

GitHub Copilot & Prompt Engineering Training

June 2025

Day 1

COMARCH

Meet Your GenAI Coaching and Coordination Team

Telco

Coaching Team



Sebastian Zaranek
Over 14 years of experience as a Java Developer, Product Owner, Architect, and Release Manager



Łukasz Popiółek
12 years of experience as a Java Developer, R&D Subdepartment Manager



Daniel Słotwiński
Seasoned software engineer and full-stack developer, with expertise in Java and Scala



Tomasz Krygier
Senior Java Developer and Team Leader with 10 years of experience



Wojciech Biskowski
Software Development Engineer in Test



Marta Fiedorowicz
Frontend developer with experience in team-leading, 5+ years in Comarch



Kamil Nowak
10 years of experience as a Java Developer and Team Leader



Przemysław Zagórski
Seasoned software expert with 12+ years in the Telco industry, focused on solution design, technical leadership, and hands-on development



Katarzyna Lewczuk
11 years of experience as a Java Developer

Coordination Team



Bartłomiej Górný
Project Coordinator

Meet Your GenAI Coaching and Coordination Team

Services

Coaching Team



Wojciech Guziak
Head of the competency team



Artur Laskowski
Java Developer | University
Lecturer | Trainer | Simplifying
Complexity with Passion



Grzegorz Miącz
Data Scientist and Architect



Marek Pownug
Software Developer and Team
Leader



Krzysztof Bieniecki
Experienced software professional
with over 15 years at Comarch,
specializing in development,
design, & leadership



Marcin Moczała
15+ years of software
development experience, recently
mainly in loyalty projects for
banking sector

Coordination Team



Maria Wróblewska
Project Coordinator

Meet Your GenAI Coaching and Coordination Team

ERP

Coaching Team



Kamil Przenioslo
Senior .NET developer with 8 years of experience



Damian Parobiec
Software developer



Dawid Dobrowolski
.NET Software Developer in MSP



Tomasz Nowak
Software engineering professional with over 10 years of experience at Comarch



Mateusz Wolanin
Team leader with 7 years of experience in software development

Coordination Team



Kamil Migacz
Project Coordinator

Meet Your GenAI Coaching and Coordination Team

Financial Services

Coaching Team



Bartek Pietraszuk
Senior Software
Engineer with 14
years of experience



Szymon Jakubiec
Software Engineer



Piotr Walkusz
Lead developer and
full-stack software
engineer



Marcin Gadamer
JVM developer with
a Ph.D., GeeCON
co-organizer and
Polish JUG co-leader



Mikołaj Petecki
Software Engineer



Krzysztof Szczęśniak
Software engineer,
mountains and new
technologies enjoyer



Bartosz Ryndak
Senior software
engineer with 12
years of experience



Tomasz Berger
Senior Software
Engineer



**Sebastian
Fijałkowski**
Full Stack Developer



**Mateusz
Niewiadomski**
Software Engineer



Jarosław Bańkowski
System Architect,
Senior Software
Engineer with **14**
years of experience



Paweł Suchanicz
Senior Java Developer



Tadeusz Szatan
Senior Software
Engineer



Grzegorz Dzioba
Software Architect



Witold Zabłocki
Java developer and
team leader



Mateusz Różański
Software Engineer

Coordination Team



Wojciech Kordeusz
Project Coordinator

Training schedule and expectations



Program schedule for the GitHub Copilot and prompt engineering training

Each module mixes instruction with demos and hands-on code labs

Hands-on activity



Additional time for Mini Project

All times shown in CET

Day 1: Introduction to mini project, feature overview, prompt techniques, context setting and project setup			Day 2: Code generation, unit testing and risk management			Day 3: Code documentation, refactoring, and translation		
9:00	Welcome and intro	15 min		Recap & Quiz	15 min		Recap & Quiz	15 min
9:30	Training schedule and expectations	15 min		Lecture, Demo	15 min		Lecture, Demo	10 min
	Mini project introduction & How to work with material	10 min		Code generation	Code lab, participant screen sharing	50 min	Code review & Communication	Code lab
10:00	Feature overview	30 min					Lecture, Demo	17 min
	Setting context	Lecture, Demo	10 min				Code documentation	10 min
10:30		Code lab	15 min	Break	15min		Code lab	18 min
	Break	15 min					Break	15 min
11:00	Prompt techniques	Lecture, Demo	15 min		Lecture, Demo	15 min	Code refactoring	Lecture, Demo
		Code lab	20 min		Code lab, participant screen sharing	50 min	Code translation	Lecture, Demo
11:30	Custom instructions	20 min					Code refactoring & translation	Code lab, participant screen sharing
		Lecture, Demo	10 min					65 min
12:00	Domain knowledge & Software architecture	Code lab	30 min	Risk management	Lecture, Demo	10 min	MCP overview	25 min
					Code lab	25 min	Feedback & conclusion	5 min
12:30	Feedback & suggestions	5 min		Participant screen sharing	Feedback & suggestions	10 min		
							Work on Mini Project (up to 2 hours)	
				Work on Mini Project (up to 2 hours)				

We will facilitate a 2-week coaching program for you, supported by continuous adoption and measurement efforts

Enablement phase	Activity type	Time	Objective
Learning	 Intro to GenAI	Week 1	Learn how to use GenAI tools and prompting techniques
	 Foundation skills	<ul style="list-style-type: none">• Plenary sessions: 3 days x 4 hrs	Focus on practical use cases like code generation, unit testing, refactoring, and translation
	 Advanced use cases for GenAI	<ul style="list-style-type: none">• Own work: 5 – 10 hrs	Work on a mini-project to apply these skills
	 1:1 Group coaching session	Week 2	<p>Get personalized support to improve your prompts</p> <ul style="list-style-type: none">• 1 coaching session: 30 min – 1hrs <p>Solve challenges</p> <p>Enhance your GenAI skills</p>
	 Community building	Week 2	<p>Join practice calls to showcase mini-projects</p> <ul style="list-style-type: none">• 1 slot per week: 45 min <p>Share your experiences</p> <p>Connect with the GenAI community for Q&A and support</p>
Adoption	 Support in day-to-day activities	Week 3-4	<p>Receive hands-on help to integrate GenAI into your daily tasks and workflows</p> <ul style="list-style-type: none">• 1 coaching session per week: 30 min – 1hrs <p>Learn best practices and success tips from others.</p>
Impact measurement	 Track and monitor metrics	Ongoing	Help measure the impact of GenAI by monitoring quality, speed, and improvements in your work

The impact of Copilot will be assessed using 4 different methods

Method	Source	Description	Expectation
1. Post-training feedback	Self-reported survey	Share your thoughts on the training experience, including what worked well and areas for improvement	Provide honest and constructive feedback to help improve future training sessions
2. Weekly GenAI impact metrics	Self-reported survey	Report how GenAI tools are impacting your daily work, including productivity, experience, and challenges	Complete the weekly survey to track progress and identify areas for additional support
3. GitHub Copilot metrics	GitHub Copilot Metrics APIs	Analyze usage data, such as the frequency of Copilot suggestions and their acceptance rate	Use Copilot actively to generate meaningful data that reflects its impact on your workflow
4. Delivery velocity and quality metrics	Jira, GitLab, SonarQube	Measure improvements in code quality, delivery speed, and overall team performance	Collaborate with your team to ensure accurate tracking of project metrics and outcomes

Program norms & expectations



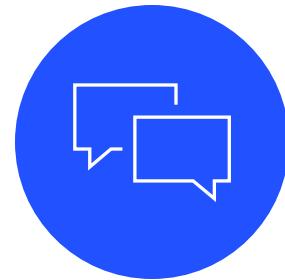
Please stay on mute to avoid any background noise



Please stay with camera turned on for a more personal experience



This session will be recorded for training purposes



Use the chat to ask questions or leave comments



We will have time for feedback at the end of each session



When returning from breakout rooms, ensure you click the button to rejoin the plenary session instead of leaving the session entirely

Moving beyond augmentation, AI now autonomously transforms software development

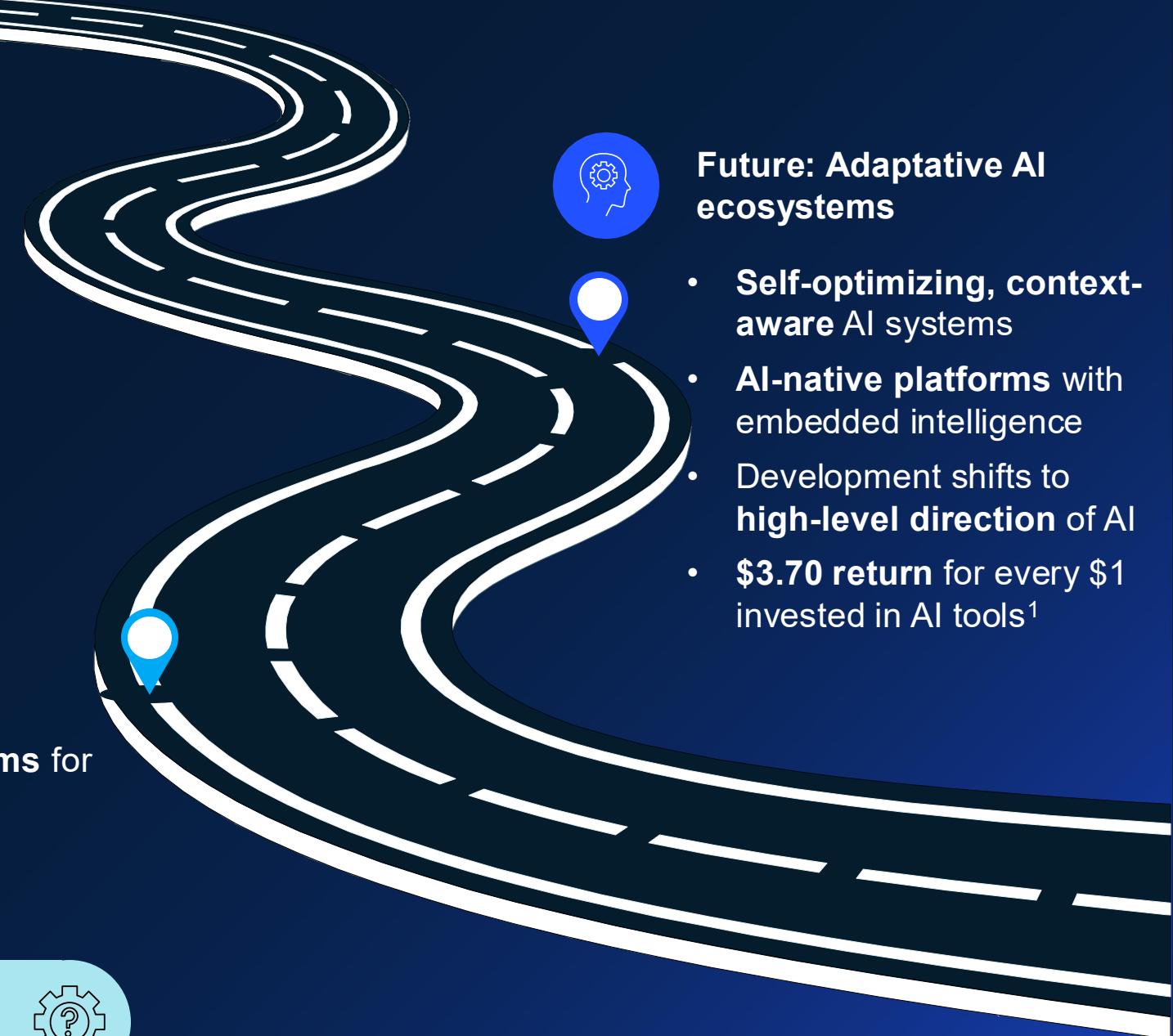


Today: intelligent orchestration

- AI orchestrates **end-to-end development workflows**
- Evolution from **assistive tools** to **autonomous agents**
- Teams deploy **multi-agent systems** for complex tasks
- **75%** of organizations use AI in development¹



Think about what the future of your day-to-day work will look like with increased adoption of AI!



1. Source: <https://news.microsoft.com/en-pee/2025/01/08/6-ai-trends-youll-see-more-of-in-2025/>

What is GitHub Copilot?



What is it?

- An **AI-powered tool that helps developers write code faster and better**
- It can suggest code completions, snippets, functions, tests based on prompts and context of your code



What data was it trained on?

- The tool is trained on a large-scale neural network model that can generate code from natural language
- The data underlying this is trained on **all languages that appear in public repositories**



Limitations

- Quality of suggestions received **varies based on volume and diversity of the training data for that language**
- May produce **suggestions that contain undesirable patterns**

[Quickstart for GitHub Copilot](#)



GitHub Copilot LLM compatibility

Choose the right model for your task

Some models are optimized for **fast coding tasks**, while others are better suited for **slower planning and reasoning tasks**.

Useful links:

- <https://livebench.ai/#/>
- <https://github.blog/ai-and-ml/github-copilot/which-ai-model-should-i-use-with-github-copilot/>
- <https://docs.github.com/en/copilot/using-github-copilot/ai-models/choosing-the-right-ai-model-for-your-task>

Model	Key strengths	When to use	
 Fast coding  OpenAI  Claude	GPT-4o	Fast responses, supports images	Quick iterations, basic code understanding
 Complex coding  OpenAI  Claude  Gemini	Claude 3.5 Sonnet GPT-o4-mini	Good performance-to-cost ratio Low latency, efficient	Documentation, code snippets, quick answers Simple coding questions, prototyping
 Reasoning / planning  OpenAI 	GPT-4.1 GPT-o3 Gemini 2.0 Flash	Better coding, larger context Step-by-step problem solving Works with images	General dev tasks, balanced performance/cost Optimization, debugging, structured code UI debugging, analyzing diagrams
	Claude 3.7 Sonnet	Deep reasoning, debugging	Complex problem-solving, multi-step logic
	GPT 4.5	Multi-file context	Context-heavy tasks, advanced reasoning

Paid premium requests



- As of June 4th GitHub Copilot introduced Paid Premium Requests:
 - The base model (GPT-4.1) remains free for all users
 - Each user receives 300 Premium Requests per calendar month
- You can still use Claude Sonnet, Gemini, or minis/o from GPT
- How to monitor actual consumption?
 - On GitHub: Coordinators are working to make usage reports available to you
 - In the IDE: A feature to check your usage will be released soon

Each model has a premium request multiplier

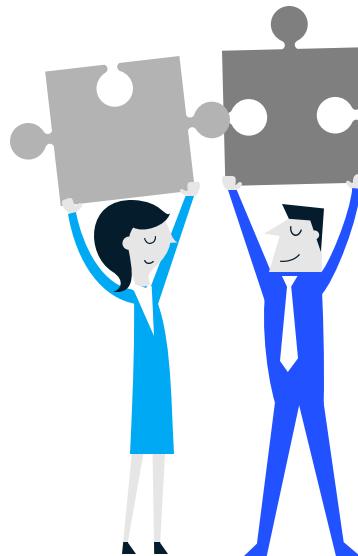
Model	Premium requests
Base model (currently GPT-4.1) [2]	0 (paid users), 1 (Copilot Free)
Premium GPT-4.1	1
GPT-4o	1
Claude 3.5 Sonnet	1
Claude 3.7 Sonnet	1
Claude Sonnet 4	1
Gemini 2.0 Flash	0.25
Gemini 2.5 Pro	1
o1	10
o3	5
o3-mini	0.33
o4-mini	0.33

<https://docs.github.com/en/copilot/managing-copilot/monitoring-usage-and-entitlements/about-premium-requests>

GenAI can be applied on different software engineering use cases

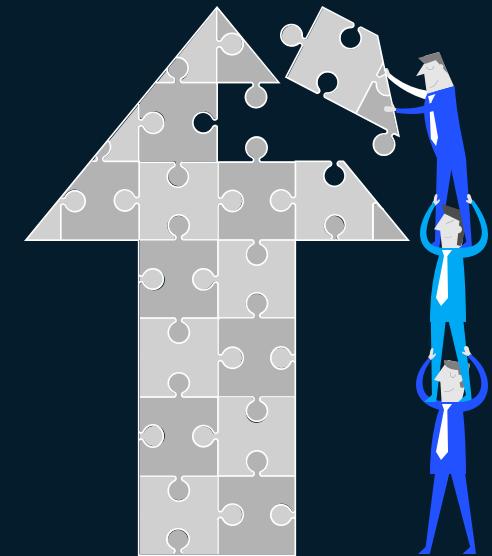
Basic use cases

- 1** Non-coding activities
- 2** Code generation
- 3** Unit testing
- 4** Documentation



Advanced use cases

- 5** Code refactoring
- 6** Code translation



Mini Project Exercise

Each module helps you build an application prototype

Choose between 3 options

- **Option 1: Join a project from the BillingTech BU**
 - They're looking for your expertise to help build a customer-facing application that enables users to:
 - Review their invoices
 - Request changes to specific line items
 - Submit and track refund requests
- **Option 2: Solve what slows you down** – Choose a real pain point from your daily work — and design a solution that could improve life for you and your team (and likely many others)
- **Option 3: Bring your idea to life** – Have an idea you're passionate about? Now's your chance to turn it into reality. Whether it's a product concept, a tool, or a feature you've always wished existed

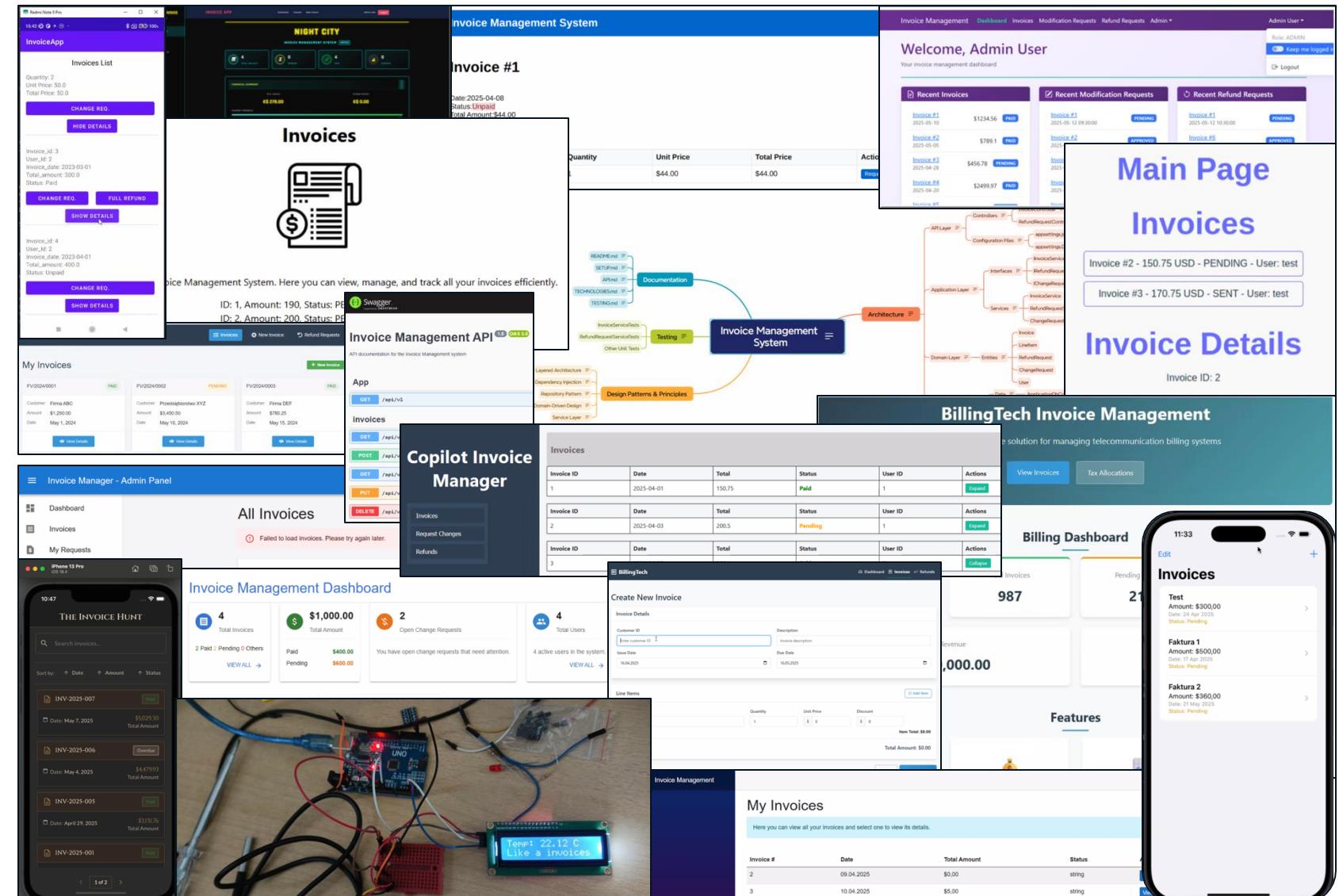
Objectives

1. Develop a three-tier application prototype (frontend, backend, database)
2. Use GenAI tools to speed up the development process
3. Present your solution in breakout rooms, group coaching, or Community of Practice calls
4. Save your code locally as a new project



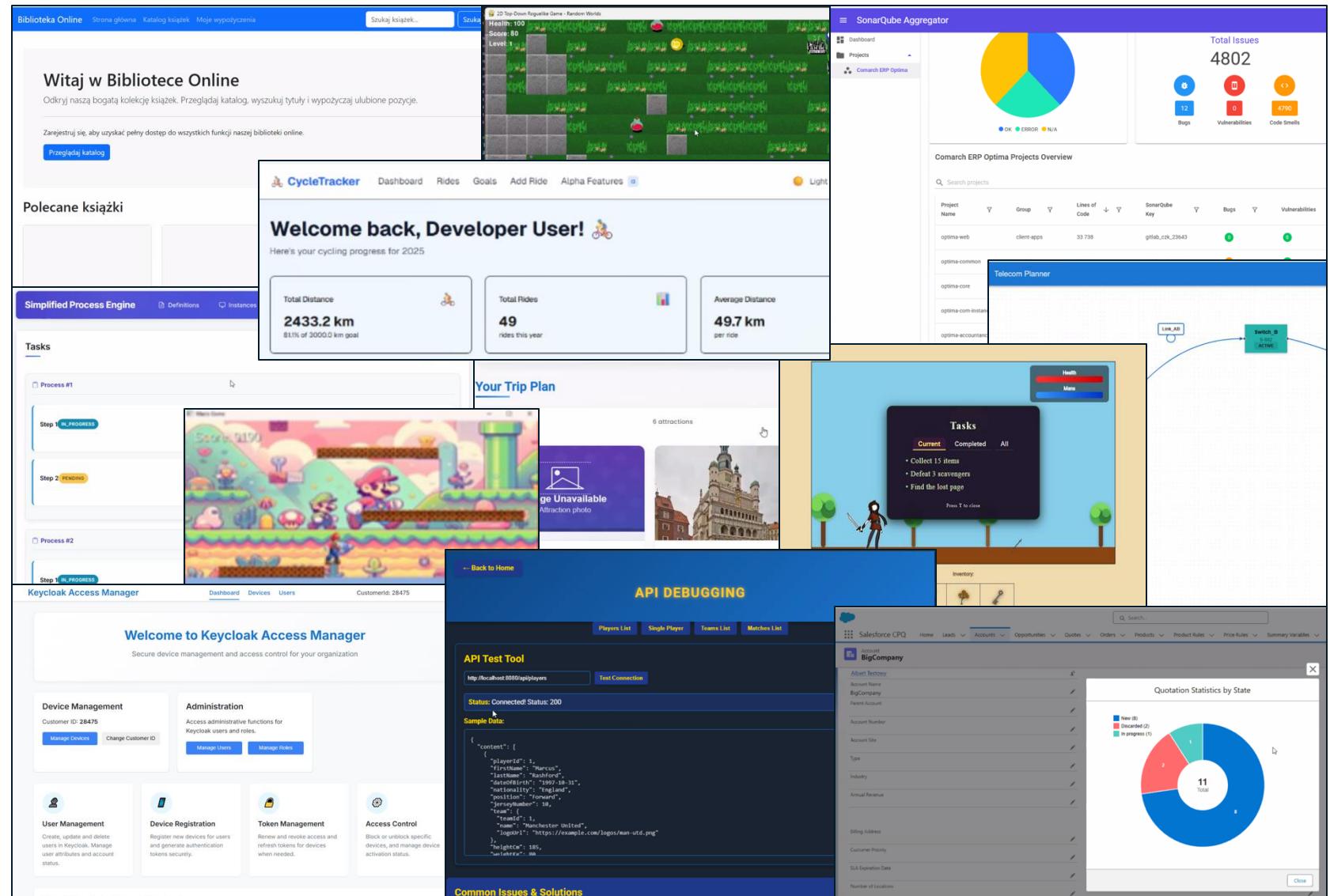
Mini Project: BillingTech prototype

Potential solutions



Mini Project: Alternative prototype

Potential solutions





Demo: Mini Project

Features to demonstrate:

1. Web application prototype
 - Frontend, Backend, Data Storage



Demo: Feature Overview

Features to demonstrate in VS Code and IntelliJ:

1. Chat Functionality:
Interact with Copilot for assistance
2. Inline Code Suggestions:
Real-time, context-aware code completions
3. Multi-Edits:
Simultaneous edits across multiple sections of code
4. Agent Mode (VS Code):
Automate repetitive tasks by delegating them to Copilot as an intelligent assistant
5. Logs:
Review activity logs for insights and debugging

VS Code vs JetBrains IDE feature overview



Chat

VS Code

Offers chat functionality with better UX and features

JetBrains IDE

Offers chat functionality



Code suggestions

VS Code

Offers code suggestions with Next Edit

JetBrains IDE

Offers code suggestions



Multi edits

VS Code

Available with better UX and features

JetBrains IDE

Available with 10 files maximum (Edit mode)



Agent mode & MCP support

VS Code

Available as part of chat mode

JetBrains IDE

Available as public preview of chat mode



Context awareness

VS Code

Highly context-aware for files, packages, folders, screenshots and more using @ and # references

JetBrains IDE

Limited context awareness (max. 10 files), @project references are supported



Logs

VS Code

Provides an integrated terminal and output pane for viewing logs

JetBrains IDE

Offers a general log file



Custom instructions

VS Code

Available, allows granular custom instructions and the ability to save prompts

JetBrains IDE

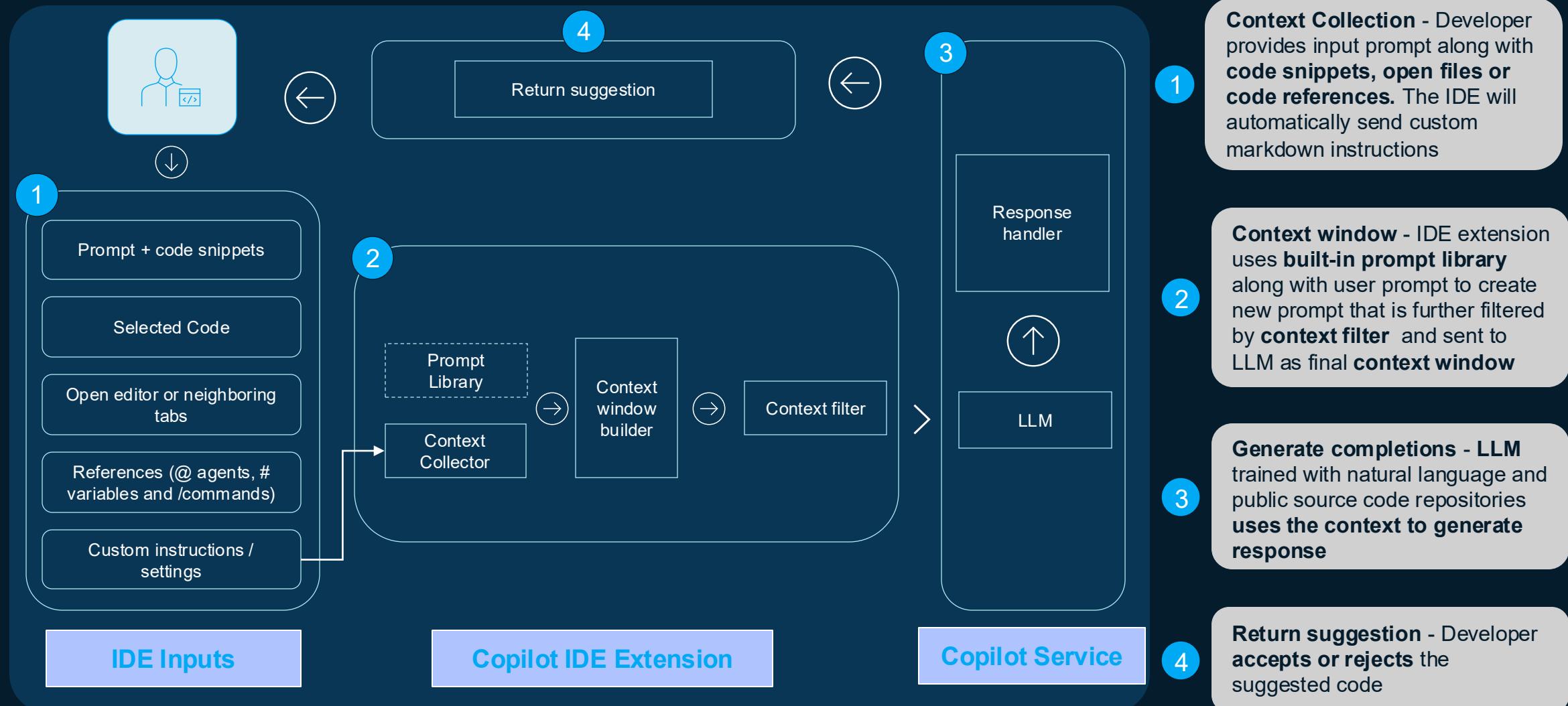
Available, custom instructions are global

Advanced Context Options

Deep dive into context option in GitHub copilot



How does GitHub Copilot create the context for the LLM?



GitHub Copilot creates separate, independent contexts for Code Completions and Chat



Code Completions

1

GitHub Copilot functions as an AI pair programmer that offers two main types of assistance

- **Code completions** that match your style as you type
- **Next Edit Suggestions (NES)** that predict both where and what your next code change will be

It works across numerous programming languages and frameworks, providing suggestions directly in your editor



Chat based context

2

Chat interface allows to interact with a codebase using natural language

Different modes optimized for various tasks - from answering questions to making multi-file edits or running agent workflows

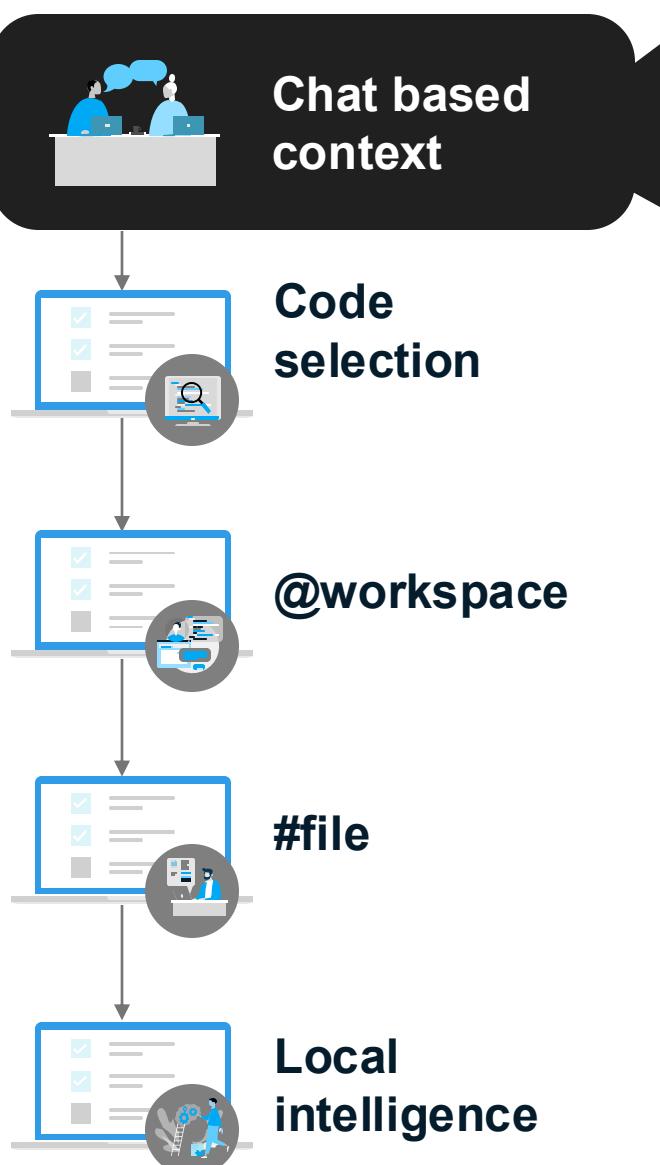
Common uses include understand code, debug issues, get code suggestions, optimize performance, learn best practices

GitHub Copilot chat-based modes

Copilot Chat Mode	1 Ask	2 Edit	3 Agent
	Ask questions about codebase or technology concepts	Make edits across multiple files in your codebase	Start an agentic coding workflow
Context	Active files + workspace info	Selected code + file context	Project-wide context
Prompt	Conversational prompts	Edit-specific instructions	Task-oriented prompts
Output	Natural language responses + code	Code transformations	Multi-step problem solving

GitHub Copilot builds independent context across Code Completions and Chat





Chatting with Copilot as pair programmer by providing information about application and then with follow-up prompts including code snippets builds the context

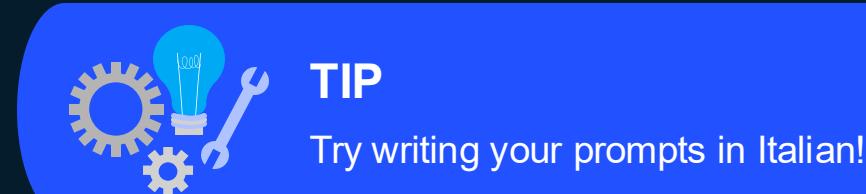


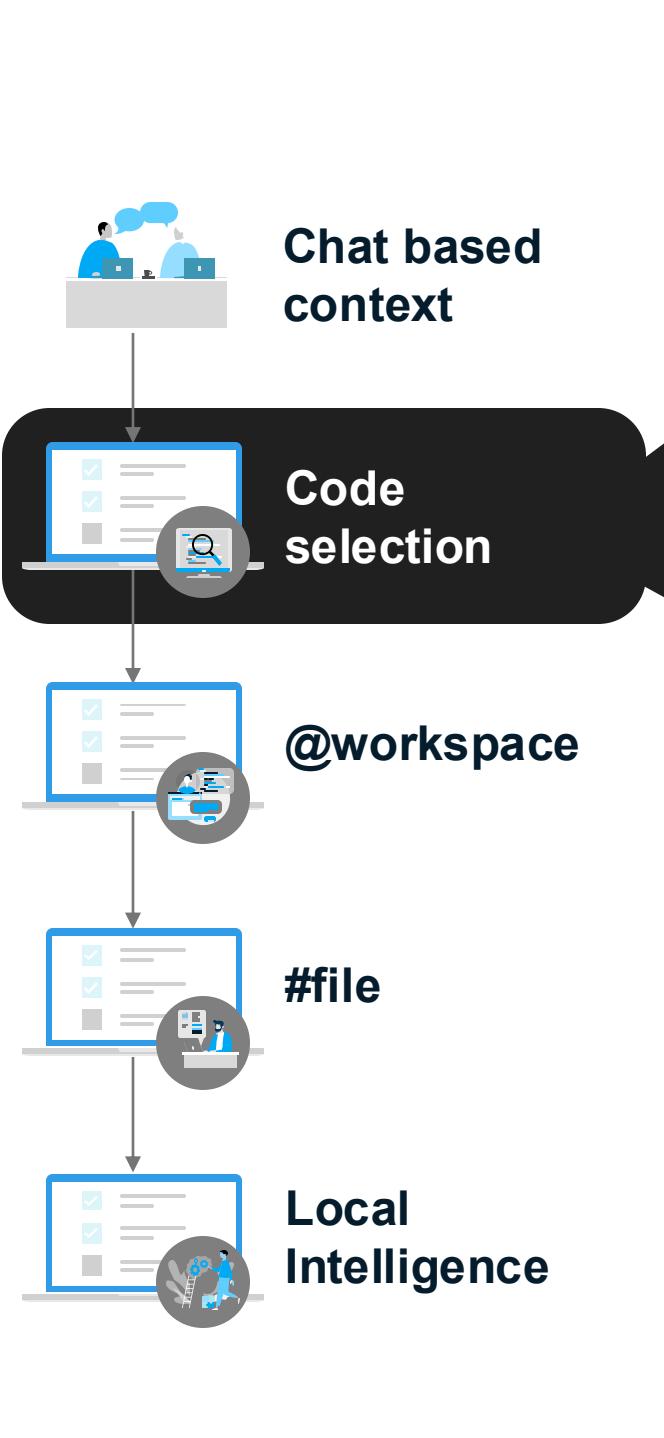
When to use

- Open ended questions
- Complex problems which need description
- Suggestions for libraries / best practices
- Suggestions on terminal commands

Limitations

- Unstable behaviour on shared with open files
- Detailed prompts needed with enough context
- Frequent hallucinations with non-existing functions





Code selection, open editors or neighboring tabs is useful when we want to narrow the context scope down to specific parts of the code.

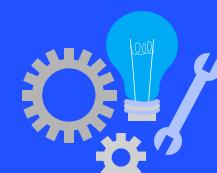


When to use

- When we require information about specific part of the code
- When we want to make a boilerplate code

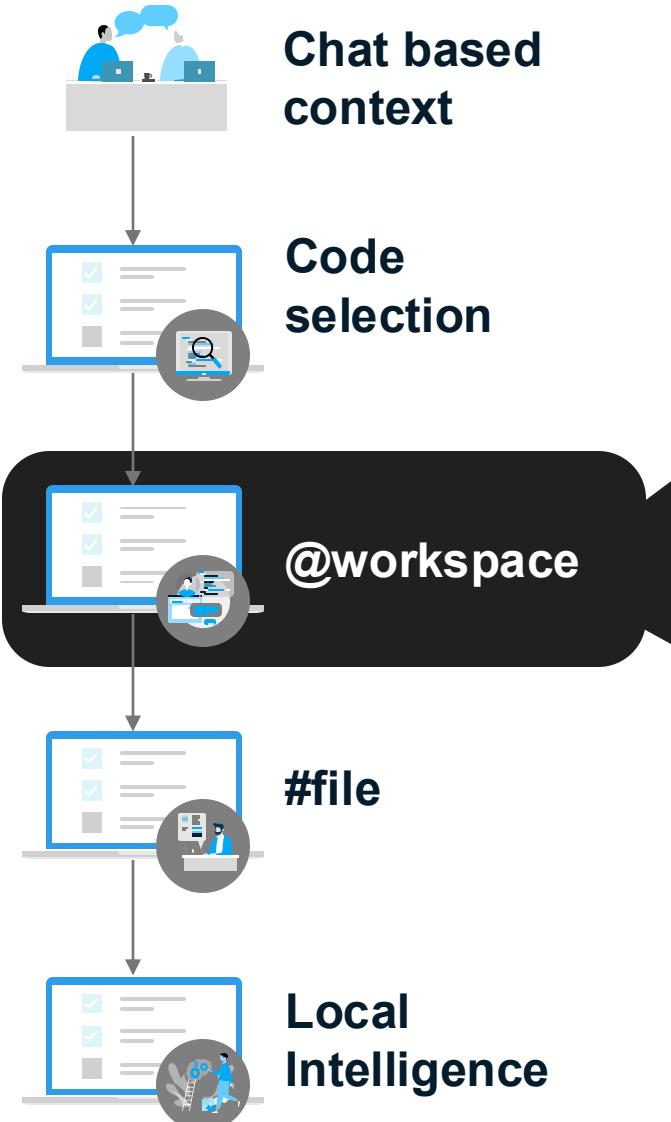
Limitations

- Number of tokens for context is limited
- Not all lines of code from open file or editor is included



TIP

When you want generate docs/tests for a specific part of the code: Select the relevant code > open inline chat > Use the commands /docs or /tests



@workspace agent is available as part of VS code extension for GitHub Copilot chat. It reads the files in workspace to generate the context for copilot.

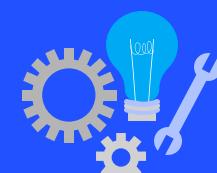


When to use

- When content from multiple files are required in the context from codebase
- Works best for explaining the application, generating documentation for project, generating suggestion using framework or pattern already used in the existing code

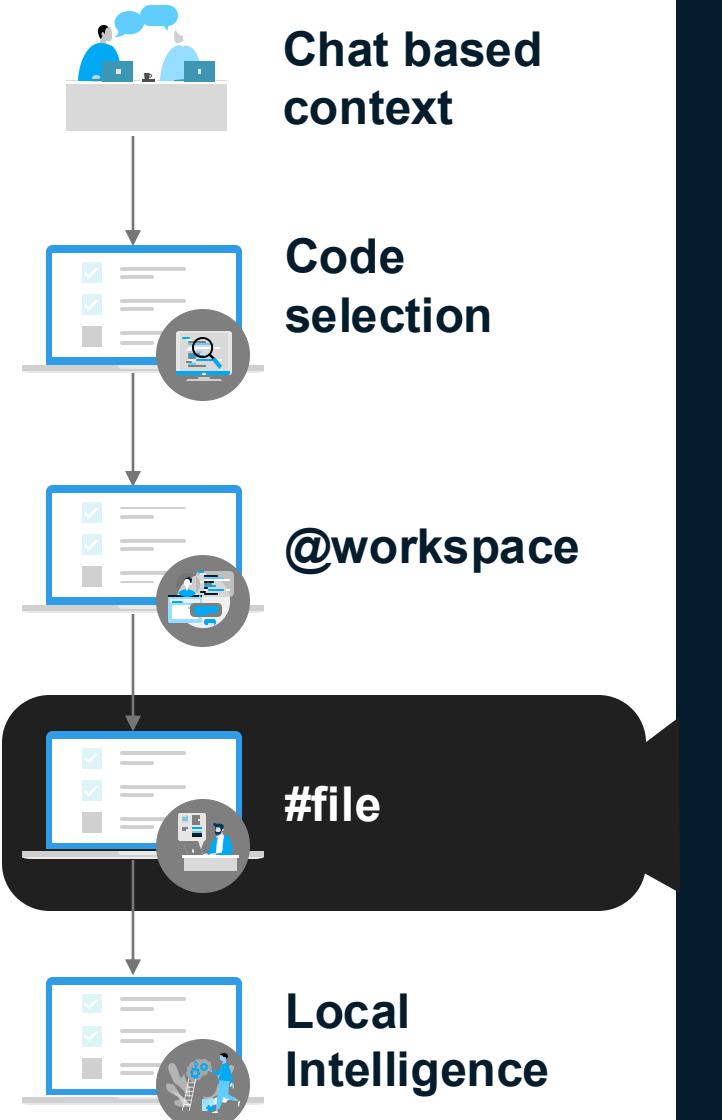
Limitations

- Selected files for references in the context are not in user control
- Not available in all IDEs



TIP

New to the project? You have no clue where to look for something? Copilot's workspace agent will gather as much information as possible.



#file <filename> tag can be used to add references of specific one or multiple files in the context

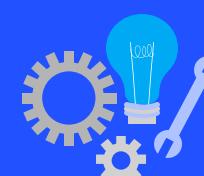


When to use

- Very useful when bringing a library or module in context by selecting the ReadMe or specific file
- Chat provides better result when references to multiple specific files are required
- When implementing a feature, that involves using multiple files.

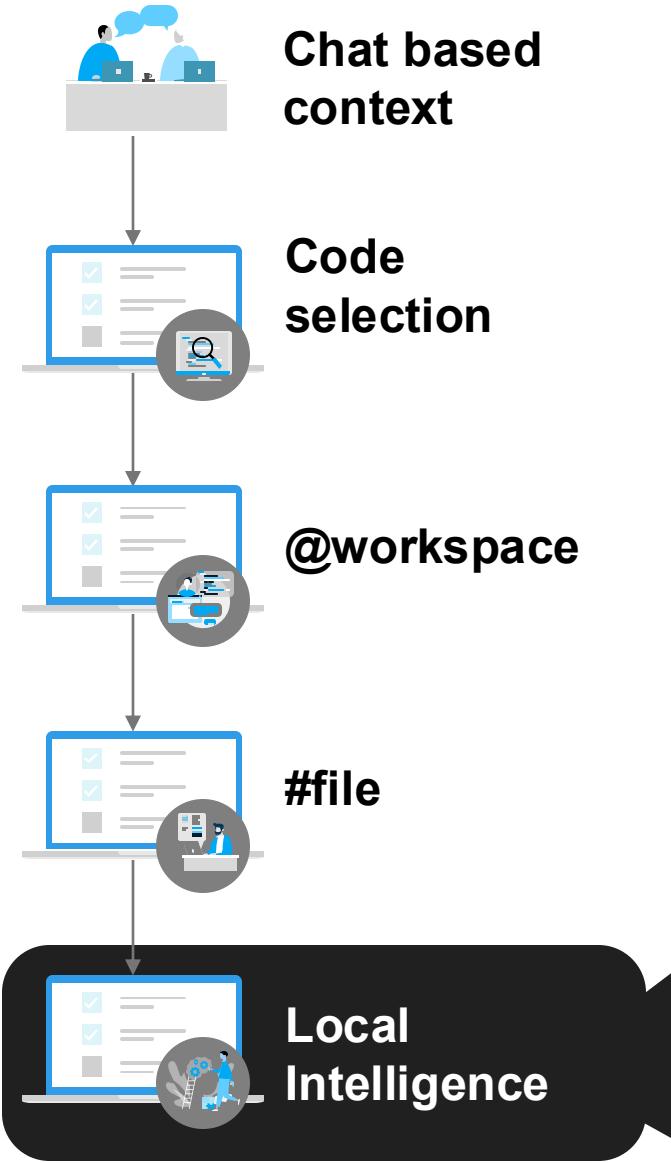
Limitations

- Not available in all IDEs but alternative options available
- Selecting too many files might still not provide the expected due to token limitations



TIP

In JetBrains you can add files by using the "+" icon at bottom of the chat



Copilot also builds local intelligence based on accepted or rejected suggestions which is used the context filter to derive better prompts for LLM

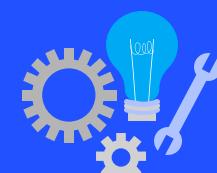


When to use

- Get code suggestions using inline prompts based on existing frameworks and code patterns available in open neighboring tabs
- Useful for generating new code in existing files and adding code comment.

Limitations

- Different context as copilot chat
- Cannot manually extend context
- Sometimes gets the context from other open tabs



TIP

Clarity of prompt and more code you open will help copilot create specific context and provide better suggestions



Demo: Context setting and result application in VS Code

Features to demonstrate:

Context setting

1. Currently active editor
2. Selected code block
3. Attach Context (paperclip) icon (Copilot Chat) or Add Files button (Copilot Edits)
4. Reference context by using chat variables (#)
5. Reference context by using symbols from the workspace (@)
6. Drag and drop editor tabs, or files or folders

Result application

1. Apply in editor
2. Insert at cursor
3. Copy
4. Insert into new file



Demo: Context setting and result application in IntelliJ

Features to demonstrate:

Context setting

1. + button to select multiple files
2. Drag & drop files
3. Right click on file and GitHub Copilot > reference file in chat
4. Keep tabs open
5. @project
6. @github

Result application

1. Copy message content
2. Insert cursor at code block

Break (15 mins)



Prompt engineering techniques

Agenda:

1. Intro and approach to prompting
2. Prompting techniques and examples
3. Prompting exercises

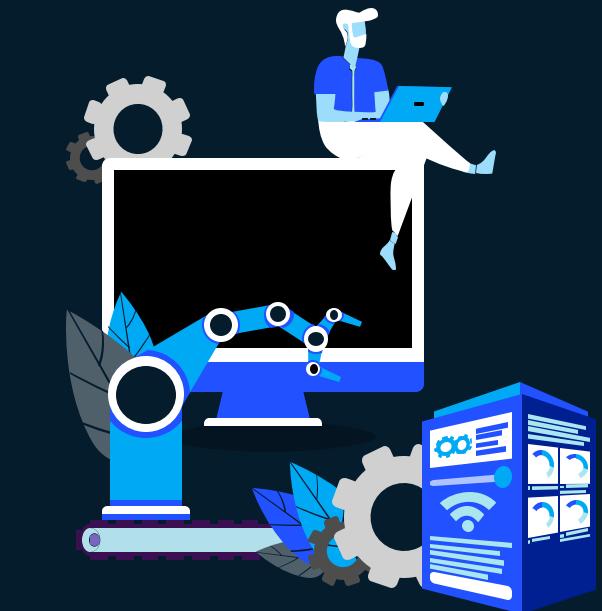


After completing this module, you will:

1. Understand how to interact with GenAI through prompting
2. Learn how to draft effective prompts
3. Understand the nuances and pitfalls when prompting GenAI



What is prompt engineering? Why is it so important?



Prompt engineering is the ability to design and implement efficient and effective inputs or prompts for AI systems to generate desired outputs

Asking the *right* questions through prompting will lead to quicker, more accurate responses, reducing frustration with GenAI tools

Prompting Techniques: How to interact with GenAI



VS



Chain-of-Thought

Providing a few chain of thought (series of intermediate steps) as exemplars in prompting to improve complex reasoning capabilities of LLMs

X-Shot Prompting

Providing some number of examples (one or few) to guide the model's behavior



Can you think of how to apply these concepts to software problems?

Chain-of-Thought Demo

Example 1: Create a simple web page

Prompt

Prompt

Prompt

I want to create a simple web page. What should I start with?

Prompt: I want to create a simple web page. What should I start with?

Answer: Let's start by creating the basic structure of an HTML document. Here's the initial code:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My Web Page</title>
  </head>
  <body>
    <!-- Content will go here -->
  </body>
</html>
```

This is the foundation of your web page. Next, we can add content to the `<body>` section.

What content should I add to the body?

Prompt: What content should I add to the body?

Answer: Let's add a title and a paragraph to the body. Here's the updated code:

```
<body>
  <h1>Welcome to My Web Page</h1>
  <p>This is a simple web page that we are building step by step.</p>
</body>
```

Now your page has some basic content. Next, we can style it to make it look better.

Prompt

How can I style the page?

Prompt: How can I style the page?

Answer: We can use CSS to style the page. Add a `<style>` block inside the `<head>` section to define the styles. Here's the updated code:

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>My Web Page</title>
  <style>
    body {
      background-color: #lightblue;
      font-family: Arial, sans-serif;
      text-align: center;
      padding: 20px;
    }
    h1 {
      color: #darkblue;
    }
    p {
      color: #darkslategray;
    }
  </style>
</head>
```

X-Shot Prompting Demo

Example 2: Write a C# method based on the description

Prompt

Prompt:

Write a C# method based on the description.

Example 1:

Task: Return the sum of two numbers.

```
csharp
public int Add(int a, int b)
{
    return a + b;
}
```

Example 2:

Task: Return the length of a string.

```
csharp
public int GetLength(string text)
{
    return text.Length;
}
```

Your turn:

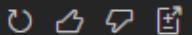
Task: Return the difference between two numbers.

Output

GitHub Copilot

Here's the C# method based on the task description:

```
public int GetDifference(int a, int b)
{
    return a - b;
}
```



Single shot prompting requires existing examples based on SW use case and functional area



Use case



Functional area

Code Generation	Code Translation	Code Testing	Code Refactoring	Code Documentation
Outline exact task Include function names and parameters in prompt	Input code must be syntactically correct Include input code language and output code language	Include testing framework Include test assertions in prompt	Input code must be syntactically correct Include refactoring improvement in prompt	AI models can explain and summarize complex code in single prompts



Full stack	Front end	Back end	Cloud	DE / DS
Framework(s) Requirements Modularity	UI framework UI component library Browser / runtime	ORM Table structure Data types	Cloud provider Cloud service IAC language	Table structures Data types Processing framework

Chain of thought prompting revolves around conversing with AI based on SW and functional area



Use case



Functional area

Code Generation

Ask AI to describe or **explain** portion of codebase before generating code

Code Translation

Generate translated code first
Iteratively improve translated output by providing feedback to model

Code Testing

Prompt AI to describe code under test before generating unit tests

Code Refactoring

Implement complex refactoring in series of prompts
Generate refactored code
Iteratively improve refactored code

Code Documentation

Ask AI to first understand code
Iteratively discuss code in conversation with AI



Full stack

Limited context of full application
Consider IDE AI tools
Generate code at function level

Front end

Generate specific components
Describe design in plain text

Back end

Generate boilerplate to define environment
Ask for sequence of steps to implement a design pattern
Refactor functions

Cloud

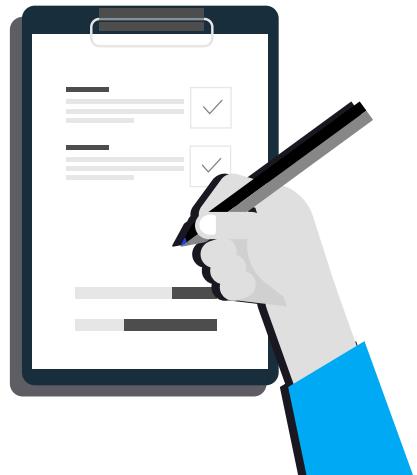
Ask AI to describe task
Ask follow-up requests depending on answer

DE / DS

Data cleaning via series of prompts
Model selection and evaluation
Pipeline design and optimization

How to craft effective prompts?

Task Specification



Developer activities



Clearly define the task or objective you want the model to accomplish

Developer actions



Provide additional limitations that matter to your task (e.g., optimize result for memory, better readability)

Ask questions rather than statements

Repeat important information

Example

“Break out the following code into separate functions for greater reusability and readability”

Pitfalls



Assuming it understands prior context every time
Asking math-related tasks

How to craft effective prompts?

Context Setting



Developer activities



Relevant context information can help the model generate accurate responses

Developer actions



Directly: Add small chunks of code to prompt for referencing
Indirectly: Open files you want to reference

Example

"Use the following ORM models for Users to query the user's specific shopping cart id"

"The tests are setup with the following pytest fixtures"

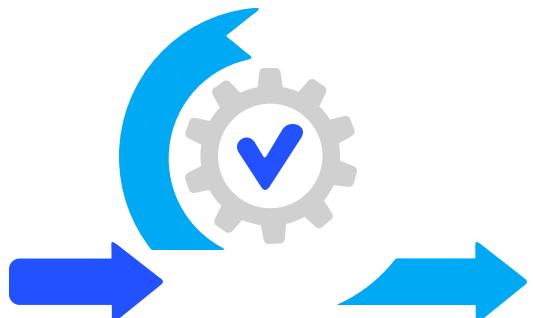
Pitfalls



Giving multiple, large amounts of code in one go
Not knowing what files to take information from

How to craft effective prompts?

Iterative Refinement



Developer activities



Analyze the model's output, identify any issues or errors, and make necessary adjustments to the prompts to obtain better results

Developer actions



Ask with incremental details to refine output
Be specific on what needs to be modified and create dialogue with the code

Example

“Split the unit tests into separate tests for success and failure”
“Use bcrypt instead of sha256 library”

Pitfalls



Creating long prompts to obtain “best” answer on the first try
Wasting time nitpicking details that can be fixed faster manually

How to craft effective prompts?

Anticipating failures

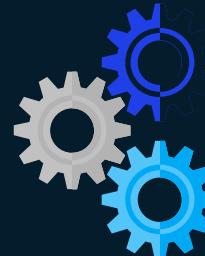


Developer activities



Anticipate potential errors or edge cases and include instructions to handle them. This helps improve the reliability and usability of the software application

Developer actions



Ensure your understanding of potential areas for unexpected failing behaviors from code and have a plan to instruct

Example

"Create a function to filter the products by category. If the query returns null, return an error message using flash"

Pitfalls



Assuming the AI will handle edge-cases automatically
Providing vague directions to handle potential errors

Examples - Good VS Bad Prompts

Using Java and React.js

	Good Prompt	Bad Prompt
Java API Prompt	Create a Java Spring boot REST API controller to handle GET requests for retrieving a list of books from a SQL Server database. Use dependency injection for database access and implement pagination for large datasets. Follow the repository pattern and include a unit test for the controller	Write a Java API
React.js Prompt	Build a React.js component that fetches customer data from the Java API using Axios and displays it in a paginated table. Use TypeScript, React hooks, and error handling with Microsoft Fluent UI for styling	Create a React component that shows data
Specific Tool Usage	Generate a Java Spring boot REST API with Swagger integration for API documentation. Ensure the API can be tested in Swagger UI	Make a Java API for documentation
State Management	Implement state management in a React.js app using Redux Toolkit for managing the customer list fetched from the API. Include asynchronous actions for API calls	Manage state in React
Error Handling	Add global exception handling in the Java API using middleware. Return meaningful HTTP status codes for validation errors, not found errors, and internal server errors	Handle errors in the API

Key Takeaways for good prompts



Structure: Well-structured, breaking down the steps required for implementation



Clarity: Clear, specific, and describes the desired code behavior, inputs, and outputs



Scope: Focuses on a specific task, function, or feature with appropriate complexity



Context: Provides necessary context, such as the framework, purpose, or specific tools



Use of Keywords: Includes relevant tool keywords to guide the LLM



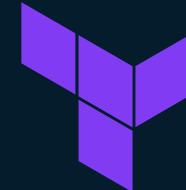
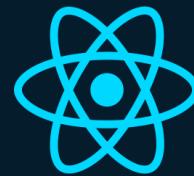
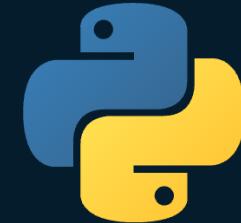
Demo: Mini Project Setup Guide

Steps to demonstrate:

1. Set up the project structure
2. Start adding functionality
3. Expose API endpoints
4. Build frontend

Follow the instructions to work on **Prompt Engineering Techniques**

CodeLab



Tips

- Use the Code Lab as a baseline but adjust prompts using your teams' specific frameworks, tools and coding standards

Custom Instructions

Agenda:

1. What are custom instructions?
2. Steps to generate custom instructions
3. Demo



After completing this module, you will:

1. Understand what are custom instructions and the benefits of using for each use case
2. Evaluate the productivity gains in terms of time, quality and efficiency
3. Create your own custom instructions files with Copilot



Understanding Custom Instructions in GitHub Copilot¹



What are custom instructions ?

- Customizable files used to guide and enhance Copilot AI capabilities
- Define the context, role, and specific behaviors for tailored assistance
- Stored in the .github folder, enabling easy reuse across projects



Benefits of Custom Instruction Files

- Gain the ability to codify and automate your expertise — freeing up time to focus on solving complex problems, mentoring your team, and driving technical excellence
- Leverage this tool to set new standards in productivity and impact within your projects



Productivity Gains

Time Saved

- Automates repetitive tasks, such as code reviews and refactoring
- Reduces time spent on generating boilerplate code

Quality Improvements

- Provides consistent code standards across teams
- Reduces debugging and rework efforts



You can create a file that automatically adds information to all questions you ask to Copilot Chat!

Collaborative Efficiency

- Scales expertise across junior and senior developers
- Accelerates SDLC timelines

Steps to generate custom instructions in GitHub Copilot

Step	Action	Example
1 Enable Custom Instructions	Turn on the "Custom Instructions" option to allow personalized suggestions.	<i>To enable custom instructions in VS Code or Visual Studio, go to GitHub Copilot settings in the editor, type 'instruction file' and toggle "Custom Instructions" on, and save your preferences</i>
2 Identify Repeating Tasks	Map out frequently needed AI assistance scenarios (e.g. preferred code styles or frameworks, example patterns to follow or avoid)	<i>Refactor code for readability and performance</i>
3 Create the Prompt File and define key attributes	Use roles: Specify AI behavior	<i>Act as a senior software DevOps engineer</i>
	Specify context: Provide a clear focus area	<i>Optimize Angular components</i>
4 Save and Leverage	Include goals: List deliverables or outcomes	<i>Generate efficient code snippets</i>
	Save in .github/copilot-instructions/ directory	
	Call directly during development for consistent, high-quality support	

Common pitfalls in writing custom instructions with GitHub Copilot

Bad usage	Example	Why avoid?
1 Referring to External Resources	<i>Always conform to the coding styles defined in styleguide.md in repo my-org/my-repo when generating code</i>	Copilot cannot reliably access or apply external resources like a style guide in your repository
2 Mandating a Specific Response Style	<i>Answer all questions in the style of a friendly colleague, using informal language</i>	Copilot's tone adapts to context and may not consistently maintain a predefined style
3 Imposing Response Length or Detail Restrictions	<i>Answer all questions in less than 1000 characters, and words of no more than 12 characters</i>	Artificial constraints can lead to incomplete or unclear responses
4 Requesting Context from Tools or Specific Modes	<i>Use @terminal when answering questions about Git</i>	Tool-specific or mode-specific instructions may not integrate well with Copilot's behavior

Effective custom instructions should focus on clear, relevant, and actionable information without relying on external references or unrealistic constraints





Demo: Custom Instructions

Features to demonstrate in VS Code:

1. Use settings
2. Use a .github/copilot-instructions.md file
3. Use prompt files

Features to demonstrate in IntelliJ:

1. Use settings
2. Use a .github/copilot-instructions.md file

Tips for providing context to GitHub Copilot

- **Provide context to your application**, tech stack and in the chat
- **Good naming convention** goes a long way in building better context
- **Provide detailed prompts** with specific libraries and references
- **Open relevant tabs** and ask it to explain the file
- **Provide sample code** if you need a specialized output
- **Use agents and slash commands** to provide wider context (e.g., @workspace, /explain)
- **Use context variables** in prompts to refer to specific files (e.g., #file, #selection)
- **Incrementally provide feedback** to help it understand your coding style



Treat copilot like a pair programmer!

Use case: domain knowledge

Agenda

- Explanation
- Live demo



Topics

- **What it is, Why** it is critical
- **Demo 1:** Translate domain knowledge into feature requirements
- **Demo 2:** Create a data model in SQL for the requirements outlined
- **Demo 3:** Create api responses in json and OpenAPI specification

Domain knowledge bridges the gap between technical expertise and real-world application

Domain knowledge

The specialized understanding of the problem space, industry, or business area



What it is

- ✓ **Concepts and terminology:** Understanding the specific terms, processes, and jargon used within the domain
- ✓ **Business rules and processes:** Knowing how things work in the relevant field, including the established procedures, regulations, and decision-making factors
- ✓ **User needs and behaviors:** Comprehending the challenges, goals, and expectations of the people who will use the software within that specific domain

Why it is critical

- ✓ **Leads to better requirements:** with a strong understanding of the domain, developers can gather more accurate and specific requirements
- ✓ **Informs design decisions:** Domain knowledge helps developers choose the right architecture, data structures, and algorithms for the specific problem
- ✓ **Enables effective testing:** Understanding the domain allows for the creation of relevant test cases that reflect real-world scenarios users might encounter



Demo: Mini Project Requirements

Requirements to demonstrate:

1. Translate domain knowledge into feature requirements
2. Create a data model in SQL for the requirements outlined
3. Create API responses in json and OpenAPI specification

Use case: software architecture

Agenda

- Explanation
- Live demo



Topics

- **What it is, Why it is critical**
- **Demo 1:** Produce user journeys based on feature requirements
- **Demo 2:** Describe the entities using class diagrams

Software architecture is the foundation for building robust, maintainable, and scalable software systems

Software Architecture
The high-level structure of a software system.



What it is

- ✓ **Structural Elements:** These are the software components, their external properties, and their relationships with other components.
- ✓ **Behavioral Elements:** These describe how the components interact and behave, defining the control flow within the system.
- ✓ **Architectural Styles or Patterns:** These are the high-level strategies that guide the organization and interaction of components, such as layered architecture, microservices, or event-driven architecture.

Why it is critical

- ✓ **Foundation for Design and Development:** The architecture serves as the blueprint for the system. It defines the high-level structure and organization of the software, guiding design and development decisions.
- ✓ **Quality Attributes:** Architecture is crucial in defining non-functional requirements like performance, scalability, reliability, security, and maintainability, which must be considered from the project's outset.
- ✓ **Communication Among Stakeholders:** The architecture provides a common language for stakeholders to discuss high-level design decisions and trade-offs.



Demo: Mini Project Diagrams

Diagrams to demonstrate:

1. Produce user journeys based on feature requirements
2. Describe the entities using class diagrams

Feedback & Suggestions

Day 1

