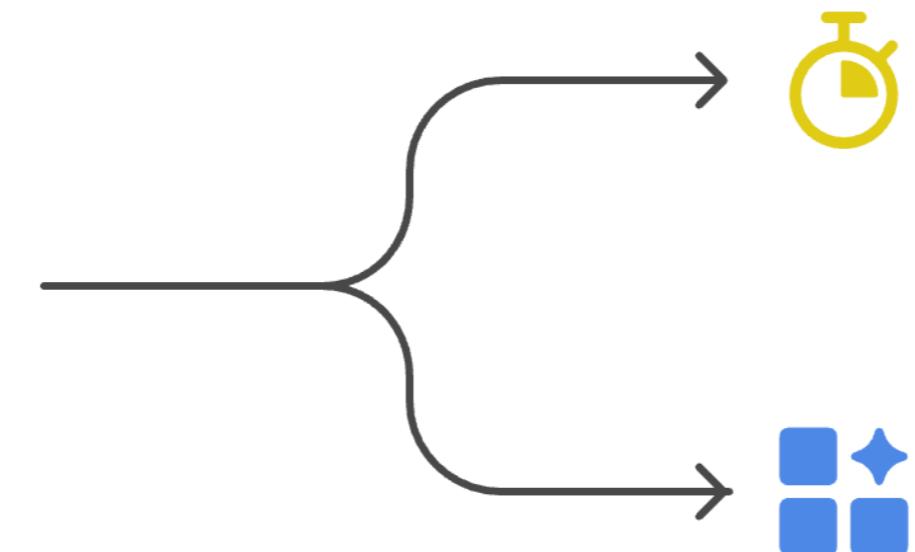




Choosing the  
right model for  
the task



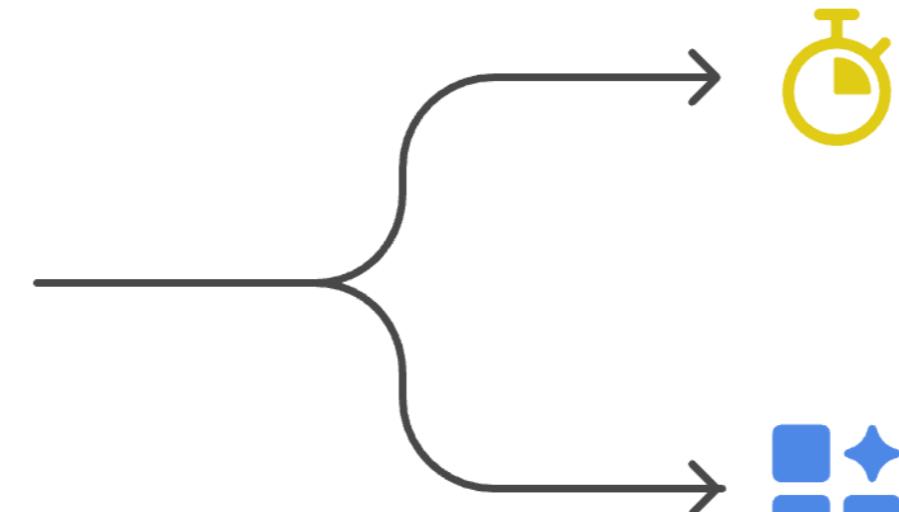
 Choosing the right model for the task



 **Quick Response Models**

Ideal for autocomplete

## Choosing the right model for the task



### Quick Response Models

Ideal for autocomplete



### Reasoning Models

Best for refactoring, debugging or planning

# Not all models are created equal

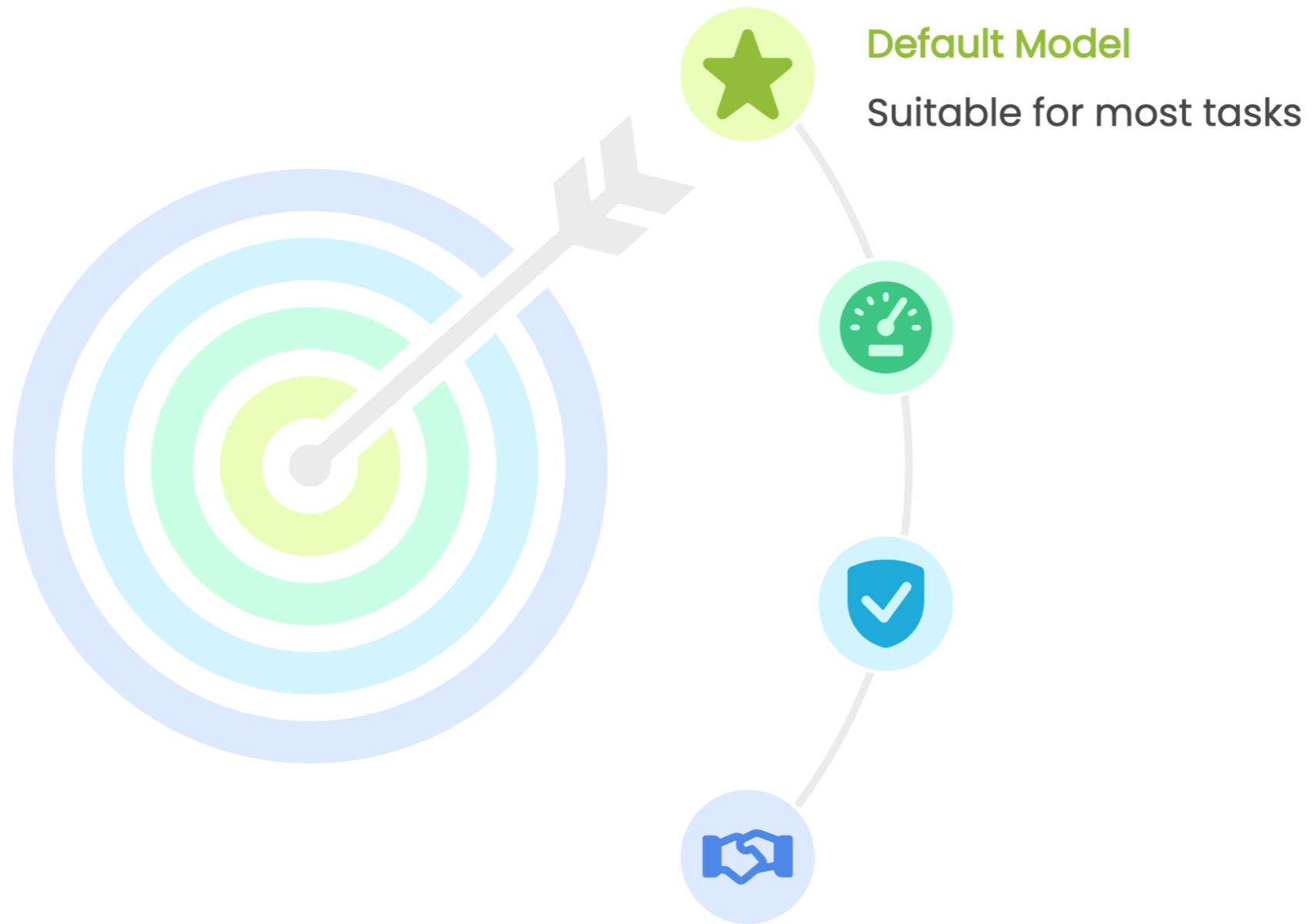
Model name	Provider	Release status	Agent mode	Ask mode	Edit mode
GPT-4.1	OpenAI	GA	✓	✓	✓
GPT-5-Codex	OpenAI	Public preview	✓	✓	✓
GPT-5 mini	OpenAI	GA	✓	✓	✓
GPT-5	OpenAI	GA	✓	✓	✓
o3	OpenAI	Closing down: 2025-10-23	✗	✓	✓
o4-mini	OpenAI	Closing down: 2025-10-23	✓	✓	✓
Claude Opus 4.1	Anthropic	GA	✓	✓	✓
Claude Opus 4	Anthropic	Closing down: 2025-10-23	✗	✓	✓

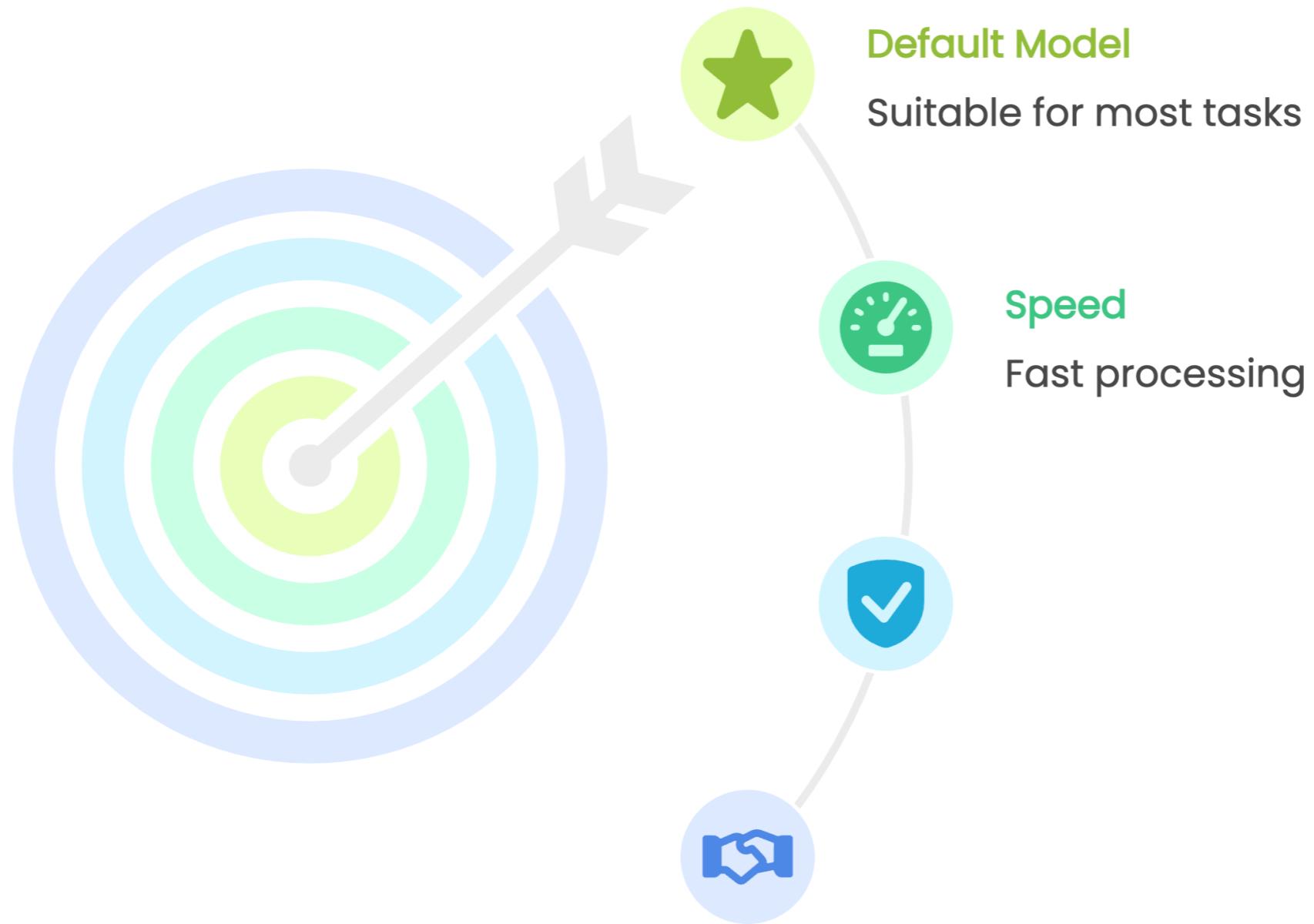
<sup>1</sup> <https://docs.github.com/en/copilot/reference/ai-models/supported-models>

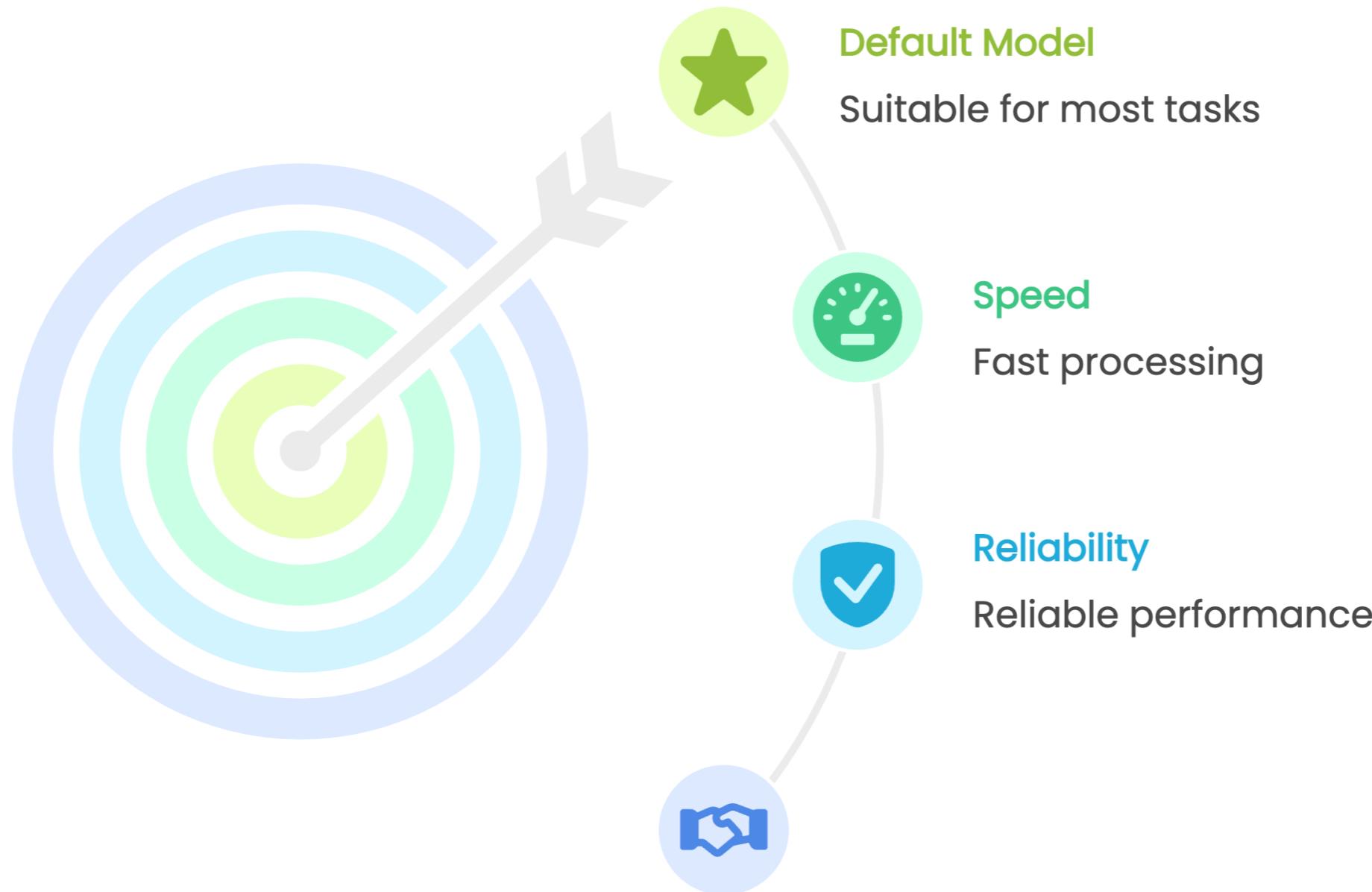
# Not all models are created equal

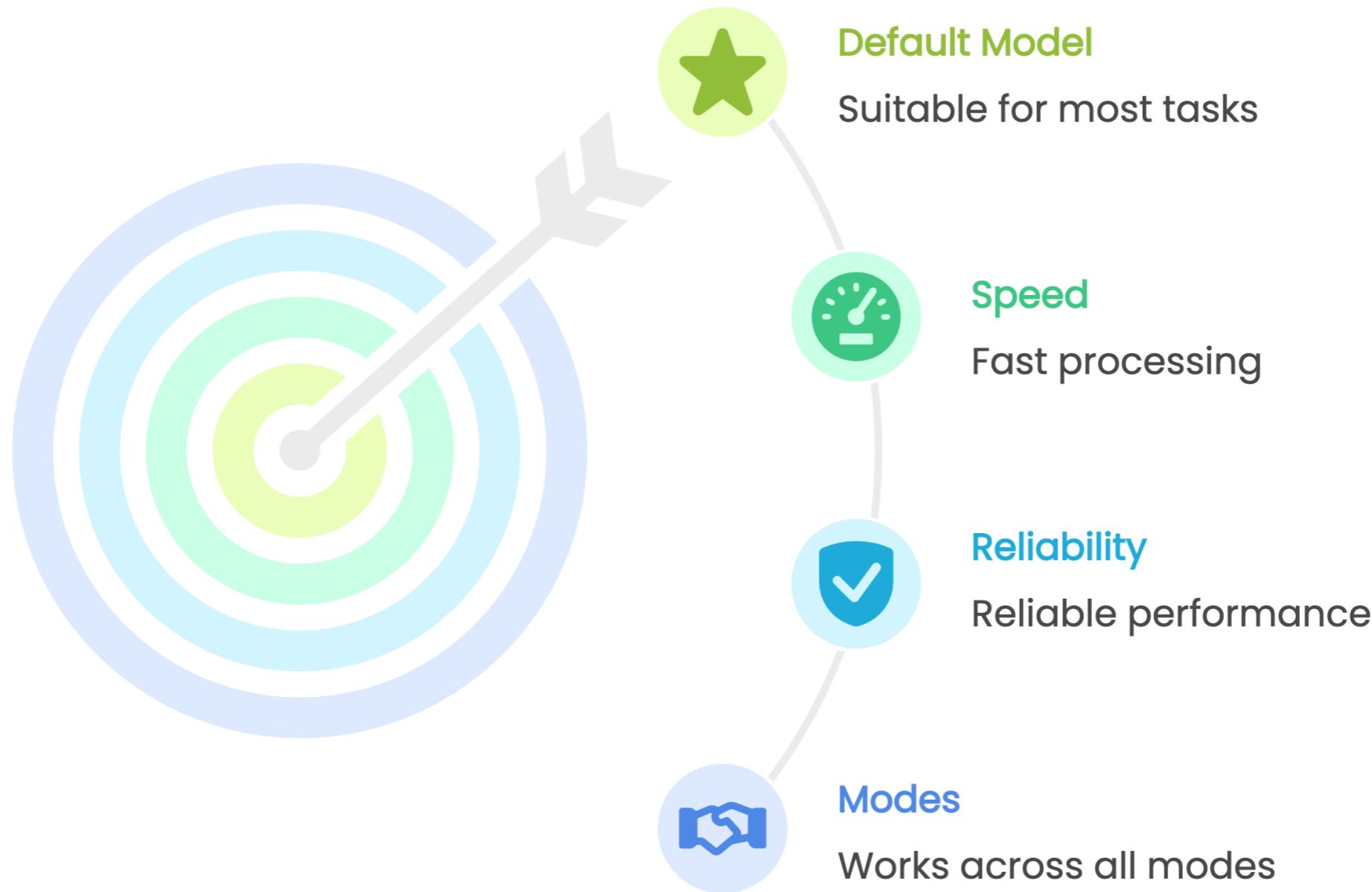
Model name	Provider	Release status	Agent mode	Ask mode	Edit mode
GPT-4.1	OpenAI	GA	✓	✓	✓
GPT-5-Codex	OpenAI	Public preview	✓	✓	✓
GPT-5 mini	OpenAI	GA	✓	✓	✓
GPT-5	OpenAI	GA	✓	✓	✓
o3	OpenAI	Closing down: 2025-10-23	✗	✓	✓
o4-mini	OpenAI	Closing down: 2025-10-23	✓	✓	✓
Claude Opus 4.1	Anthropic	GA	✓	✓	✓
Claude Opus 4	Anthropic	Closing down: 2025-10-23	✗	✓	✓

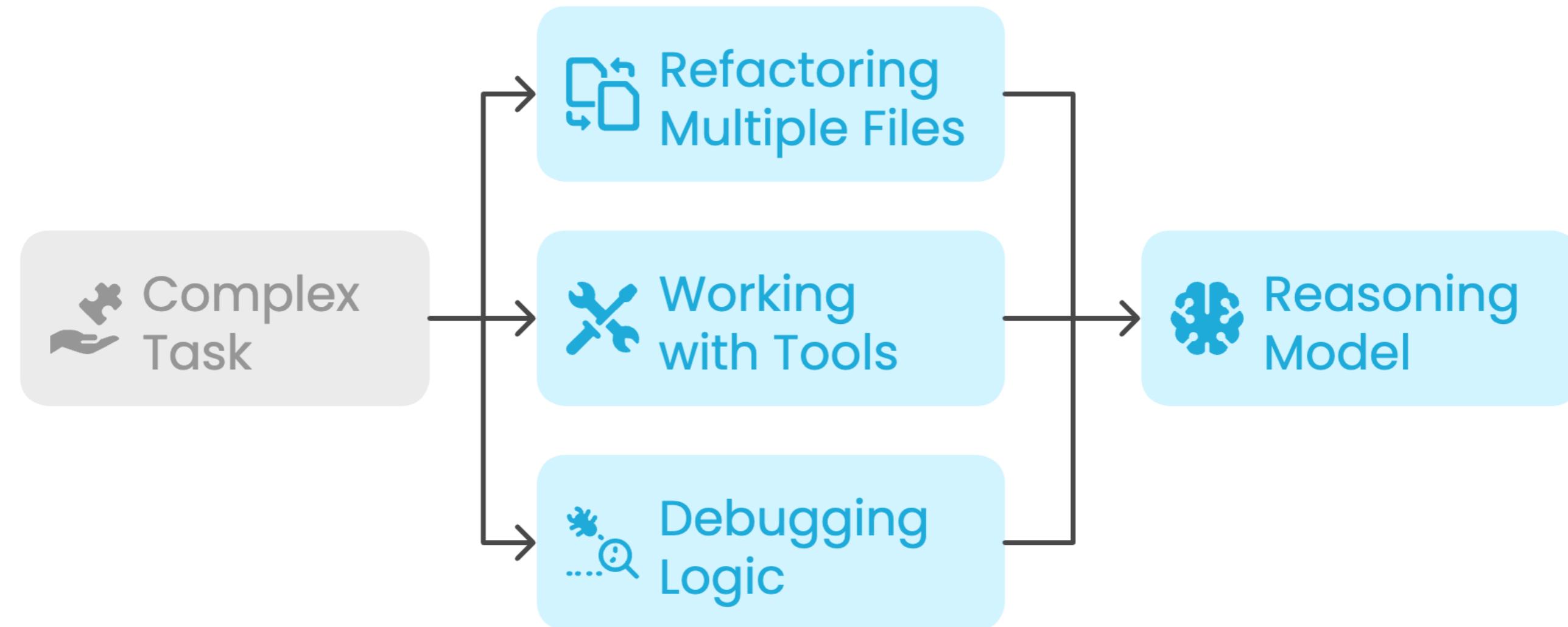
<sup>1</sup> <https://docs.github.com/en/copilot/reference/ai-models/supported-models>







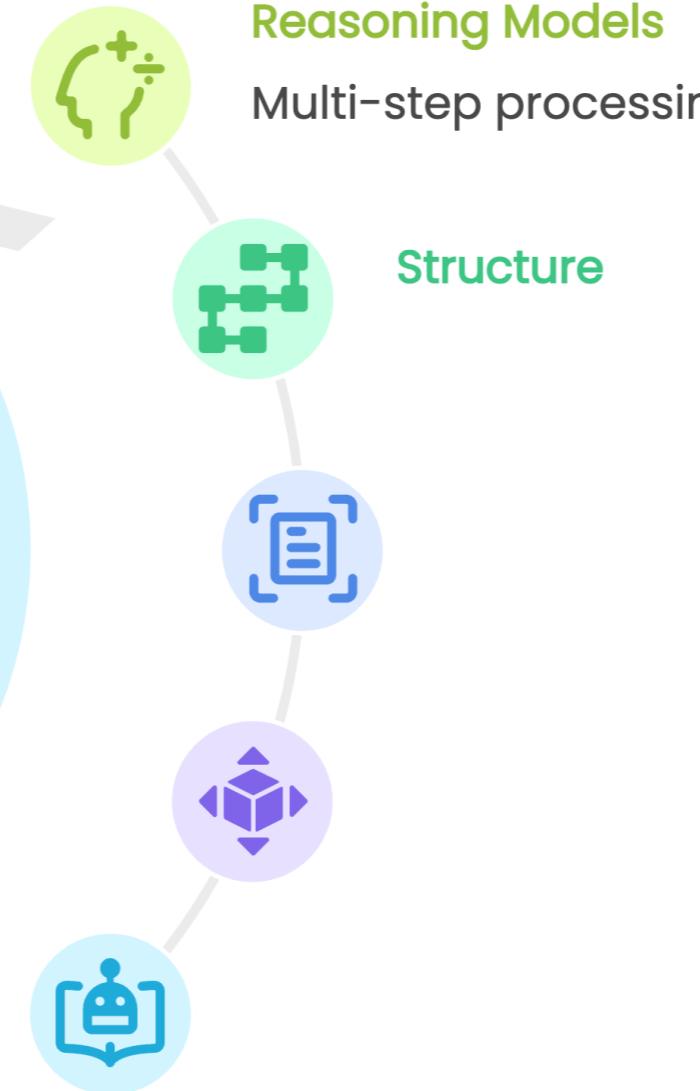






**Reasoning Models**  
Multi-step processing







**Reasoning Models**  
Multi-step processing

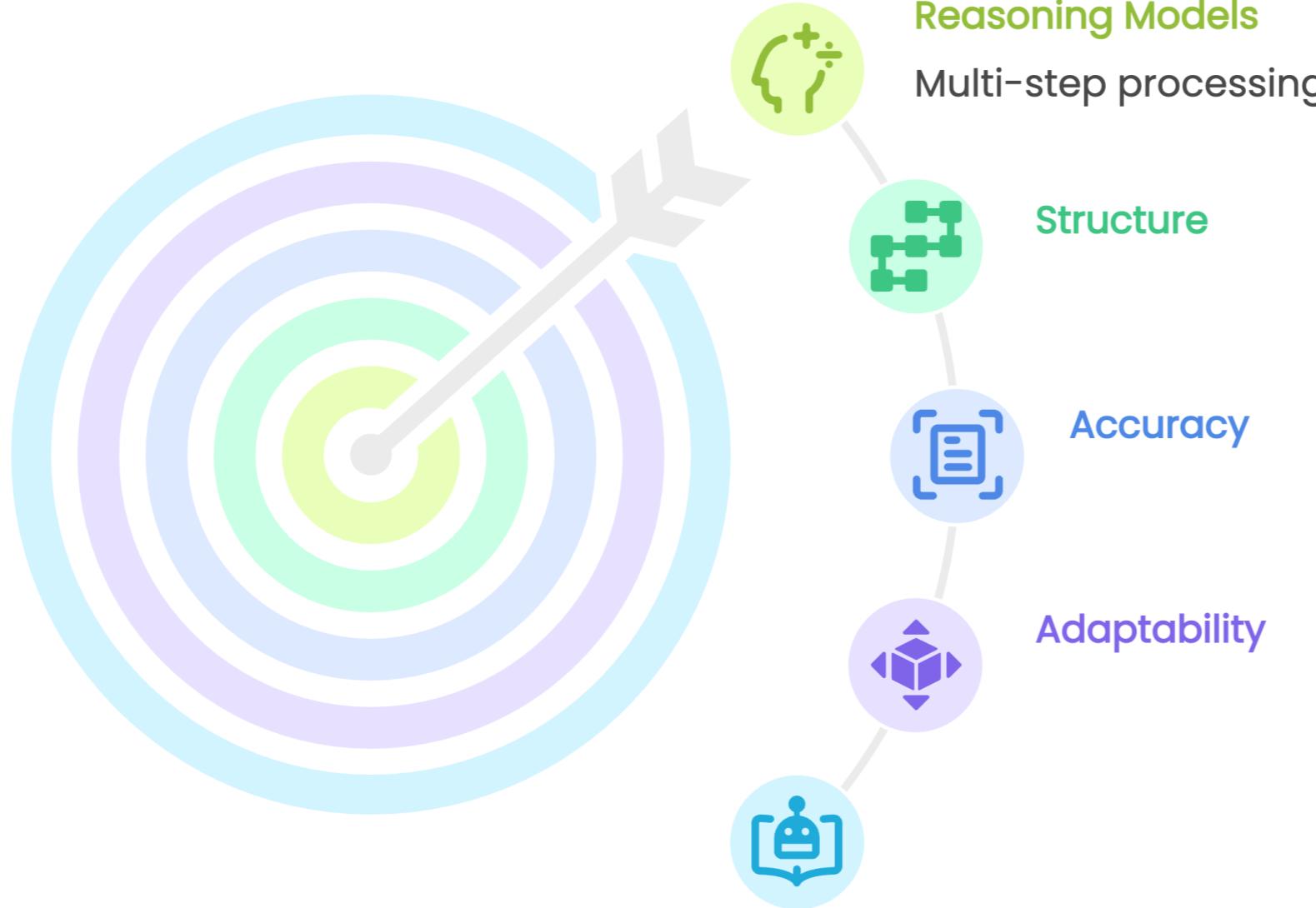


**Structure**



**Accuracy**





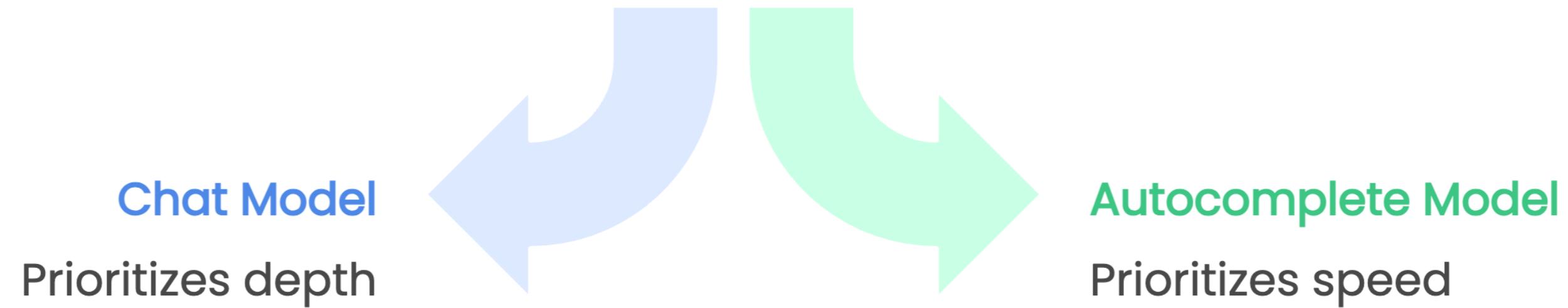


# Choosing the right model for the interface

**Chat Model**  
Prioritizes depth



# Choosing the right model for the interface





# Bring your own API key

- Test newer models
- Use local or private deployments
- Bypass rate limits
- Explore advanced use cases

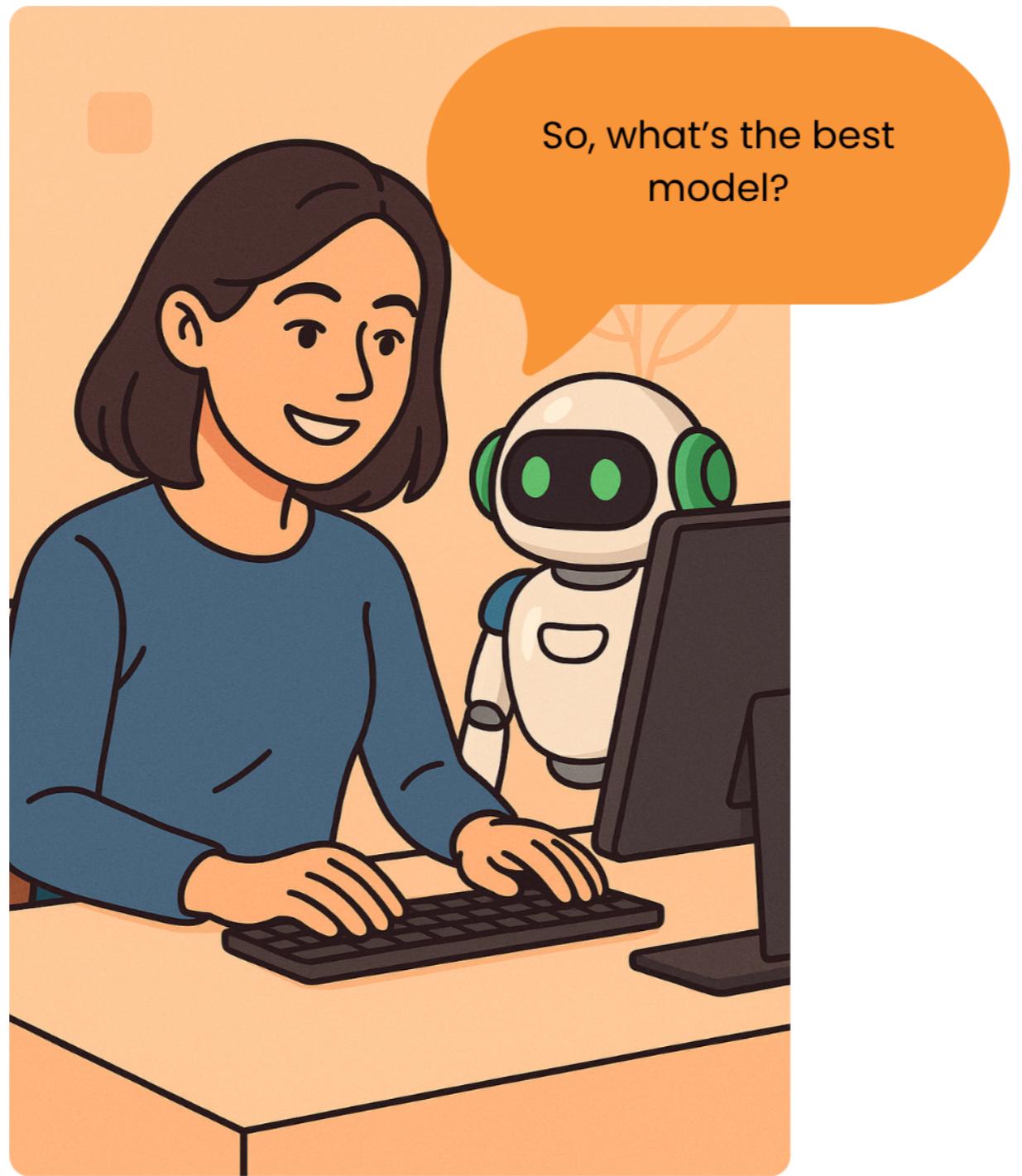
**Note:** Only available to individual users

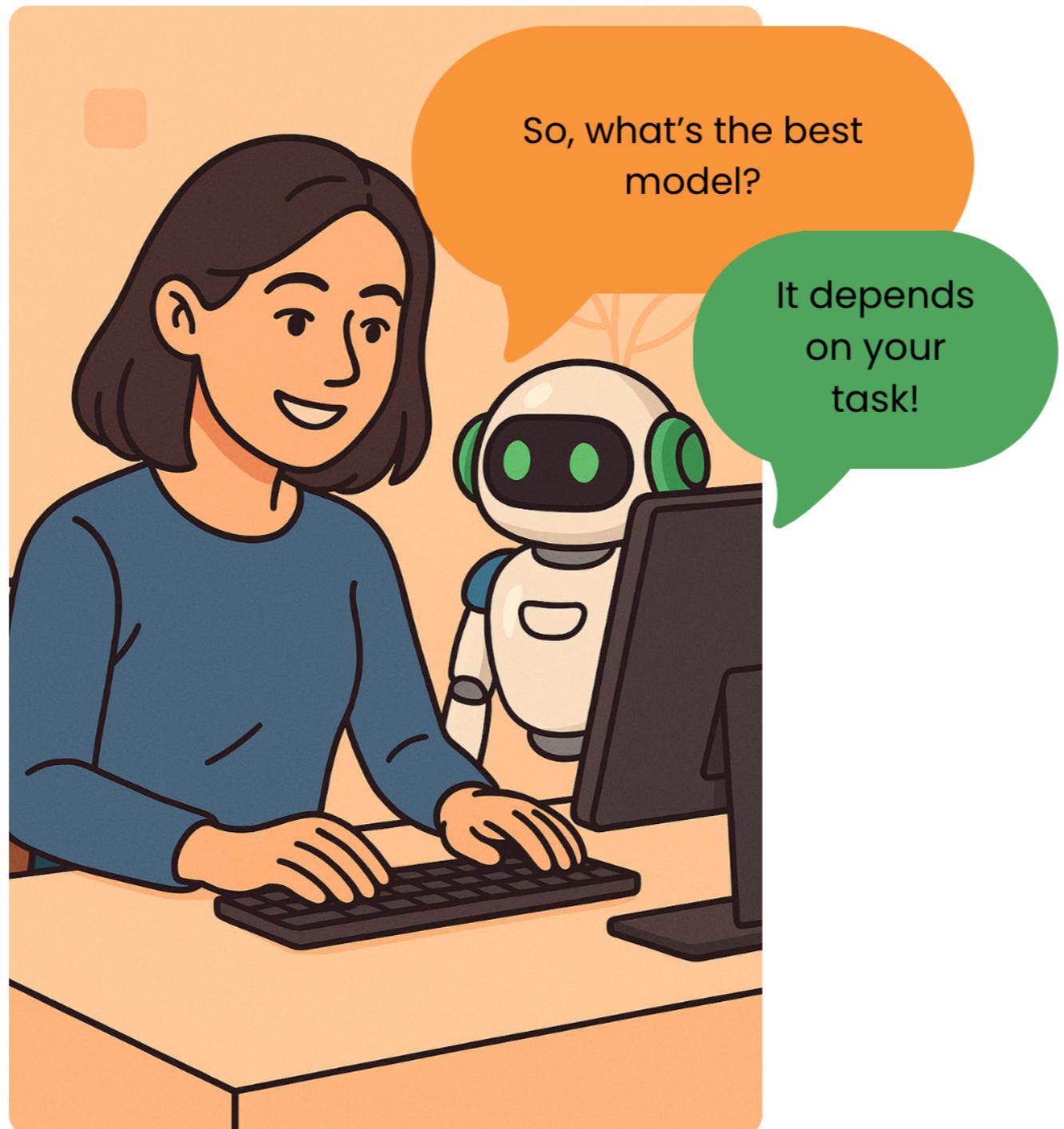


# Bring your own API key: limitations

- Only applies to **chat** (not completions or commit messages)
- May lack **features**: tool support, vision, syntax awareness
- Copilot relies on its own systems for **indexing**







# **Let's practice!**

**SOFTWARE DEVELOPMENT WITH GITHUB COPILOT**

# Extending Copilot with MCP tools

SOFTWARE DEVELOPMENT WITH GITHUB COPILOT



Thalia Barrera

AI Engineering Curriculum Manager,  
DataCamp



What is MCP?



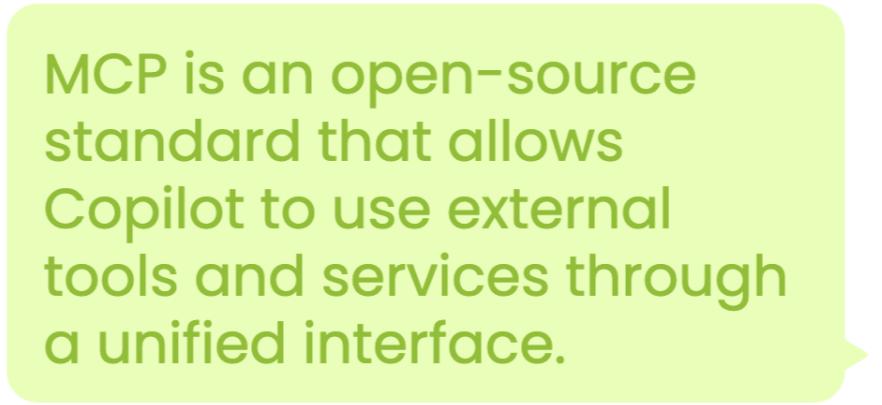
### What is MCP?

MCP is an open-source standard that allows Copilot to use external tools and services through a unified interface.





What is MCP?

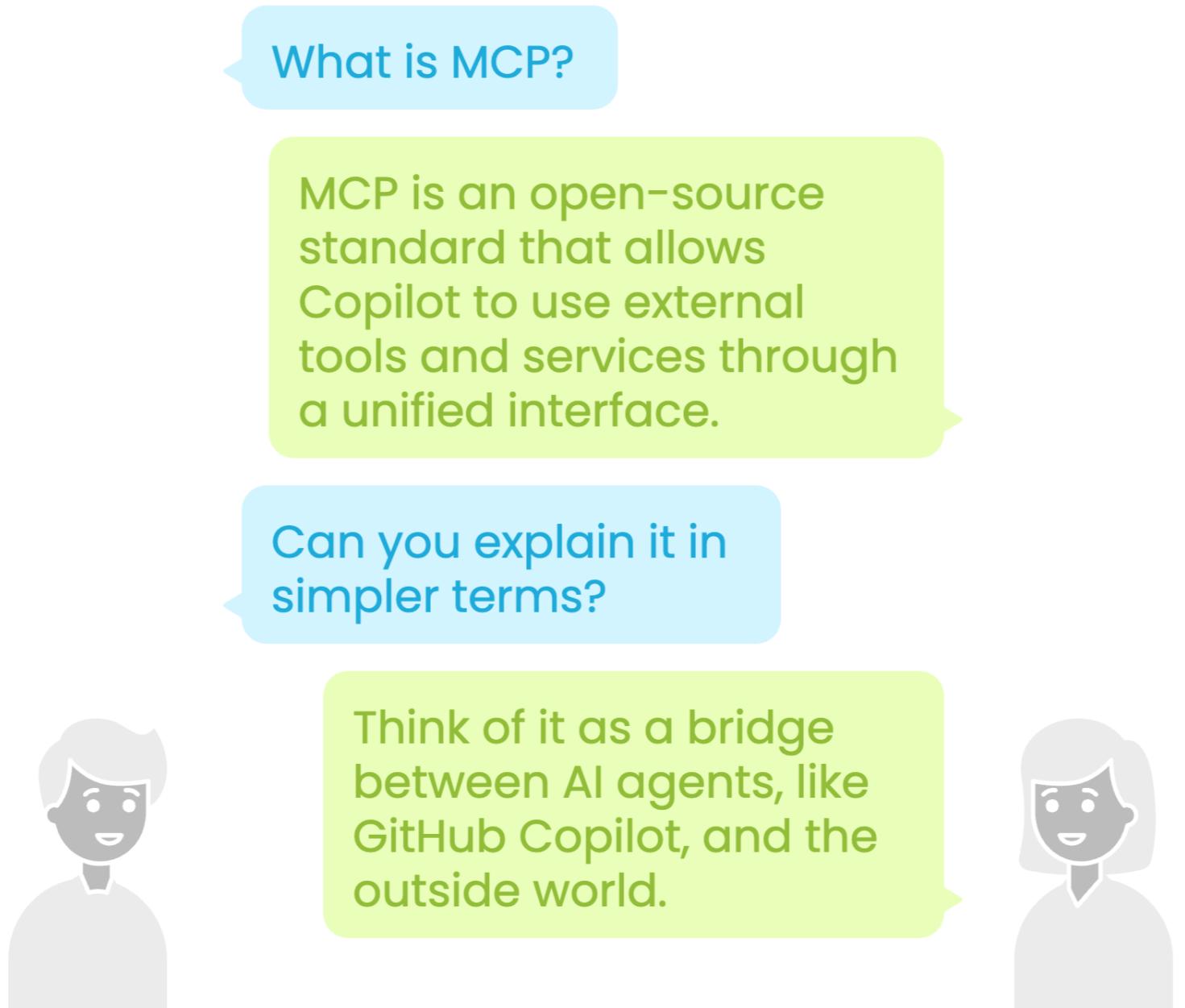


MCP is an open-source standard that allows Copilot to use external tools and services through a unified interface.



Can you explain it in simpler terms?





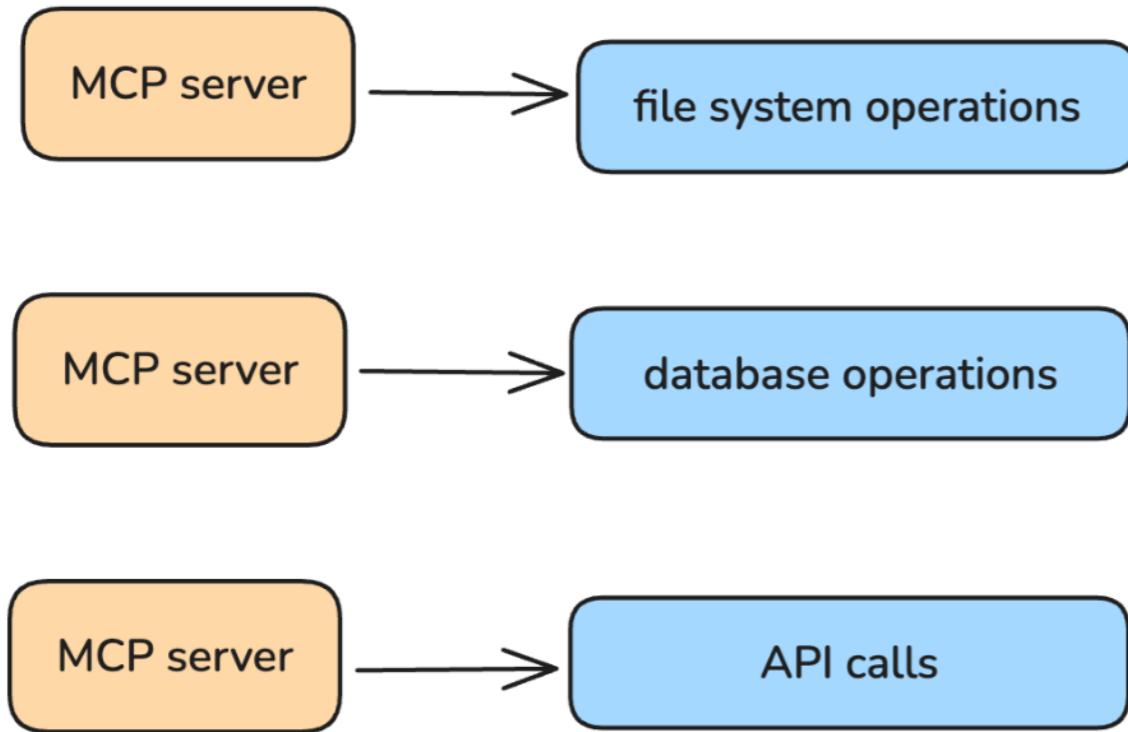
What is MCP?

MCP is an open-source standard that allows Copilot to use external tools and services through a unified interface.

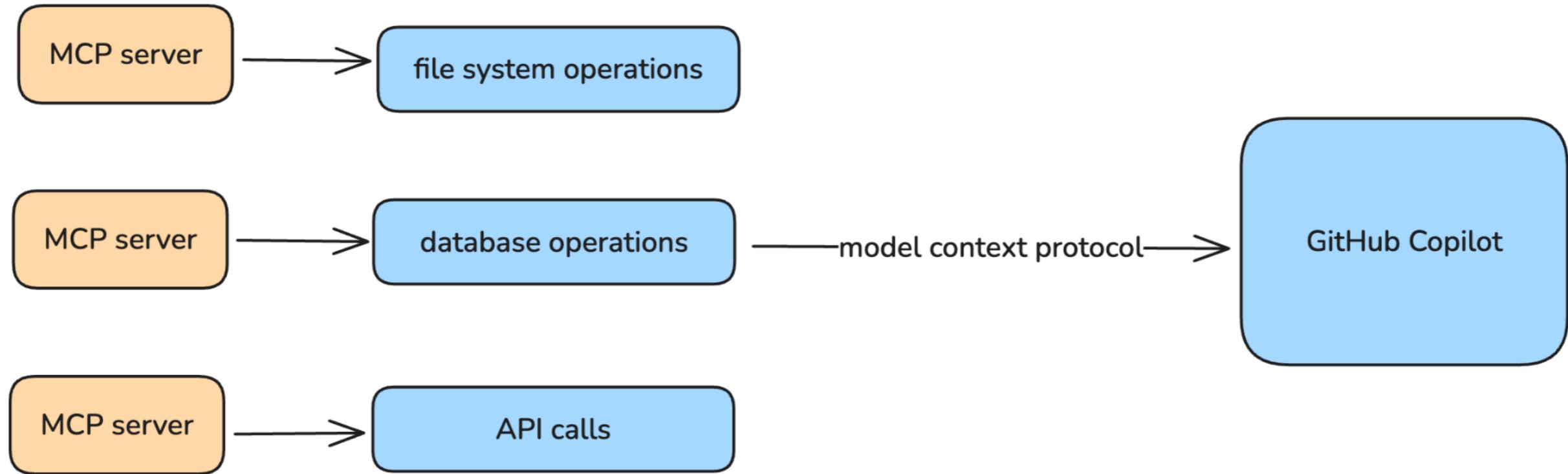
Can you explain it in simpler terms?

Think of it as a bridge between AI agents, like GitHub Copilot, and the outside world.

# How MCP servers work



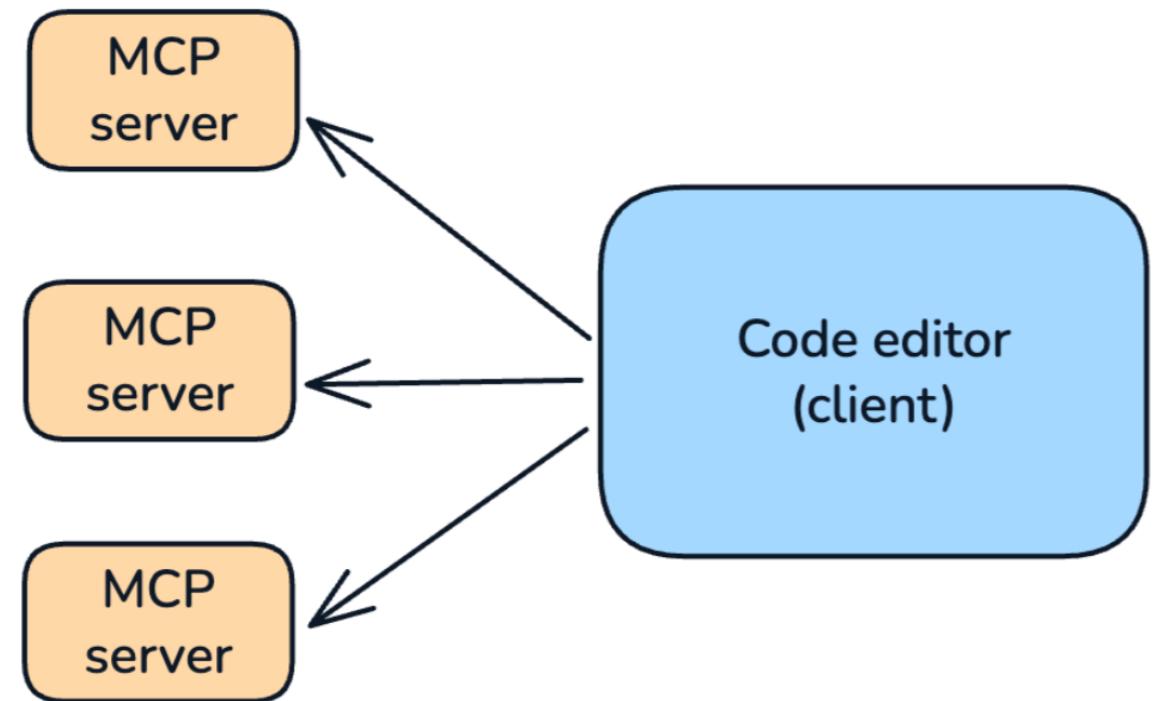
# How MCP servers work



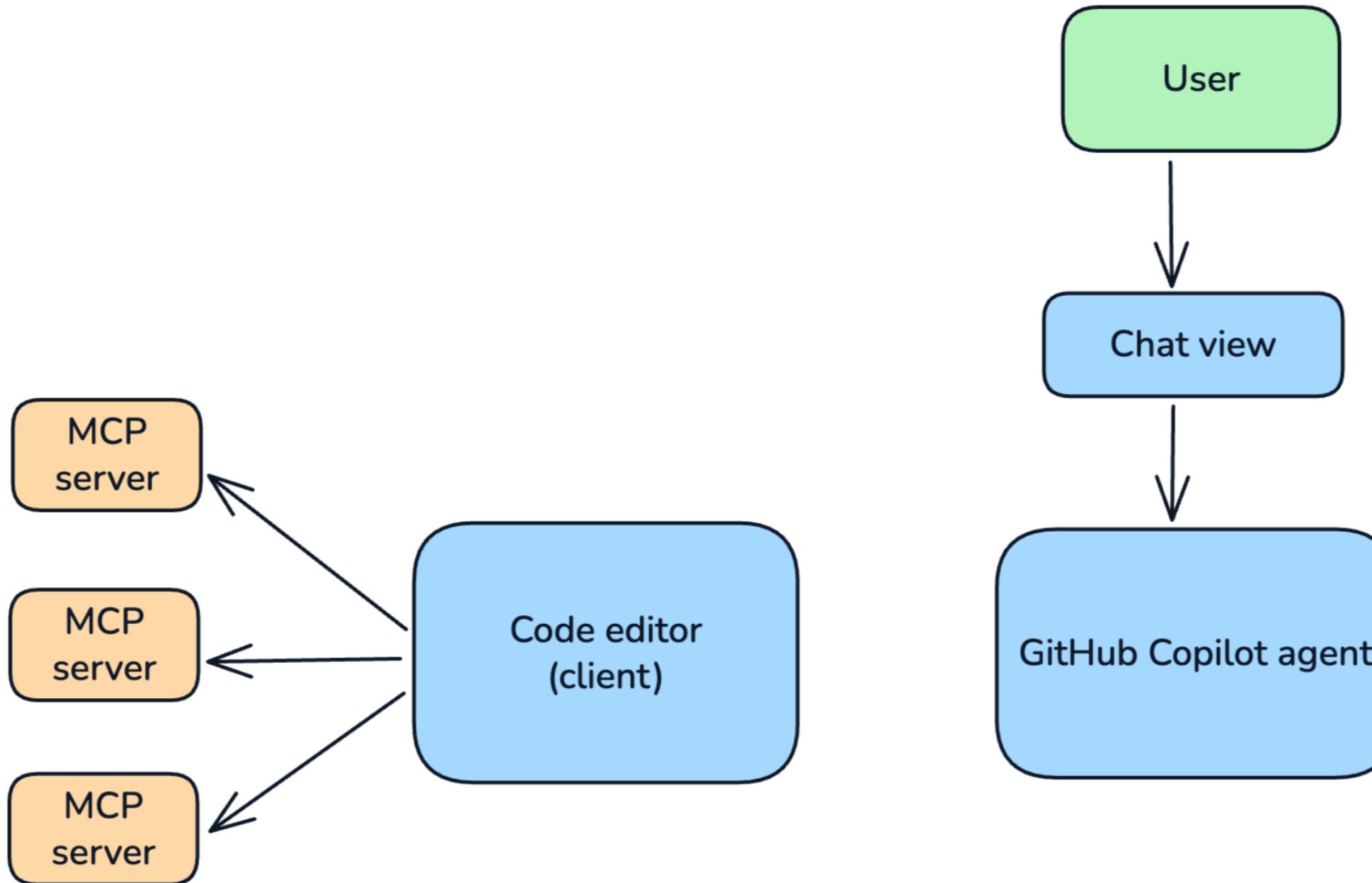
# How the MCP works with GitHub Copilot



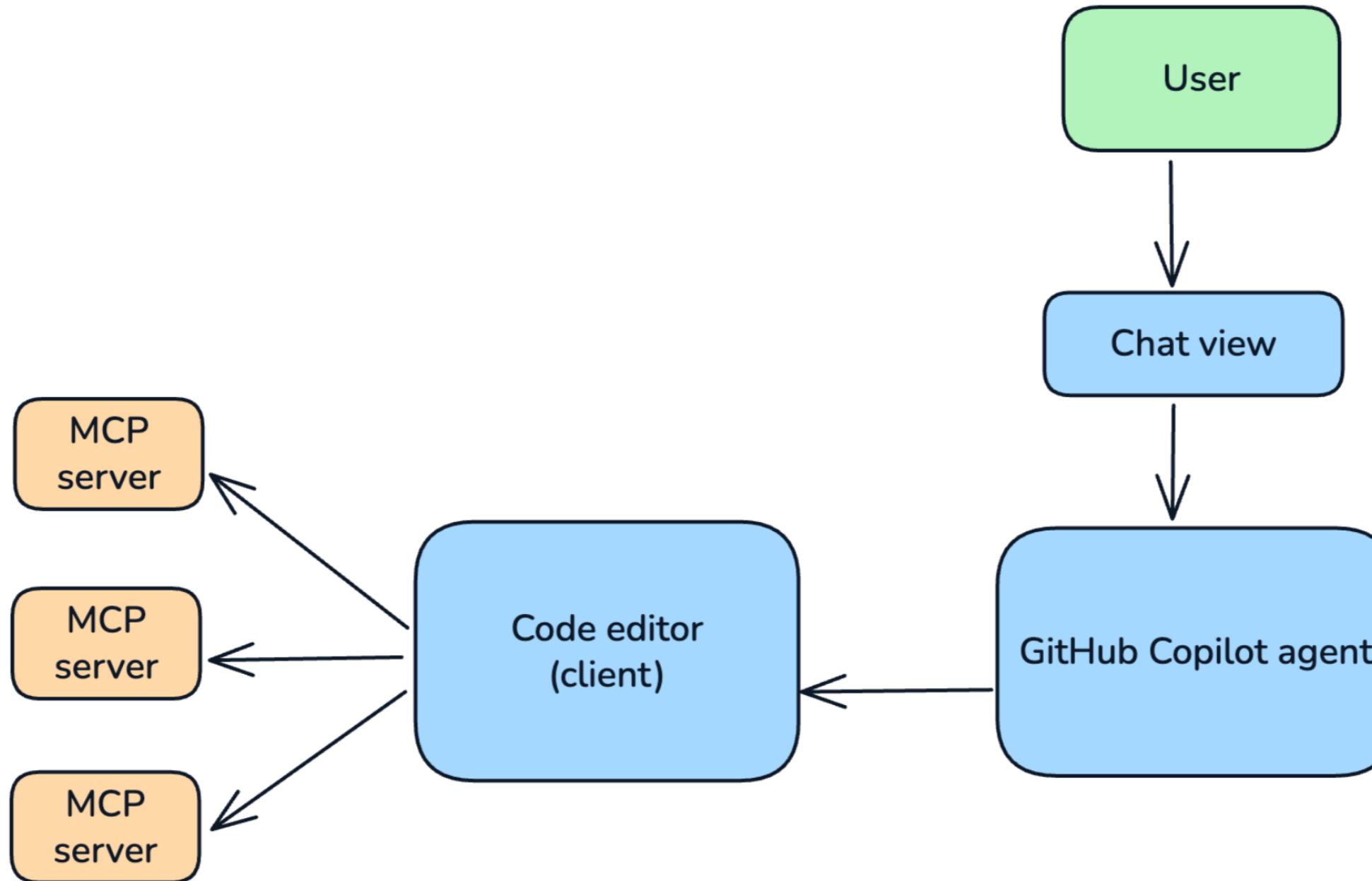
# How the MCP works with GitHub Copilot



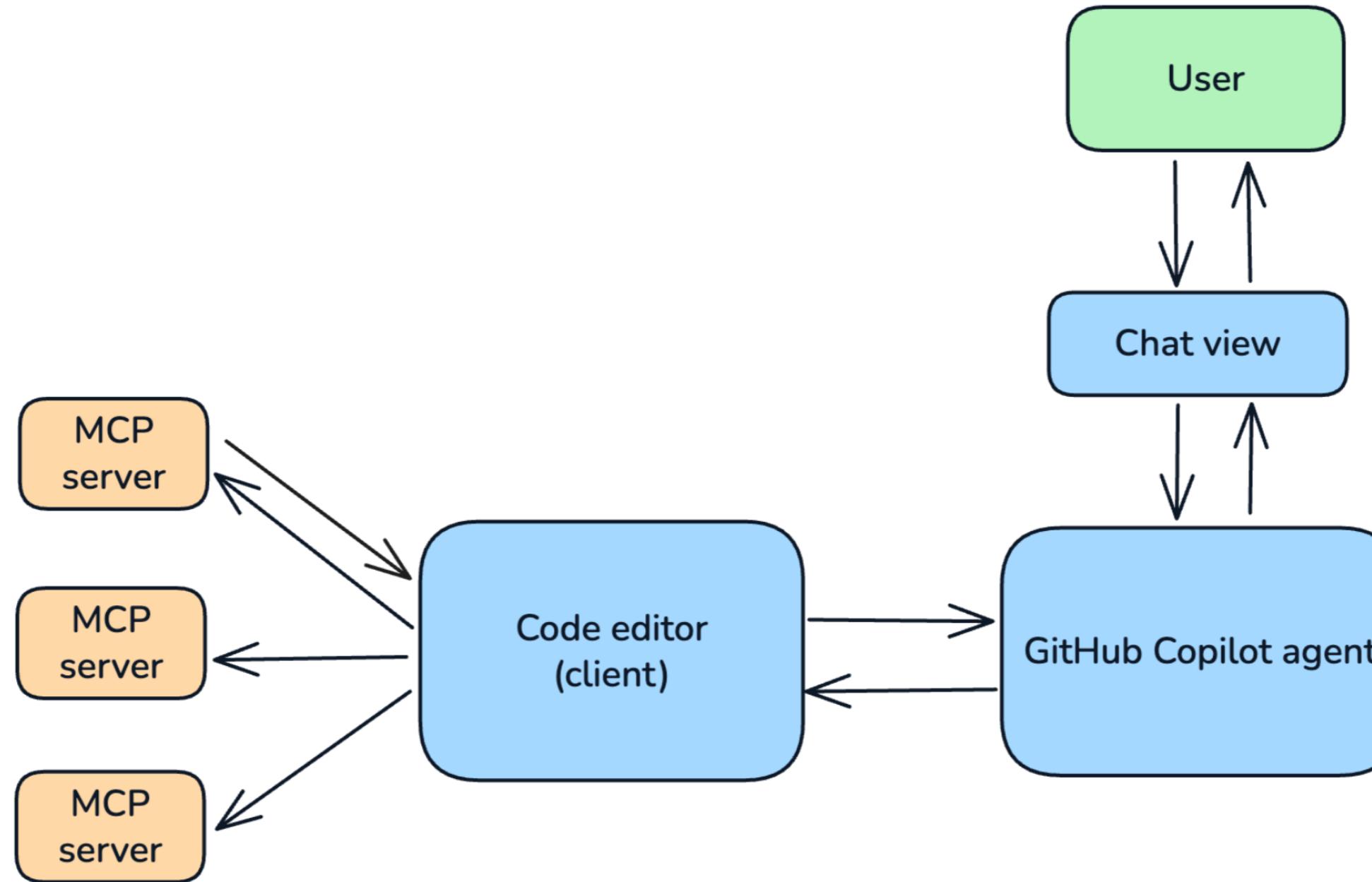
# How the MCP works with GitHub Copilot

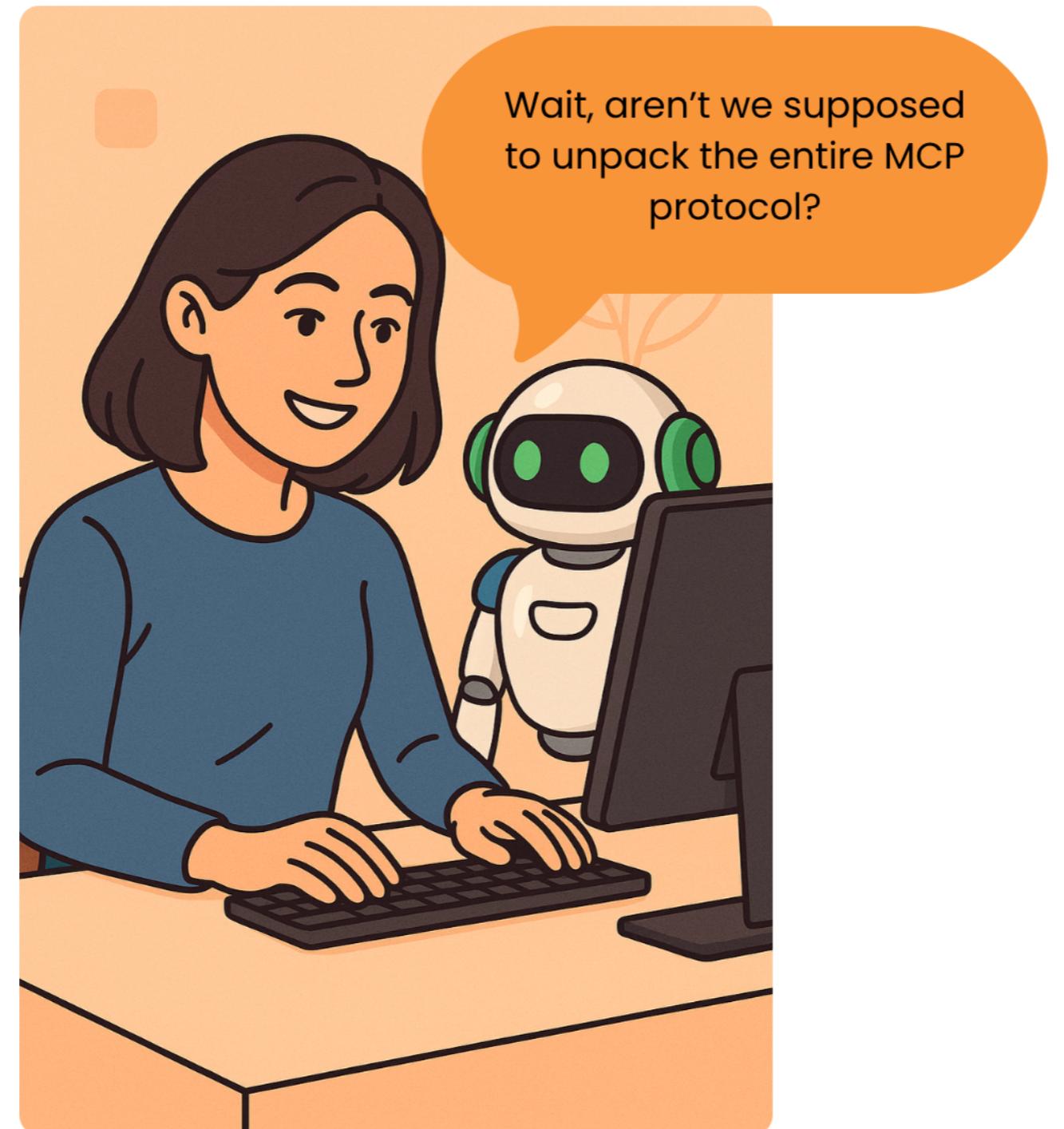


# How the MCP works with GitHub Copilot



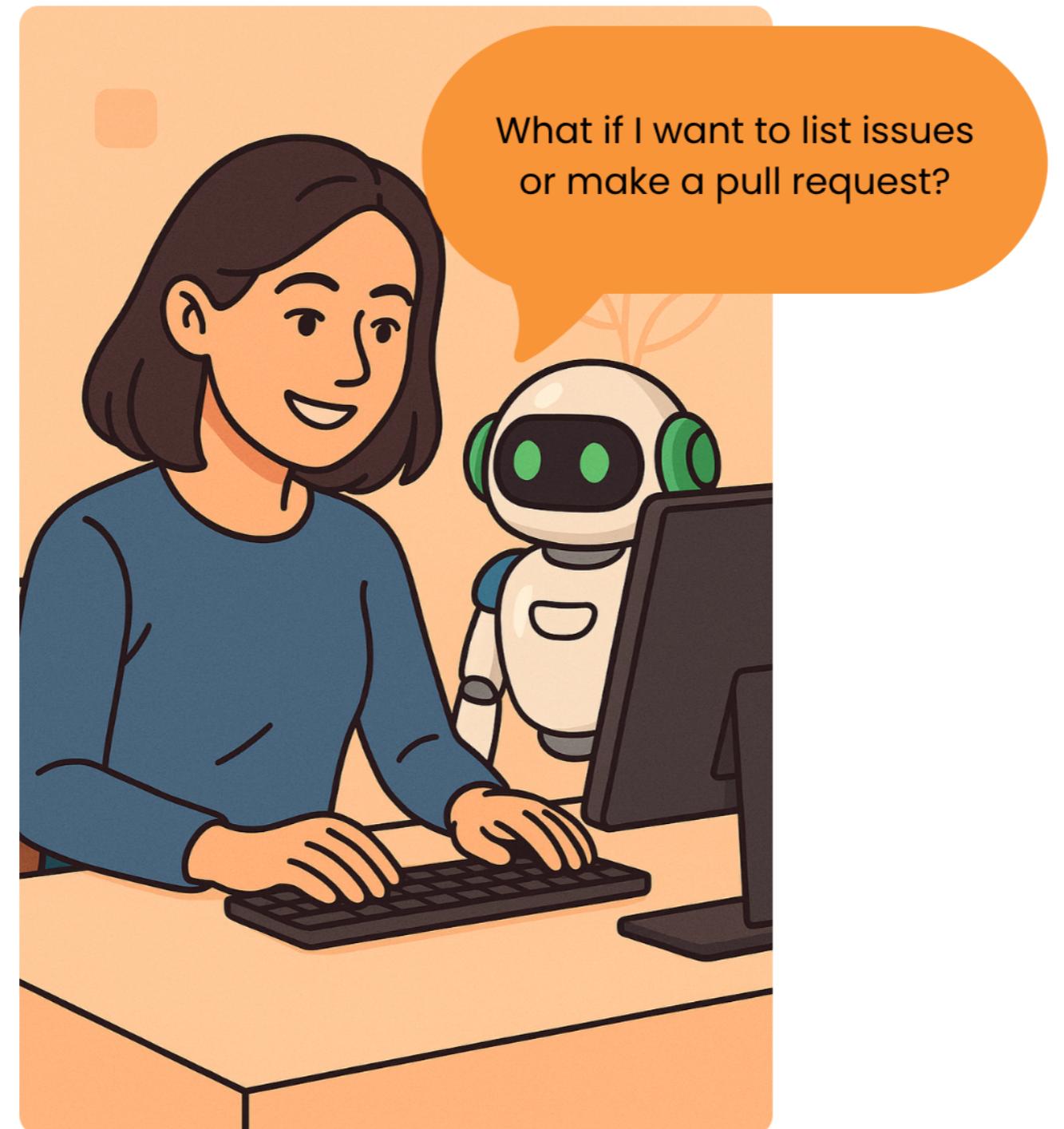
# How the MCP works with GitHub Copilot

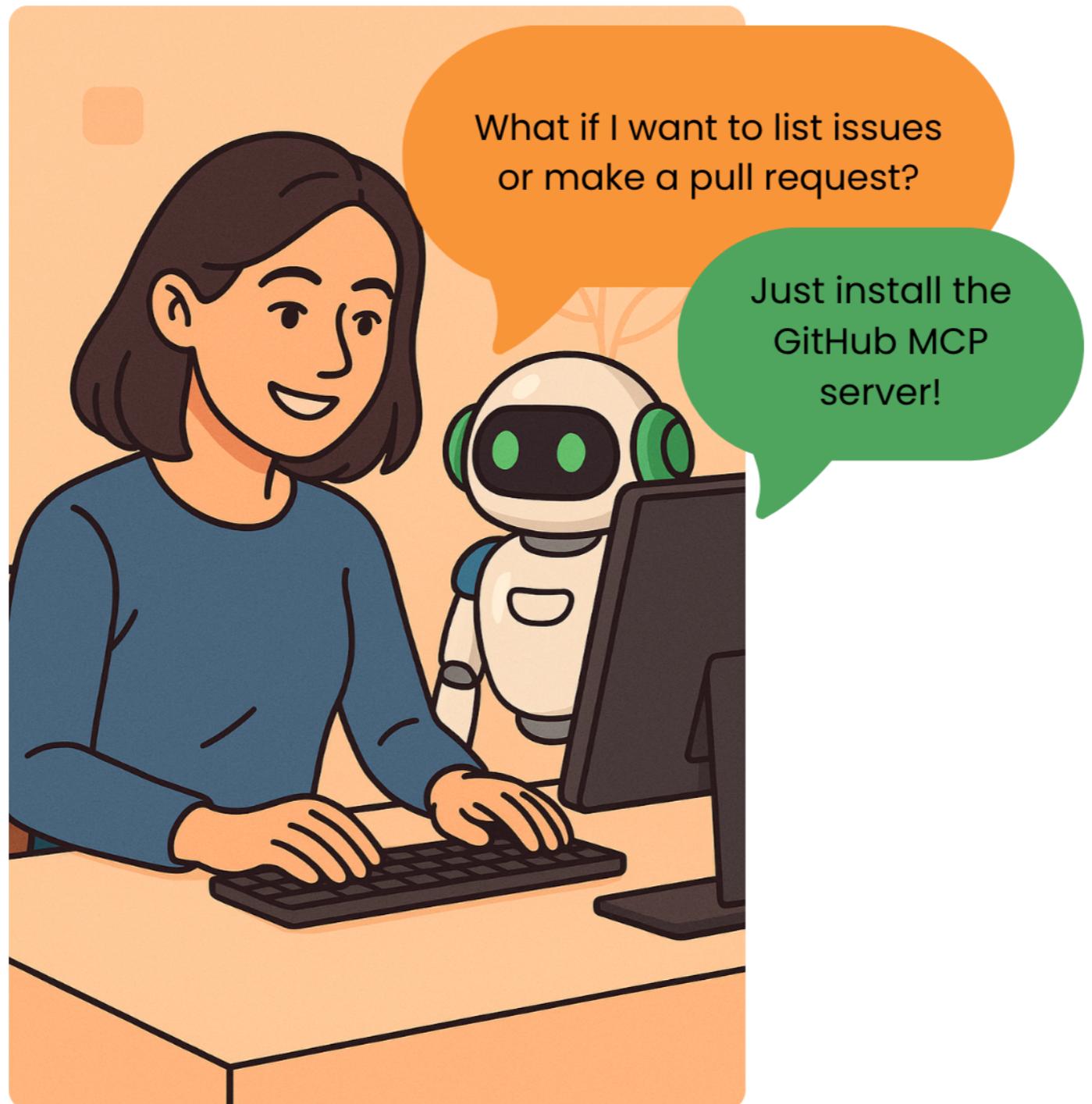




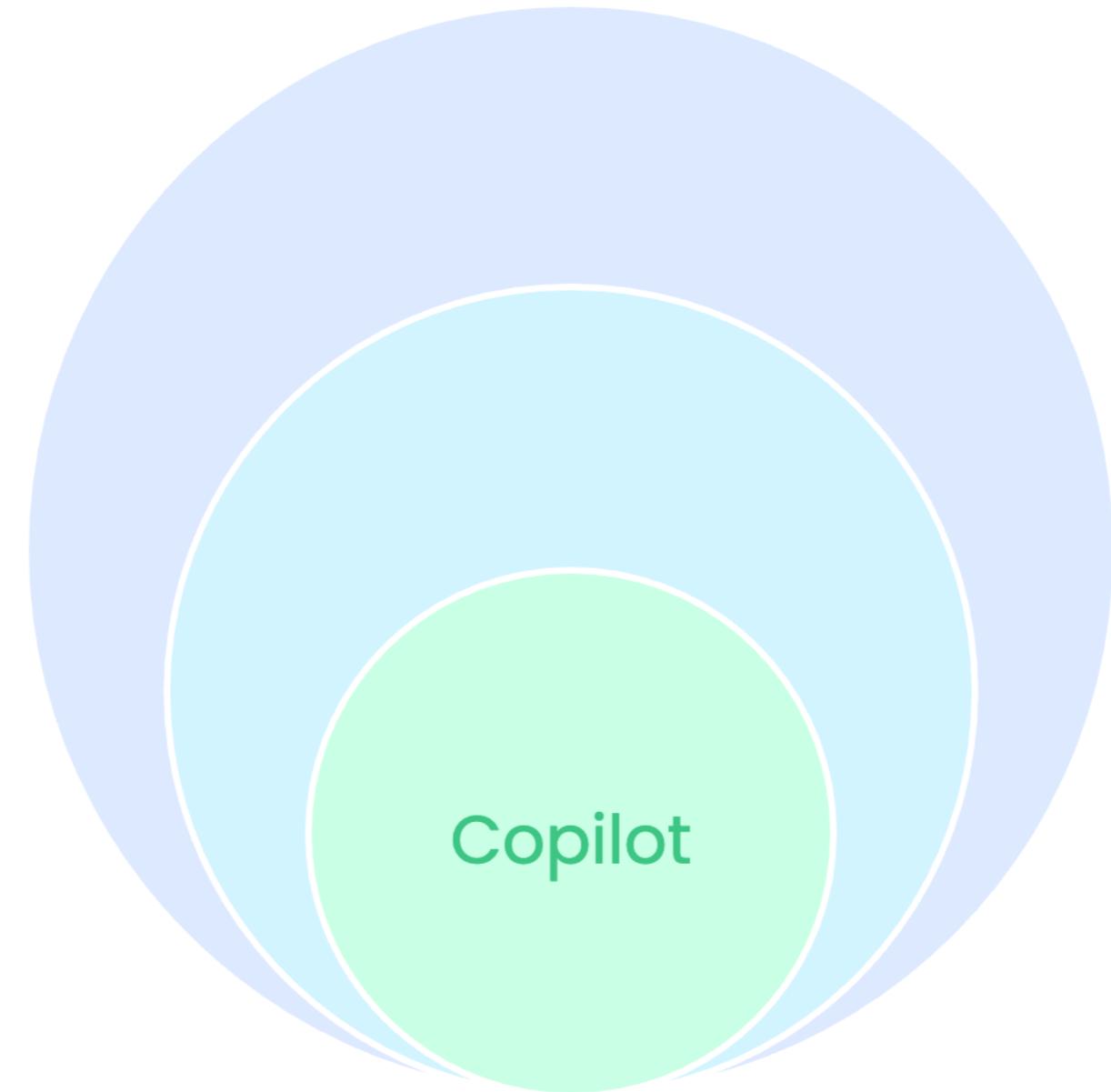




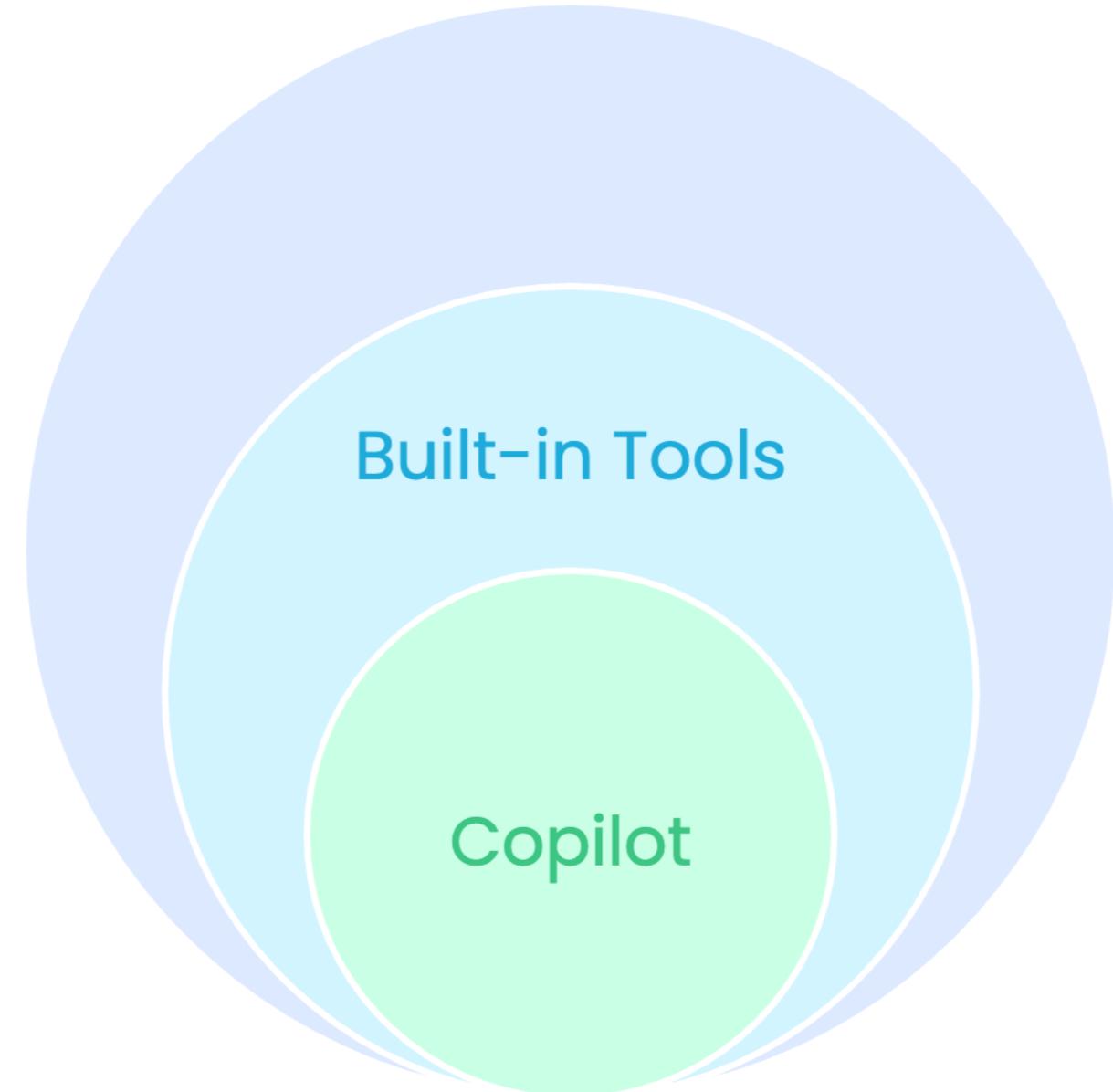




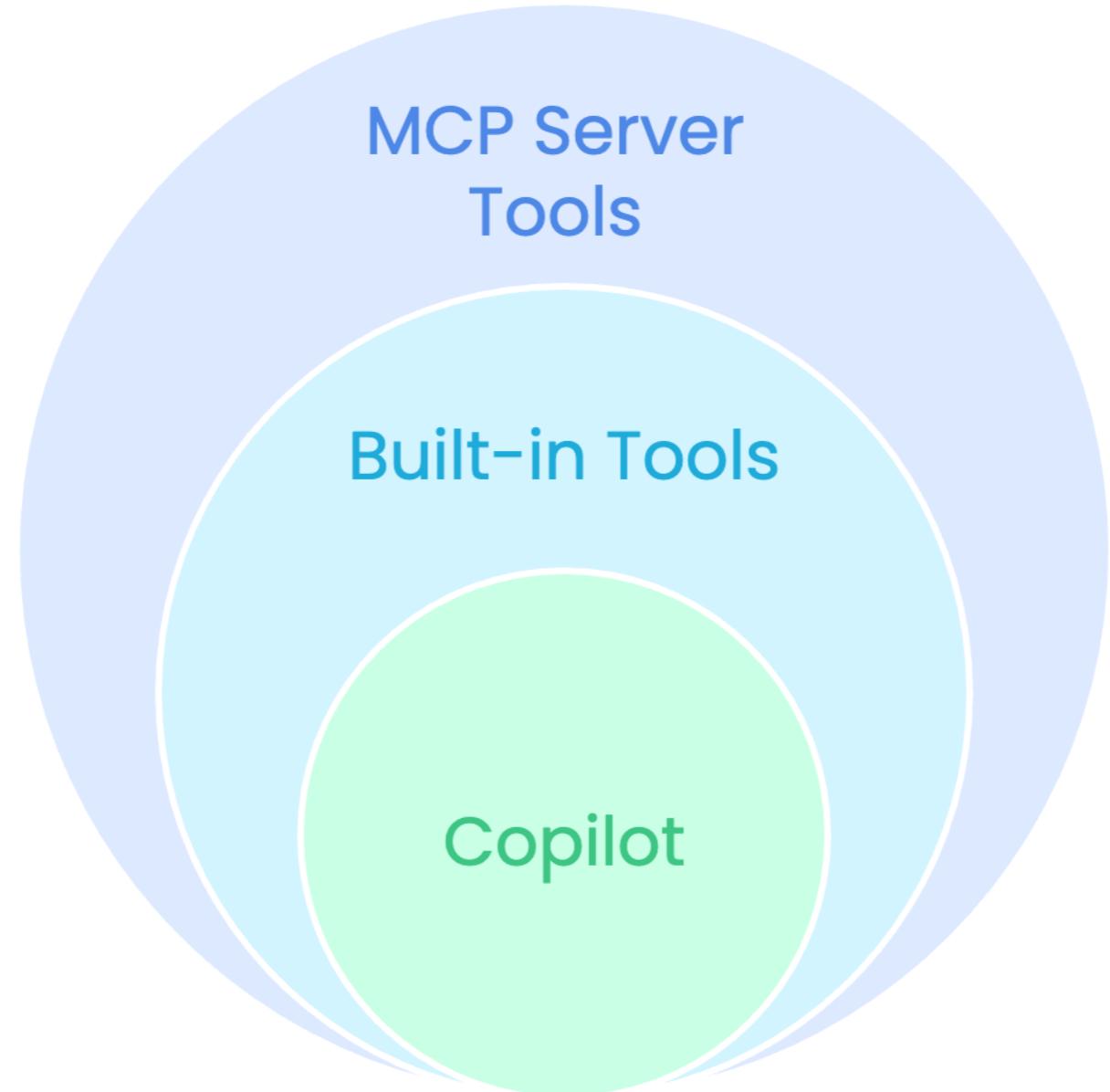
# MCP servers extend Copilot's toolset



# MCP servers extend Copilot's toolset

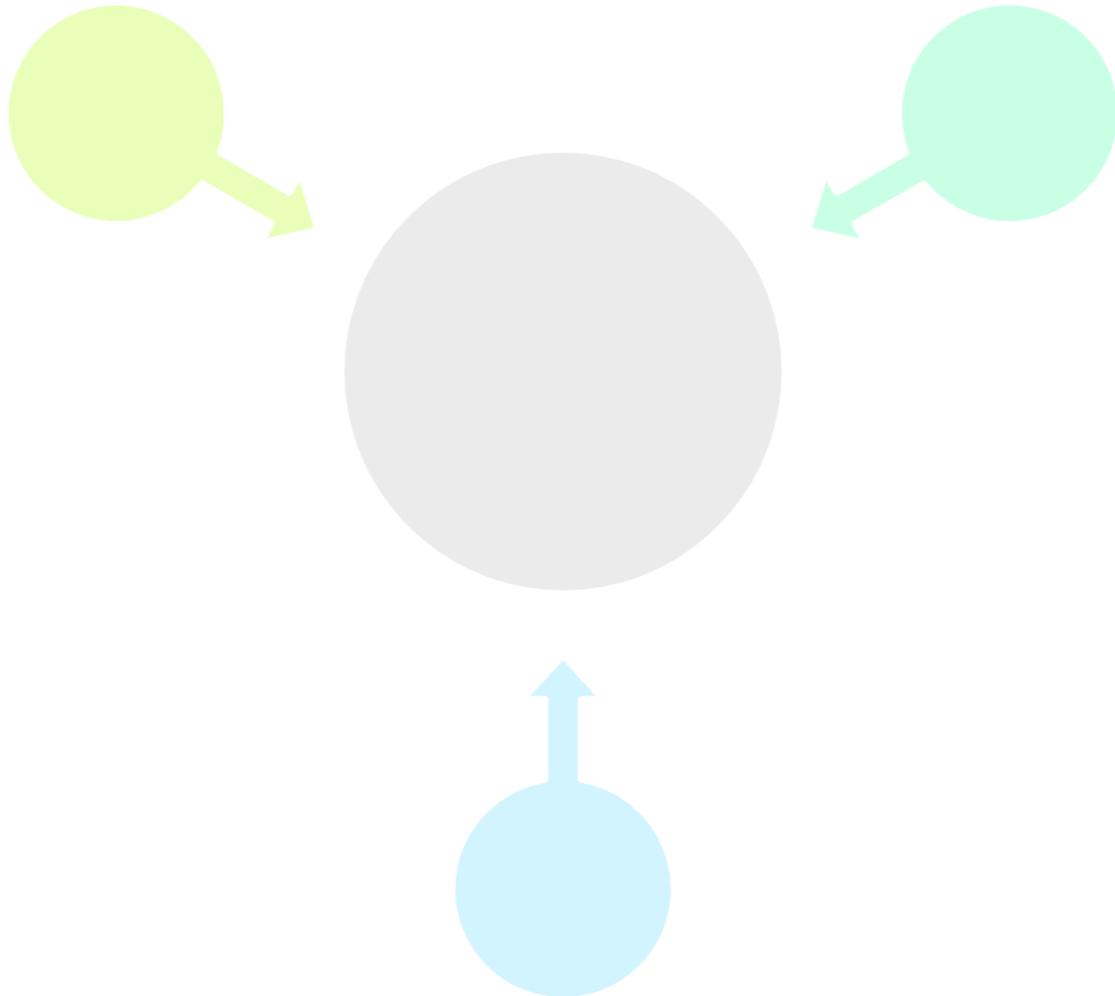


# MCP servers extend Copilot's toolset

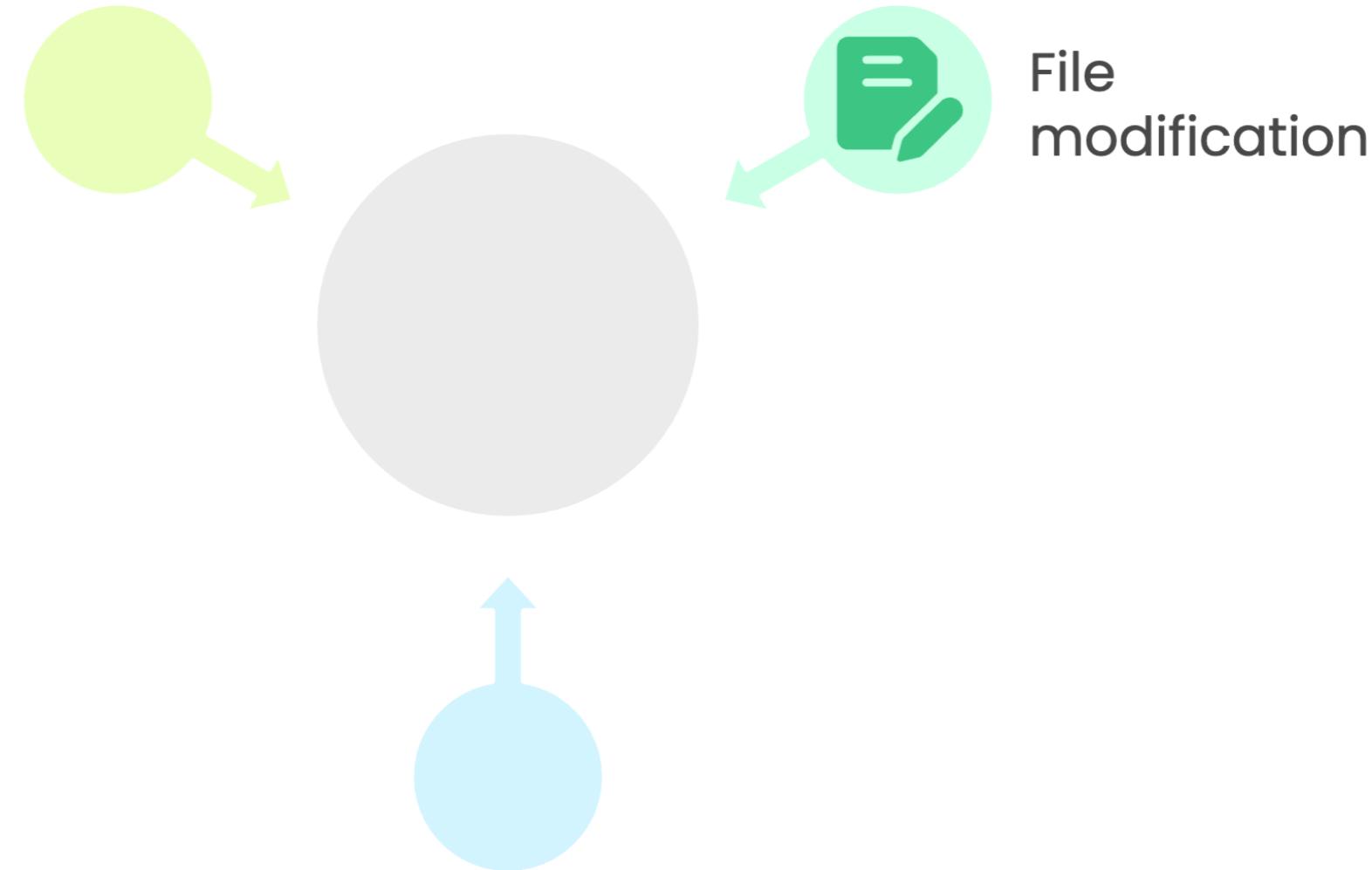




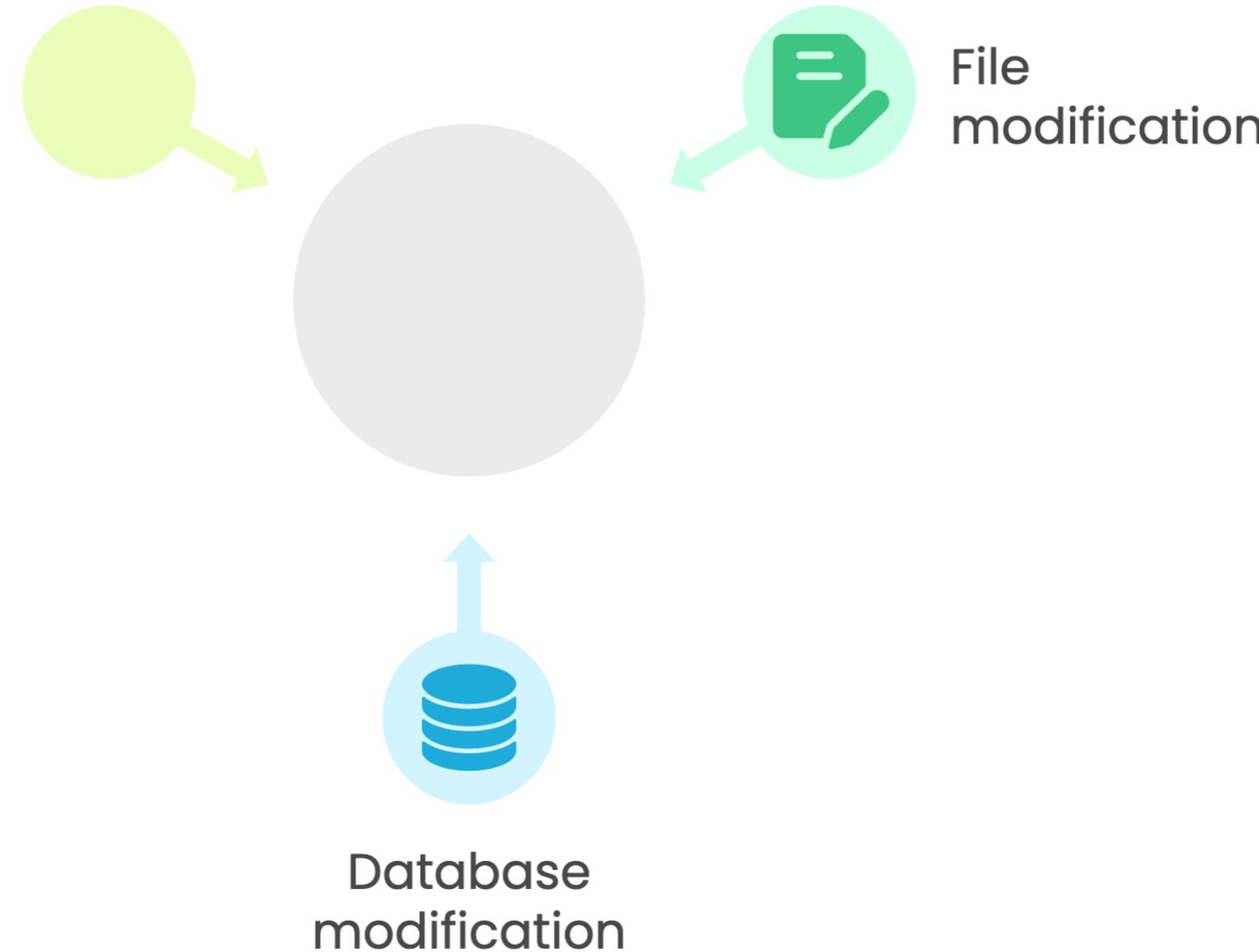
# MCP security considerations



# MCP security considerations



# MCP security considerations



# MCP security considerations



# MCP security considerations

Resource  
modification



File  
modification



Database  
modification

# MCP security considerations



# **Let's practice!**

**SOFTWARE DEVELOPMENT WITH GITHUB COPILOT**

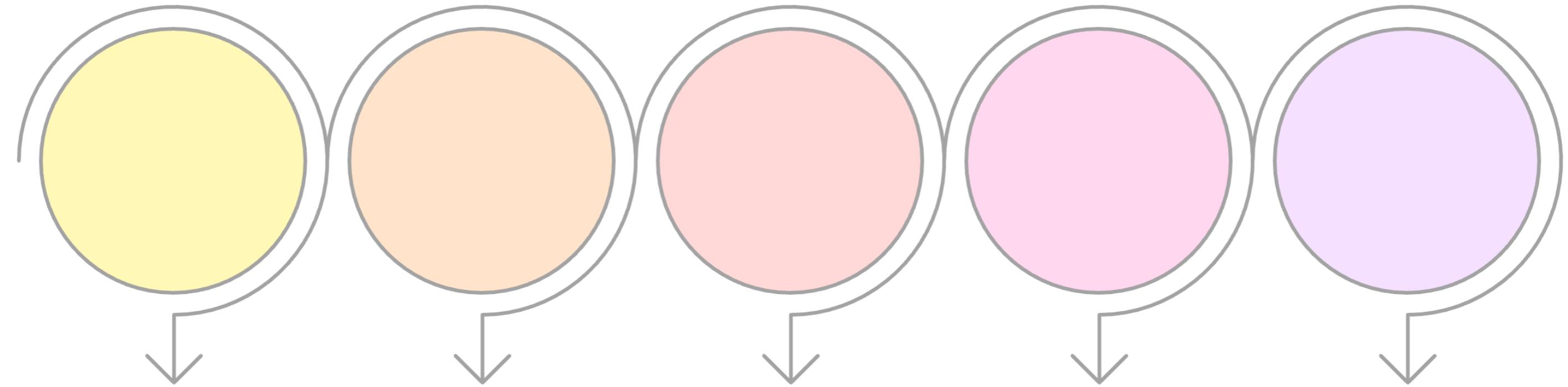
# Congratulations!

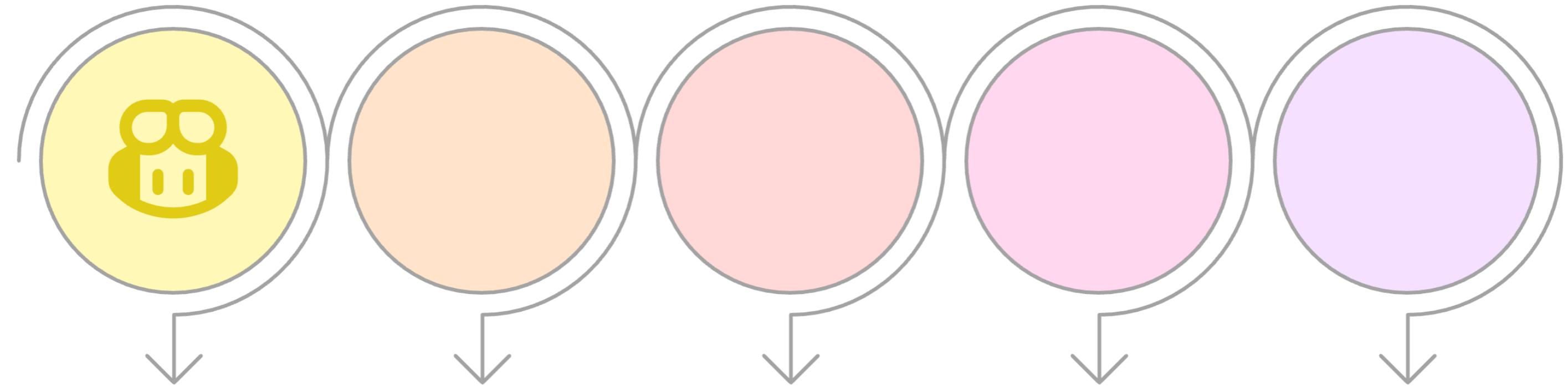
SOFTWARE DEVELOPMENT WITH GITHUB COPILOT



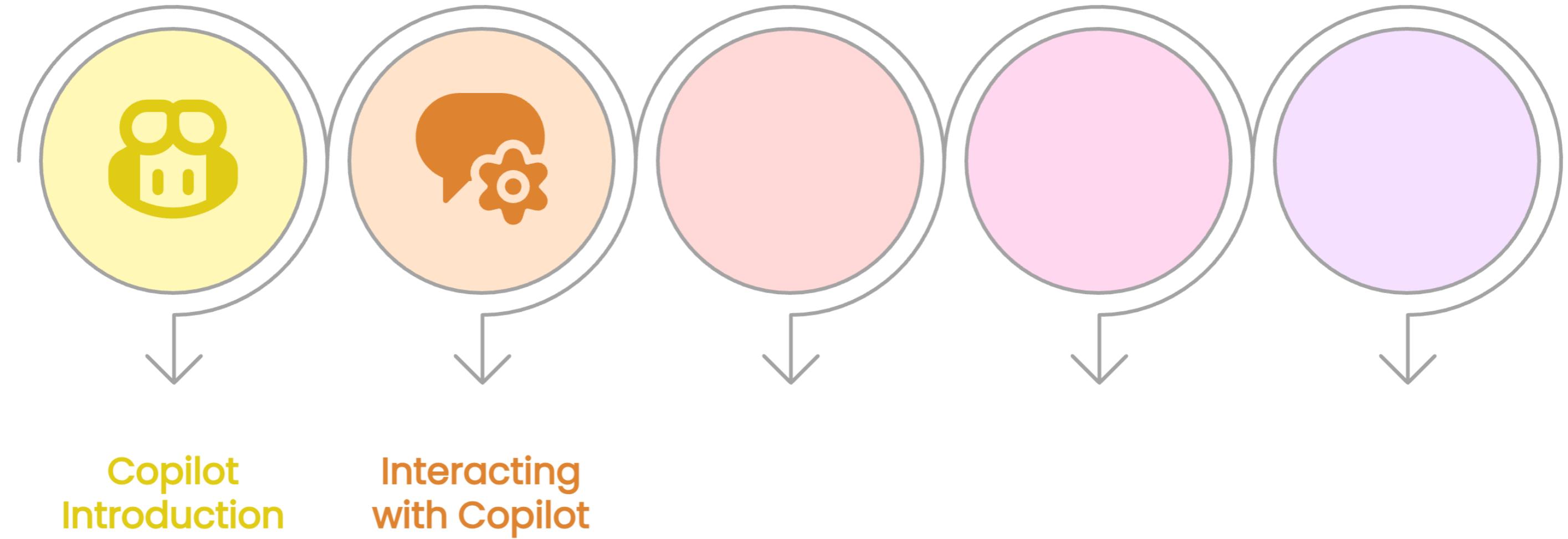
Thalia Barrera

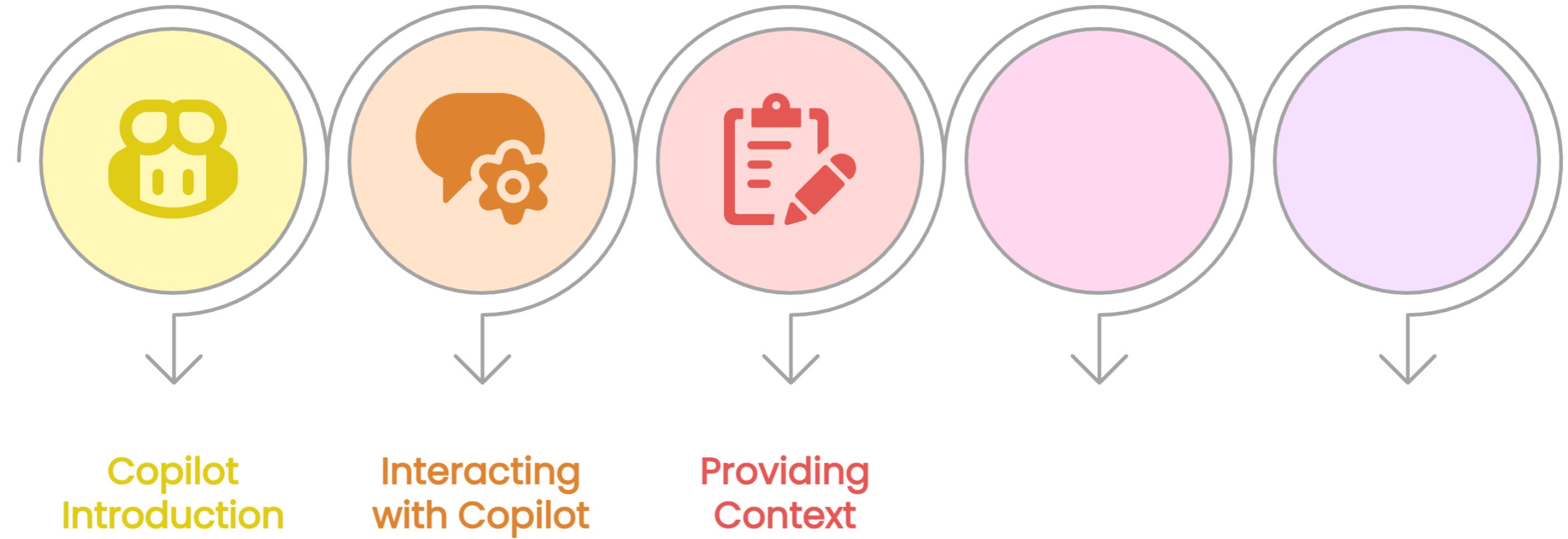
AI Engineering Curriculum Manager,  
DataCamp

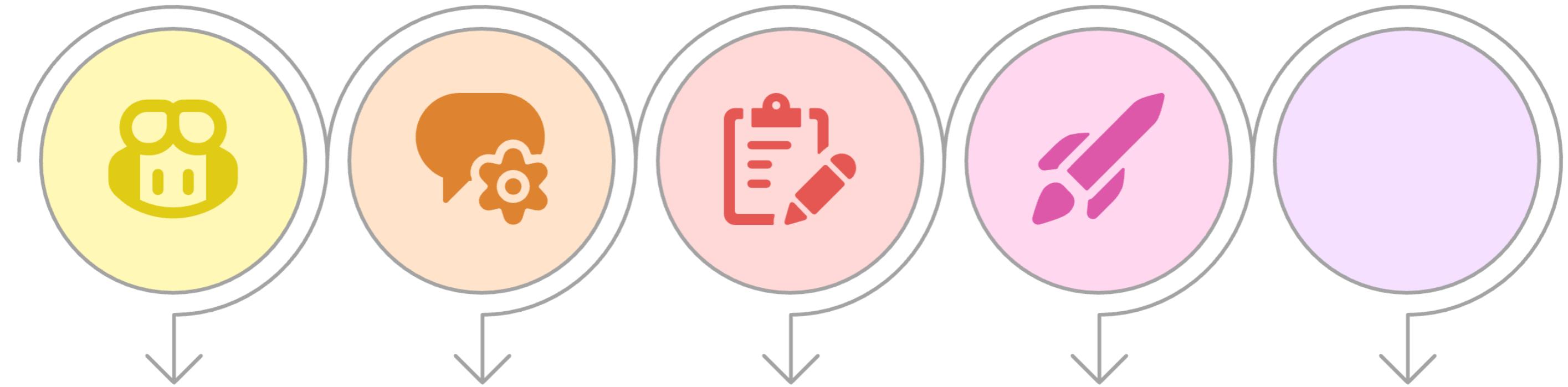


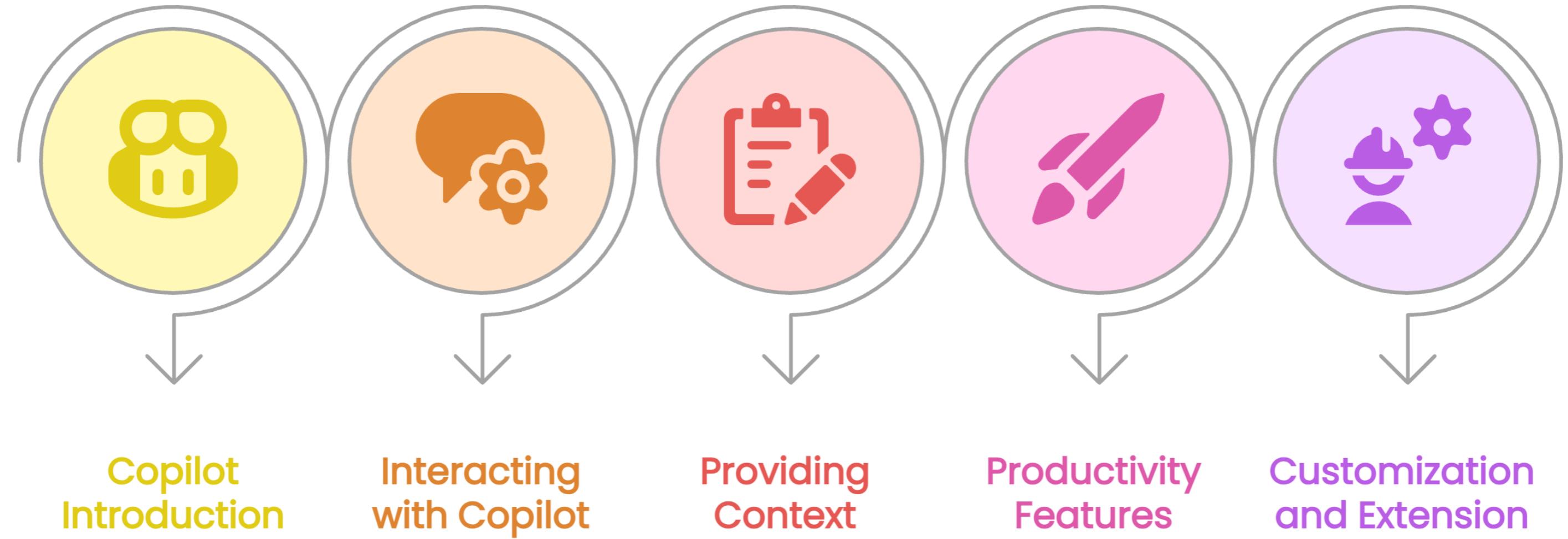


**Copilot  
Introduction**

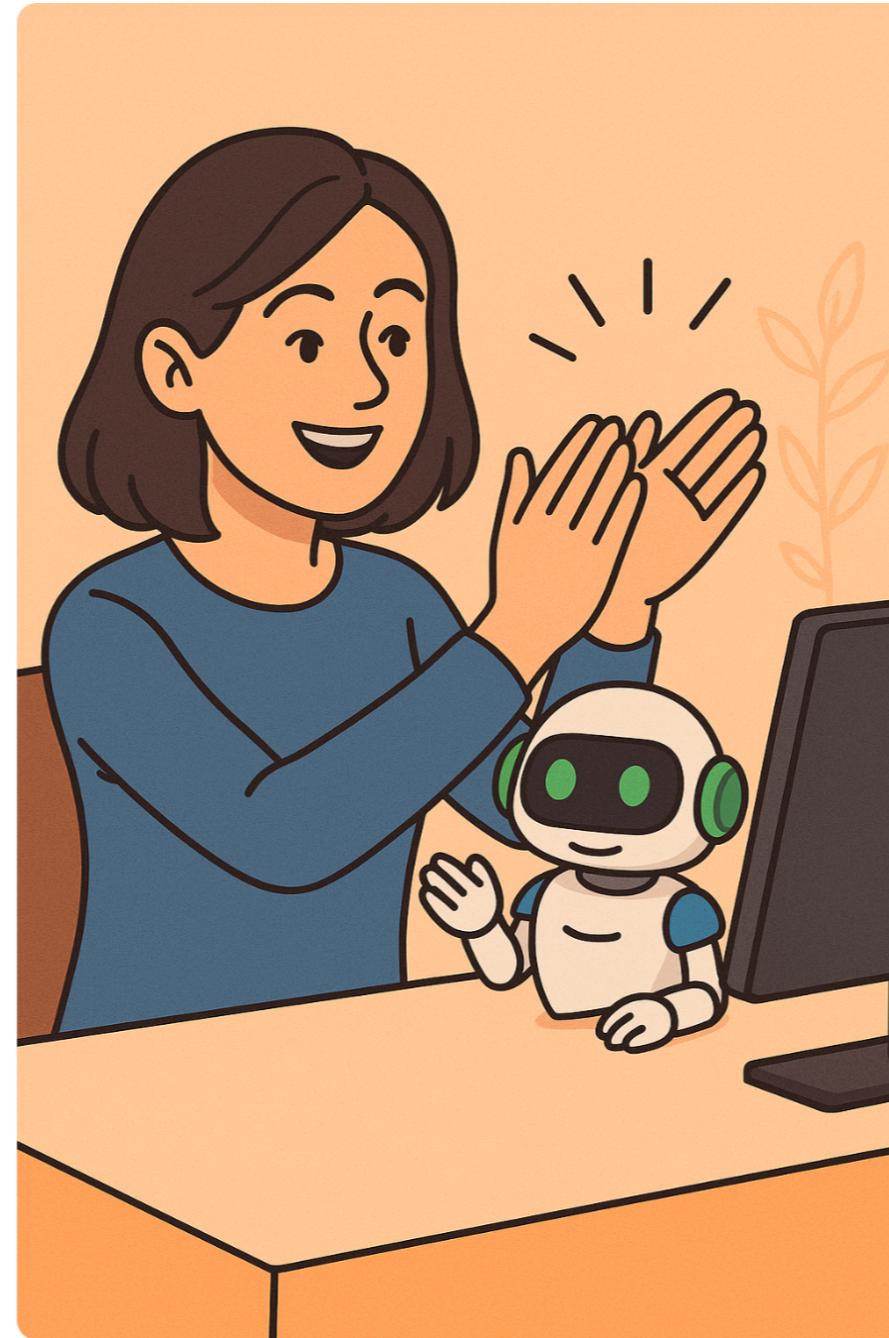








# It's your turn!



# Keep building!

SOFTWARE DEVELOPMENT WITH GITHUB COPILOT