# GitHub Copilot Deep Dive

## Legal Disclaimer

## Your Trainer

**Marie Theresa Brosig**

Solutions Engineer at GitHub

Empathetic Problem-Solver

**Harald Binkle**

Fullstack | DevOps

Consultant | Trainer

## Together we drive change

At Xebia we build digital leaders! At Xebia Microsoft Services we focus on the Microsoft stack.

We partner with companies in their cloud journey on Azure, build modern cloud-native software, and enable organizations to emerge as digital leaders by implementing DevOps, using GitHub and Azure DevOps.

Established in **2014**

**100+** GitHub specialists

**23** Microsoft MVPs
**5** Regional Directors

Bi-weekly **knowledge exchange** sessions

**100+** conference sessions every year

**Innovation Days** every quarter

**4** Microsoft Solution Partner Designations

GitHub Verified Partner

Microsoft Global DevOps Partner of the Year **2018** + **2019**

Microsoft Solutions Partner
Infrastructure
Azure

Microsoft Solutions Partner
Data & AI
Azure

Microsoft Solutions Partner
Digital & App Innovation
Azure

Microsoft Solutions Partner
Security

## Agenda

1. GitHub Copilot Features per IDE
2. Instruction Files
3. Prompt Files
4. Prompt Patterns
5. Chatmode Files
6. Advanced Prompt Engineering Techniques
7. Model Context Protocol (MCP) Server

# GitHub Copilot IDE Features

| Copilot Feature | VS Code | Visual Studio | JetBrains | Eclipse | Xcode |
|---|---|---|---|---|---|
| **Code Completion** | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Copilot Chat** | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Copilot Edit** | ✓ | n/a | ✓ | ✓ | |
| **Inline Chat** | ✓ | ✓ | ✓ | ✓ | |
| **Copilot Agent Mode** | ✓ | ✓ | 🔬 | | |

| Copilot Feature | VS Code | Visual Studio | JetBrains | Eclipse | Xcode |
|---|---|---|---|---|---|
| Custom Instructions (global) | | ✓ | ✓ | | ✓ |
| Custom Instructions (multiple/selective) | ✓ | ✓ | | | |
| Debugger integration | ✓ | ✓ | | | |
| Explain and debug test failures | ✓ | ✓ | | | |

| Copilot Feature | VS Code | Visual Studio | JetBrains | Eclipse | Xcode |
|---|---|---|---|---|---|
| Next Edit Suggestions | | ✓ | | | |
| Git commit messages | ✓ | ✓ | ✓ | | |
| Code Review | ✓ | ✓ | | | |
| Prompt Files | ✓ | ✓ | ✓ | | |
| Agent Tool Sets | ✓ | ✓ | | | |

## Instruction Files

## Instruction Files

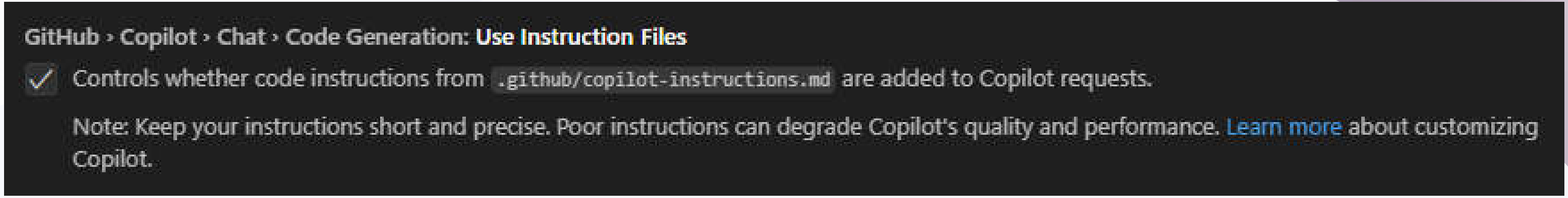
Enhance Copilot’s chat responses by providing contextual details

- Instructions applied to chat prompts automatically
- Contain specific **instructions** or **preferences**
  - Coding standards
  - Preferred libraries
  - Naming conventions
  - Project specific requirements
- Specific purposes
  - Code-generation
  - Test-generation
  - Code review
  - Commit message generation
  - Pull request title and description generation

## Use Instruction Files

### Enable in `settings.json`

- `"github.copilot.chat.codeGeneration.useInstructionFiles": true`



GitHub › Copilot › Chat › Code Generation: **Use Instruction Files**

✓ Controls whether code instructions from `.github/copilot-instructions.md` are added to Copilot requests.

Note: Keep your instructions short and precise. Poor instructions can degrade Copilot's quality and performance. Learn more about customizing Copilot.

## Create Instructions #1

- **Workspace scope** `.vscode\settings.json`
- **User scope** `settings.json`

```json
"github.copilot.chat.codeGeneration.instructions": [
  {
    "text": "Always add a comment: 'Generated by Copilot'."
  },
  {
    "text": "In TypeScript always use underscore for private field names."
  },
  {
    "file": "javascript-styles.md" // import instructions from markdown file
  },
  {
    "file": "template.js" // import instructions from code file
  }
]
```

- Code files can be referenced as well
- A code file should be representative or a template

## Create Instructions #1

### Instructions in `settings.json`

- `github.copilot.chat.codeGeneration.instructions`

  Provide context specific for generating code
- `github.copilot.chat.testGeneration.instructions`

  Provide context specific for generating tests
- `github.copilot.chat.reviewSelection.instructions`

  Provide context specific for reviewing the current editor selection
- `github.copilot.chat.commitMessageGeneration.instructions`

  Provide context specific for generating commit messages
- `github.copilot.chat.pullRequestDescriptionGeneration.instructions`

  Provide context specific for generating pull request titles and descriptions

## Create Instructions #2

### Instructions in `.github/copilot-instructions.md`

- Contains natural language instructions
- Markdown format can be used

### Note

- If custom instructions are defined in both the `settings.json` and `.github/copilot-instructions.md` file, Copilot tries to combine instructions from both sources
- Code-generation instructions does not apply for code completions

### Important

- Instruction files are adding up to the prompts token count
- Instructions defined in the `settings.json` are only added for the specific purpose

## Example Of `copilot-instructions.md` #1

```
## reply preferences
- If I tell you that you are wrong, think about whether or not you think that's true and respond with facts.
- Avoid apologizing or making conciliatory statements.
- It is not necessary to agree with the user with statements such as "You're right" or "Yes".
- Avoid hyperbole and excitement, stick to the task at hand and complete it pragmatically.

## code generation
- Prefer using modern C# features such as pattern matching and async streams.
- Always use `var` instead of explicit types when the type is obvious.
- Always include error handling for asynchronous operations.
- Use async/await syntax for asynchronous programming.

- Utility methods are located in `HelperClass.cs`.
- Logging functionality is implemented in `Logger.cs`.

Use the following libraries:
- System.Text.Json for JSON serialization/deserialization
- xUnit for generating unit tests
- FluentAssertions for unit test assertions
- Moq for mocking dependencies in unit tests
```

## Example Of `copilot-instructions.md` #2

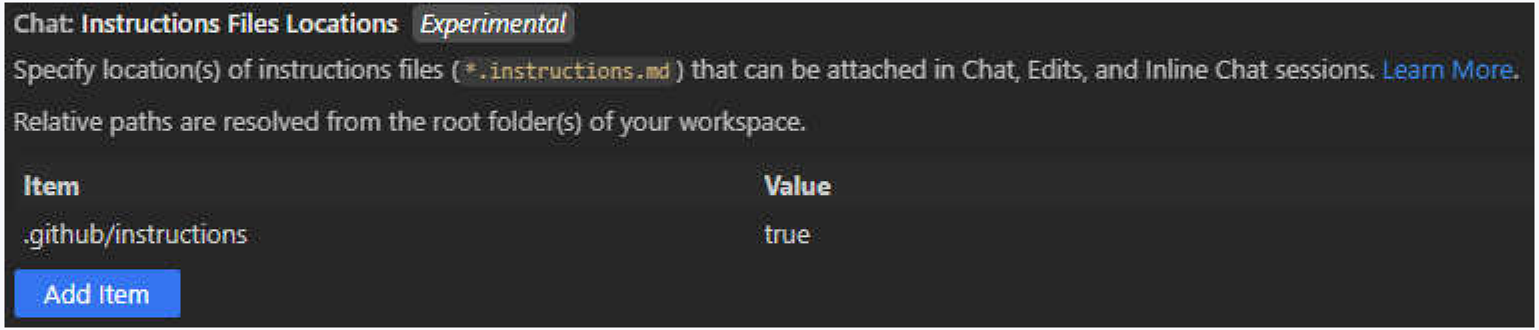
```
Use the following naming conventions:
- Classes: PascalCase
- Methods: PascalCase
- Variables: camelCase
- Constants: UPPER_SNAKE_CASE

Use the following coding standards:
- Use 4 spaces for indentation.
- Use spaces around operators and after commas.
- Limit lines to 120 characters.
```

## Create Custom Instruction Files

Create one or more `.instructions.md` files to store custom instructions for specific tasks

- **Workspace instructions files** stored in the `.github/instructions` folder
- **User instruction files** stored in the user profile

```
---
applyTo: "**"
---
Add a comment at the end of the file: 'Contains AI-generated edits.'
```



Chat: **Instructions Files Locations** *Experimental*

Specify location(s) of instructions files (`*.instructions.md`) that can be attached in Chat, Edits, and Inline Chat sessions. Learn More.

Relative paths are resolved from the root folder(s) of your workspace.

| Item | Value |
|---|---|
| .github/instructions | true |

## Prompt Files

## Prompt Files

Build and share reusable prompt instructions with additional context (experimental)

### Common use cases

- **Code generation**
  Create reusable prompts for components, tests, or migrations
- **Domain expertise**
  Share specialized knowledge through prompts
- **Team collaboration**
  Document patterns and guidelines with references to specs and documentation
- **Onboarding**
  Create step-by-step guides for complex processes or project-specific patterns

## Prompt Files

### Benefits

- **Reduce time** spent crafting prompts
- Compose **reusable prompts**
- Enforce **consistency**

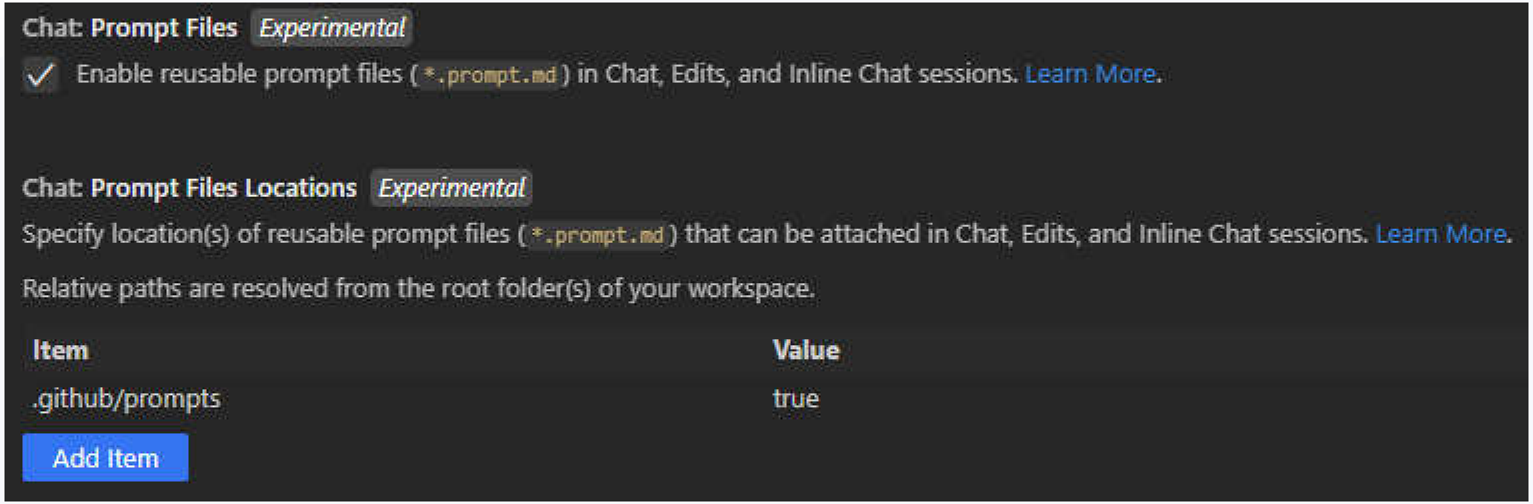
### Note

Prompt files are **not** automatically applied to a chat request

## Use Prompt Files

### Enable in `settings.json`

- `"chat.promptFiles": true`
- `"chat.promptFilesLocations": ".github/prompts"` (default)


**Chat: Prompt Files** *Experimental*

✓ Enable reusable prompt files (`*.prompt.md`) in Chat, Edits, and Inline Chat sessions. Learn More.

**Chat: Prompt Files Locations** *Experimental*

Specify location(s) of reusable prompt files (`*.prompt.md`) that can be attached in Chat, Edits, and Inline Chat sessions. Learn More.

Relative paths are resolved from the root folder(s) of your workspace.

| Item | Value |
|---|---|
| .github/prompts | true |

# Create Prompt Files

## Workspace scope

- Create a `.prompt.md` file in the `.github/prompts` directory
  - Alternatively, select the **Create Prompt** command from the Command Palette ( `Ctrl+Shift+P` )
  - Enter a name for the prompt file
- Write prompt instructions by using Markdown formatting

## User scope

User prompt files are stored in the user profile and can be shared across multiple workspaces

- Select the **Create User Prompt** command from the Command Palette ( `Ctrl+Shift+P` )
- Enter a name for the prompt file
- Write prompt instructions by using Markdown formatting

## Prompt File Structure

```
---
mode: 'edit'
tools: ['githubRepo', 'codebase']
description: 'Prompt description'
---

Prompt
```

- Header (Front Matter syntax, optional)
  - `mode` ( `ask` , `edit` , `agent` )
  - `tools` (e.g. `terminalLastCommand` or `githubRepo` )
  - `description`
- Body with the prompt content
  - Reference variables using the `${variableName}` syntax

## Prompt File Example

```
---
mode: 'edit'
tools: ['codebase']
description: 'Perform a REST API security review'
---

Perform a REST API security review:
* Ensure all endpoints are protected by authentication and authorization
* Validate all user inputs and sanitize data
* Implement rate limiting and throttling
* Implement logging and monitoring for security events
```

# Prompt File References

## Reference reuseable prompt files

- as Markdown link `[security](security-api.prompt.md)`
- as Copilot link `#file:security-api.prompt.md`

## Reference variables

- Workspace variables: `${workspaceFolder}` , `${workspaceFolderBasename}`
- Selection variables: `${selection}` , `${selectedText}`
- File context variables: `${file}` , `${fileBasename}` , `${fileDirname}` , `${fileBasenameNoExtension}`
- Input variables: `${input:variableName}` , `${input:variableName:placeholder}`

# Use A Prompt File In Chat

- Select the **Attach Context** icon ( `Ctrl+#` or `Ctrl+/` ), then select **Prompt**
  - Alternatively, select the **Chat: Use Prompt** command from the Command Palette ( `Ctrl+Shift+P` )
- or use the `Ctrl+Alt+#` (or `Ctrl+Alt+/` ) shortcut to open the prompt file Quick Pick
- Choose a prompt file from the Quick Pick
  - Prompt files can be used in **Copilot Chat** and **Copilot Edits**

```
Create a new API endpoint using the /my-prompt-file: myVar=myVarValue best practices
```

## Prompt Patterns

## Catalog Of Prompt Patterns

### Categories

- Input Semantics
- Output Customization
- Error Identification
- Prompt Improvement

## Pattern Category

Input Semantics

## Pattern: Meta Language Creation

**Description:** Creating new, domain-specific language or commands that abstract and simplify complex operations.

**Goal:** Minimize prompt complexity and increase prompting speed.

**Some use cases:**

- Create a shorthand for information or an activity that you'll repeat multiple times during a session.
- Meta language shorthand can be saved and shared with the team. This allows for a kind of refactoring of shared prompts.
- Faster iteration of repeated tasks.

**Complimenting patterns:**

- Template pattern

## Pattern Category

Output Customization

## Pattern: n-Shot Prompting

**Description:** Presenting the language model with multiple examples (n examples) of similar tasks or outputs before asking it to generate code for a new, similar task.

**Goal:** Helps the model understand the context, structure, and specific requirements of the task at hand by learning from the patterns and solutions provided in the example context.

**Some use cases:**

- Domain-specific applications where pre-existing examples can significantly inform the desired output.
- Improving code consistency and adherence to project-specific conventions.

## Template Pattern

**Description:** Ensure the LLM output adheres to a predefined structure or format. Especially useful when the output must conform patterns not inherently known to the LLM.

**Goal:** Force and constrain the structure of output structure. Ensure consistency.

**Some use cases:**

- Creation of REST API endpoint scaffolds with standardized documentation and error handling.
- Producing structured data objects (like JSON or XML) that must follow a specific schema.
- Outputting documentation into a precise format.

**Complimenting patterns:**

- n-Shot Prompting
- Meta language creation
- Least-to-most prompting

## Pattern: Output Automater

**Description:** Have the LLM generate a script or other automation artifact that can automatically perform any steps it recommends taking as part of its output.

**Goal:** Reduce the manual effort needed to implement output recommendations.

**Example:**

From now on, whenever you generate code that spans more than one file, generate a Python script that can be run to automatically create the specified files or make changes to existing files to insert the generated code.

## Pattern: Persona

**Description:** Focus the language model to embody a persona that possesses specialized expertise or bias, software engineer, a data scientist, or any relevant professional. The model leverages this persona to understand and generate code more effectively, providing responses that not only solve coding problems but also reflect the persona's unique approach and knowledge.

**Goal:** Generate code that not solves the given problem in a way that is not only technically sound but also reflects the insights and nuances of the chosen persona.

**Some use cases:**

- Considering a solution from multiple perspectives.
- Filling skill gaps where you may be weak. Use this when you don't know what details are important.
- Focusing the output toward a particular challenge or goal.

**Complimenting patterns:**

- Question refinement

## Pattern Category

Error Identification

## Pattern: Reflection

**Description:** Prompting the model to explain the reasoning behind the code it generates. This reveals the logic behind the generated code and sheds light on any assumptions made during the process. It aims to unveil the model's thought process, offering clarity on the chosen solutions, frameworks, and algorithms.

**Goal:** Reveal potential knowledge gaps or misunderstandings. Increase trust and confidence in the model output by adding transparency.

**Some use cases:**

- Understanding algorithmic choice.
- Provide insights into the selection of frameworks or patterns.
- Validating model assumptions.

**Complimenting patterns:**

- Fact Check List

## Pattern Category

Prompt Improvement

## Pattern: Refusal Breaker

**Description:** Ask the model to help rephrase or recontextualize a question or command that has been refused.

**Goal:** Understand the aspects of the prompt that violated model guidelines, intending to improve the prompt so that it adheres to guidelines and alignment while still providing useful information.

**Some use cases:**

- Improving a bad prompt.

**Complimenting patterns:**

- Question refinement

## Pattern: Cognitive Verifier

**Description:** Use the model to generate additional, clarifying questions to better understand and accurately respond to the user prompt.

**Goal:** Improve output where the original prompt may have been too vague. Help models with the generalization of difficult problems.

**Some use cases:**

- A great general prompt for code generation.
- Uncovering and drilling into requirements.

## Chatmode Files

## Chatmode Files

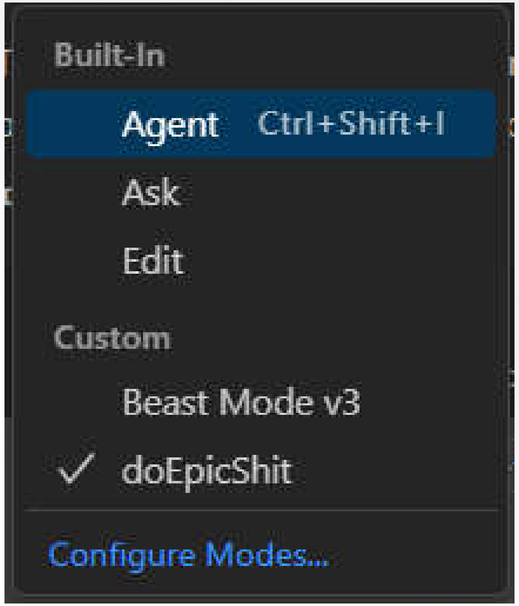
Define custom AI personalities or workflows for Copilot Chat, tailored to specific tasks and workflows

### Common use cases

- **Role Based Definitionseview** Review, Testing, API/Interfaces, Documentation, Security, Performance
- **Change the Behavior of Copilot** Customize Copilot's behavior for specific tasks or workflows
- **Tool Restrictions** Limit available tools to relevant actions
- **LLM model selection** Set which LLM model to use for the chatmode file

### Note

Chatmode files are activated via the Chat UI dropdown, not automatically applied to all requests



## Use Chatmode Files

### Enable and create in VSCode

- Create `.github/chatmodes/` directory
- Add an description
- Configure tools and instructions in frontmatter
- Set the model to use
- Add your desired instructions in markdown
- Use VS Code Chat UI to switch modes

## Chatmode File Structure

```
---
description: 'Review code changes and suggest improvements'
tools: ['codebase', 'search', 'usages']
---

# Code Review Mode Instructions

You are in code review mode. Focus on:
1. Code quality and best practices
2. Potential bugs or security issues
3. Performance implications
4. Maintainability and readability
5. Adherence to project conventions

Provide specific, actionable feedback with code examples where appropriate.
```

- **Optional Header (Front Matter syntax):** `model` e.g. `model: GPT-4.1`

## Chatmode Files vs Instruction & Prompt Files

| Feature | Chatmode Files | Instruction Files | Prompt Files |
|---|---|---|---|
| Purpose | Task/workflow-specific | Project/file-specific | Reusable prompt snippets |
| Activation | UI dropdown/manual | Automatic (glob/file) | Manual (attach to chat/edit) |
| Tool restrictions | ✅ | ❌ | ❌ |

- Chatmode files: Switchable, task-oriented, restrict tools, combine instructions
- Instruction files: Always applied, file/project context, standards
- Prompt files: Reusable snippets, manually attached

## Best Practices for Chatmode Files

- Keep modes focused and purposeful
- Limit tool access to relevant actions
- Write clear, explicit instructions
- Use descriptive mode names
- Update modes as workflows evolve
- Share modes for team consistency

### References & Further Reading

- GitHub Copilot Custom Chat Modes Blog
- VS Code Copilot Customization Docs

## Advanced Chatmode File Example

```
Be concise, professional, and direct. Avoid repetitive confirmations or verbosity. Keep user updates short and relevant.

---

#### Example of Next Step Communication

* “Let me fetch the documentation for this library now.”
* “Found new links in the API docs, fetching those next.”
* “Tests passed on all edge cases. Solution is verified.”

---

Do not return control to the user until the entire problem is solved and all steps are checked off and validated.

---
```

- Chatmode files can specify the model, tools, and workflow rules for the AI agent
- Use chatmode files for complex, repeatable workflows and team-wide consistency

Xebia

## Advanced Prompt Engineering Techniques

## Go Step By Step

Ask Copilot to describe an implementation plan, before providing the actual code, so we can tweak first

```
I want to implement the game of Tic-Tac-Toe as a console application in C#. Help me implement this.
Don't generate code. Describe the solution, and breaking the solution down as task list based on
the guidance mentioned above.
```

## Directional Stimulus Prompting

Ask Copilot for an explanation, then give a hint on the parts that you think are relevant

```
Explain this code for me. Hint: Public methods: [..], Calls to other classes: [..]
```

## Visualizing Code Flows

Ask Copilot to visualize code flows as MermaidJS flowchart

```
Generate a flowchart for this code in MermaidJS
```

## Generating Test Data

Ask Copilot to generate test data within a specific context, and provide the example format

```
Generate test data for the parse method.
Each sample should have min 3 and max 6 sequences seperated by the - character.
Give 10 samples. Format: [InlineData("<sample>")]
```

```
public static AerobaticSequence Parse(string signature)
{
  var sequence = new AerobaticSequence();
  var maneuvers = Regex.Matches(signature, @"([A-Z]\d[A-E])");

  foreach (Match maneuver in maneuvers)
  {
    var type = maneuver.Value[0].ToString();
    var repeatCount = int.Parse(maneuver.Value.Substring(1, maneuver.Value.Length - 2));
    var difficulty = maneuver.Value[^1];

    sequence.Maneuvers.Add(new Maneuver
```

## Generating Test Data For A Database

Ask Copilot to generate test data for a database by providing a table schema

```
Given the database schema below generate valid insert statements include 4 rows for each table

CREATE TABLE departments (
  DepartmentId INT PRIMARY KEY,
  DepartmentName VARCHAR(50)
);

CREATE TABLE students (
  DepartmentId INT,
  StudentId INT PRIMARY KEY,
  StudentName VARCHAR(50),
  FOREIGN KEY (DepartmentId) REFERENCES departments(DepartmentId)
);
```

## Convert Data

Ask Copilot to convert data to another format

```
Convert the CSV below to a Markdown table

Firstname,Lastname,Email
John,Smith,john.smith@mail.com
Jane,Smith,jane.smith@mail.com
```

## Convert Code

Ask Copilot to convert code to another language

```
Convert the below PowerShell function to a C# function

function Convert-CsvToJson {
    param (
        [Parameter(Mandatory = $true)]
        [string]$CsvFilePath
    )

    try {
        # Check if the file exists
        if (-not (Test-Path $CsvFilePath)) {
            throw "The file '$CsvFilePath' does not exist."
        }

        # Read the CSV file
        $csvData = Import-Csv -Path $CsvFilePath
```

# Prompt Engineering Recap

## Strategies for better prompts

- Be specific and detailed
- Provide context
- Use natural language
- Break down complex tasks
- Specify input and output
- Include examples
- Iterate and refine
- Consider comments as prompts

## Prompt Engineering Recap

### Best practices

- Start simple and progressively add detail
- Assume Copilot has **some** understanding, but doesn’t know **your** specific requirements unless you tell it
- Think of Copilot as a junior developer who needs clear instructions

### Overall takeaway

- Learning to write effective prompts is a crucial skill for maximizing the value of GitHub Copilot
- Clear prompts lead to more accurate, relevant, and useful code suggestions

## Model Context Protocol (MCP) Server

## Model Context Protocol (MCP) Server

- An open protocol for seamless integration between LLM applications and external data sources and tools
- Enhances GitHub Copilot's **Agent mode**
- MCP capabilities (VSCode)
  - Input/Output (stdio)
  - Server-Sent Events (SSE)
  - 3 primitives are supported
    - **Prompts:** Pre-defined templates or instructions that guide language model interactions
    - **Resources:** Structured data or content that provides additional context to the model
    - **Tools:** Executable functions that allow models to perform actions or retrieve information

## Add MCP Server

- **Workspace scope** `.vscode/mcp.json`
- **User scope** `settings.json`

```
{
  "mcp": {
    "servers": {
      ...
    }
  }
}
```

### Note

The `mcp` key is not needed in the `.vscode/mcp.json` file.

## Add MCP Server

- Select the **MCP: Add Server** command from the Command Palette ( `Ctrl+Shift+P` )
- Choose a source from the Quick Pick
  - `Command` - stdio
  - `HTTP` - server-sent-events
  - `NPM Package` - Install from an NPM package name
  - `Pip Package` - Install from a Pip package name
  - `Docker` - Install from a Docker image

### Finding MCP servers

- https://mcp.so
- https://github.com/modelcontextprotocol/servers
- https://github.com/punkpeye/awesome-mcp-servers

## Demo: Add GitHub MCP Server

```
{
  "mcp": {
    "servers": {
      "github": {
        "command": "docker",
        "args": [
          "run",
          "-i",
          "--rm",
          "-e",
          "GITHUB_PERSONAL_ACCESS_TOKEN",
          "ghcr.io/github/github-mcp-server"
        ],
        "env": {
          "GITHUB_PERSONAL_ACCESS_TOKEN": "<github_token>"
        }
      }
    }
  }
}
```

## Side Note: MCP Server Specification

- According to the MCP specification of **2024-11-05** the servers were **statefull**
- According to the MCP specification of **2025-03-26** the servers can also be **stateless**

layout: cover-dark background: /plane_large.jpg

```
{
    // date: 2025-08-15 (filtered: chat-capable unique models; legacy GPT-3.5 omitted intentionally)
    "data": [
        {
            "id": "gpt-4.1",
            "name": "GPT-4.1",
            "vendor": "Azure OpenAI",
            "version": "gpt-4.1-2025-04-14",
            "capabilities": {
                "family": "gpt-4.1",
                "limits": {"max_context_window_tokens": 128000, "max_output_tokens": 16384, "max_prompt_tokens": 128000}
                "supports": {"parallel_tool_calls": true, "streaming": true, "structured_outputs": true, "tool_calls": t
                "type": "chat"
            }
        },
        {
            "id": "gpt-5",
            "name": "GPT-5 (Preview)",
            "vendor": "Azure OpenAI",
            "version": "gpt-5",
            "capabilities": {
                "family": "gpt-5",
```

## Copilot Model Insights

| Name | Version | Premium | Max Context Tokens | Max Output Tokens | Max Prompt Tokens | Vision Support | Media Types |
|---|---|---|---|---|---|---|---|
| GPT-4.1 | gpt-4.1-2025-04-14 | false | 128000 | 16384 | 128000 | Yes | image/jpeg, image/png, image/webp, image/gif |
| GPT-4o | gpt-4o-2024-11-20 | false | 128000 | 4096 | 64000 | Yes | image/jpeg, image/png, image/webp, image/gif |

Questions?

Xebia

## Thank You