# GitHub Copilot Training

**Course content**
puun.ch/to/pictet-copilot

Jeremy GOBET & Joachim DESROCHES 2025

# Trainers

▶ **Jeremy Gobet**

Lecturer @HEPIA HES-SO
Web App Developer @Crafts Lab

jgobet@craftslab.ch

**Joachim Desroches** ◀

IT Architect @Kleis Technology
Software Craftsmanship Trainer

jdesroches@kleis.ch

# Agenda

▷ Theory

▷ Hands-on

**01**

Introduction

**02**

Discovering features

**03**

Refactoring

**04**

GitHub Copilot

**05**

Conclusion

# 01

# Introduction

Training objectives, previous experience and expectations from participants.

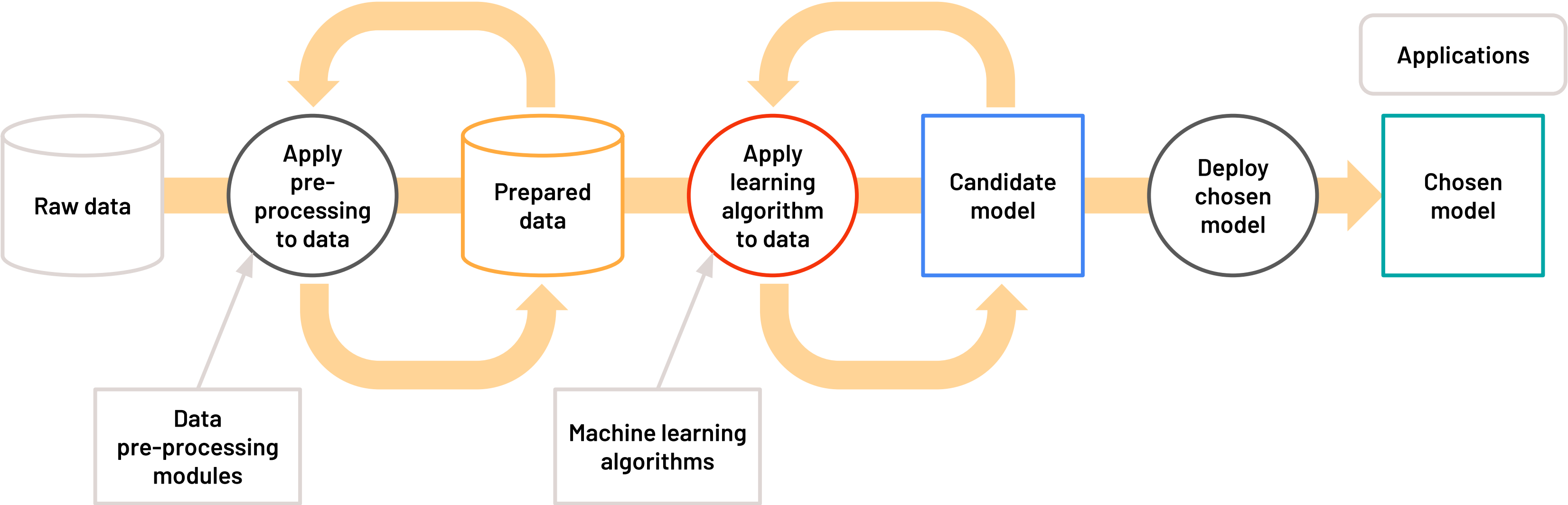
# Training objectives

**30% of theory and 70% of practice**

- To provide insights about what GitHub Copilot does and does not through best practice usage considerations.
- To experiment with GitHub Copilot in order to forge an opinion of when and why to use it.
- User from the early beginning. We will share with you our recommendations.

# What are your expectations ?

# Machine Learning process



Raw data

Apply pre-processing to data

Prepared data

Apply learning algorithm to data

Candidate model

Deploy chosen model

Chosen model

Applications

Data pre-processing modules

Machine learning algorithms

*Jeremy Gobet & Joachim Desroches. Github Copilot. 2025*

# GitHub Copilot provides contextualized assistance throughout the software development lifecycle

GitHub
Copilot

*https://resources.github.com/learn/pathways/copilot/essentials/essentials-of-github-copilot*

# GitHub Copilot integrations

## IDE Integration

- **VS Code**
- Visual Studio
- JetBrains IDEs
- Xcode
- Neovim

## GitHub Integration

- Code review
- GitHub actions
- Issues
- Third-party extensions
- ...

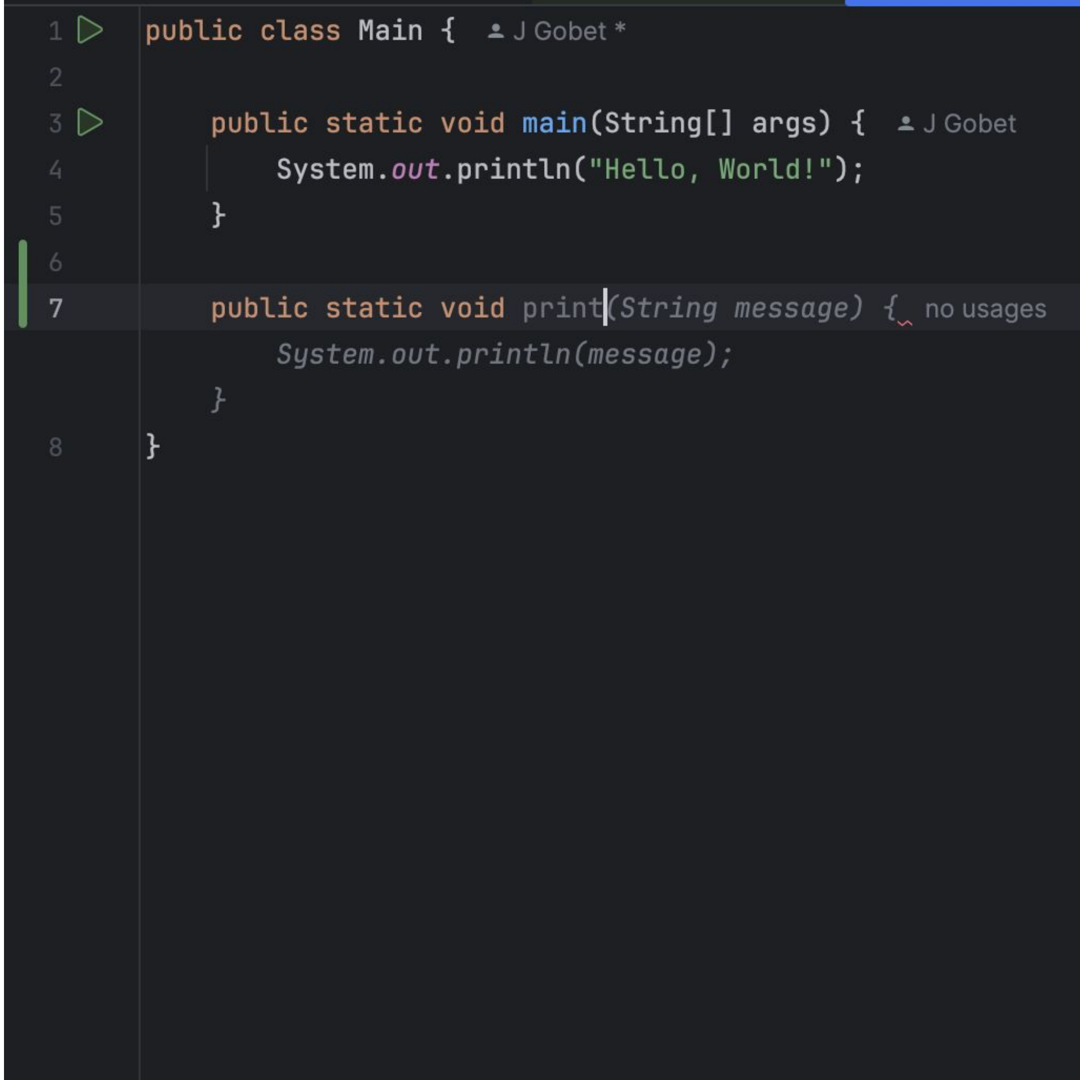
## Other integrations

- Android application
- IOS application
- CLI
- ...

*https://github.com/features/copilot*

# What is GitHub Copilot ?

- **Code Completion**
- Chat assistance
- Context-aware



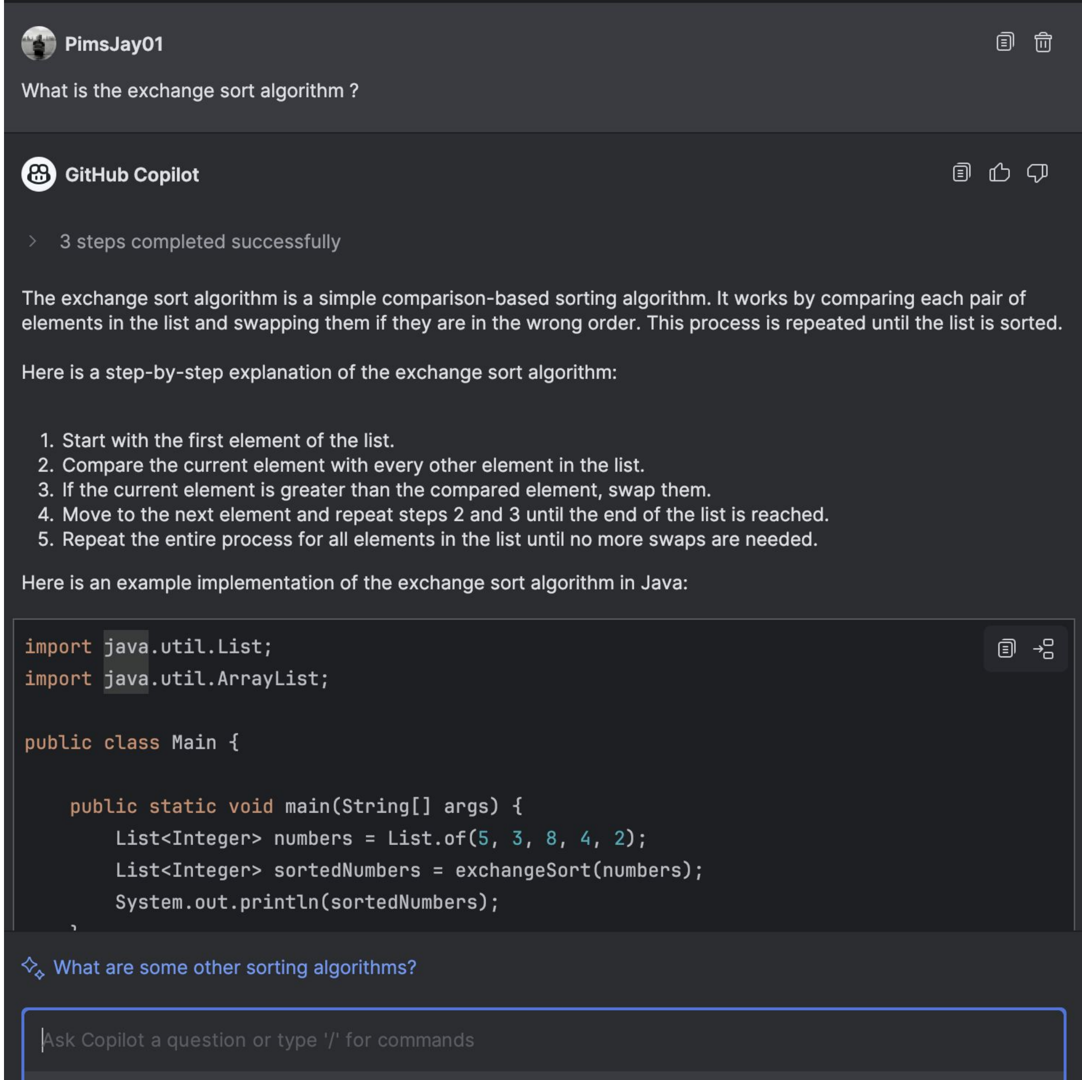
```
public class Main {

    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }

    public static void print(String message) {
        System.out.println(message);
    }
}
```

# What is GitHub Copilot ?

Code Completion

**Chat assistance**

Context-aware



**PimsJay01**

What is the exchange sort algorithm ?

**GitHub Copilot**

The exchange sort algorithm is a simple comparison-based sorting algorithm. It works by comparing each pair of elements in the list and swapping them if they are in the wrong order. This process is repeated until the list is sorted.

Here is a step-by-step explanation of the exchange sort algorithm:

1. Start with the first element of the list.
2. Compare the current element with every other element in the list.
3. If the current element is greater than the compared element, swap them.
4. Move to the next element and repeat steps 2 and 3 until the end of the list is reached.
5. Repeat the entire process for all elements in the list until no more swaps are needed.

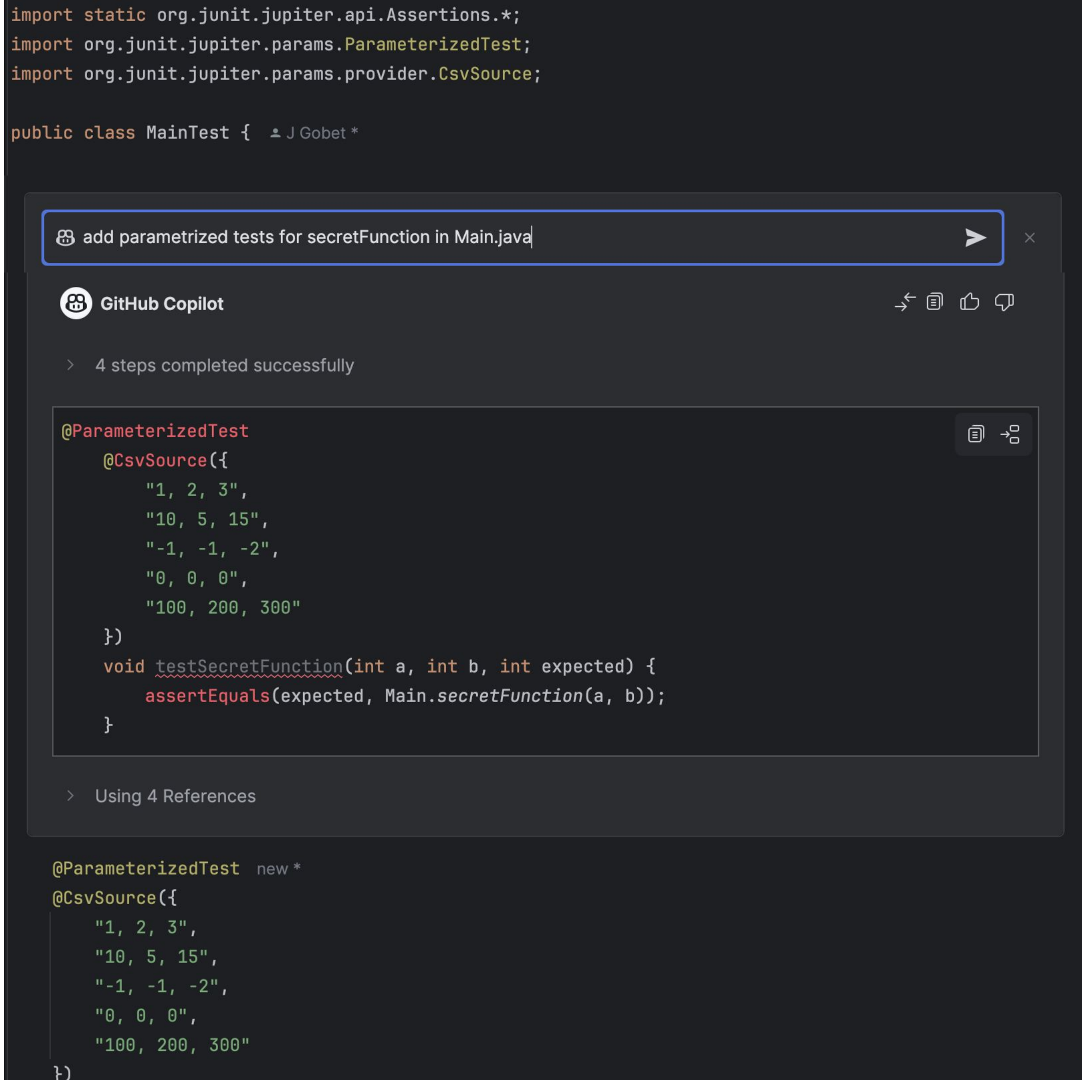
Here is an example implementation of the exchange sort algorithm in Java:

```java
import java.util.List;
import java.util.ArrayList;

public class Main {

    public static void main(String[] args) {
        List<Integer> numbers = List.of(5, 3, 8, 4, 2);
        List<Integer> sortedNumbers = exchangeSort(numbers);
        System.out.println(sortedNumbers);
```

What are some other sorting algorithms?
Ask Copilot a question or type '/' for commands

# What is GitHub Copilot ?

- Code Completion
- Chat assistance
- **Context-aware**



```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;

public class MainTest {  J Gobet *
```

add parametrized tests for secretFunction in Main.java

4 steps completed successfully

```
@ParameterizedTest
    @CsvSource({
        "1, 2, 3",
        "10, 5, 15",
        "-1, -1, -2",
        "0, 0, 0",
        "100, 200, 300"
    })
    void testSecretFunction(int a, int b, int expected) {
        assertEquals(expected, Main.secretFunction(a, b));
    }
```

Using 4 References

```
@ParameterizedTest  new *
@CsvSource({
    "1, 2, 3",
    "10, 5, 15",
    "-1, -1, -2",
    "0, 0, 0",
    "100, 200, 300"
})
```

# 02

# Discovering features

Quick exploration of all the ways
to trigger Copilot

# Discovering features

Open the provided **Python** project
and generate suggestion from Copilot with:

1. a comment
2. a function signature
3. the in-line prompt in code
4. multiple suggestions
5. chat
6. contextual menu

https://github.com/copilot-training-2025/discovering-features-python

# Discovering features (1/2)

Open the provided **Python** project and generate suggestion from Copilot with:

1. **a comment**:

```python
# method to compute a bubble sort
```

2. **a function signature**:

```python
def calcul_sum_for_even_values(number_list):
```

3. **the in-line prompt in code**:

   Select the previous method, open in-line prompt , and ask:
   "refactor to use list comprehensions"

# Discovering features (2/2)

Still on the provided **Python** project, use Copilot's:

4. **multiple suggestions**:

   Open the suggestions pane, and then prompt for:
   `# memoized fibonacci function`

5. **chat**:

   Open the chat, and ask:
   **"write parameterized tests with five examples for a generic sorting function"**

6. **contextual menu**:

   Select the bubble sort and in the contextual menu, click
   **"Simplify This"** and then **"Generate Docs"** or "**/docs**" directly in prompt

# 03

# Refactoring

Experimentation of how to refactor code using GitHub Copilot.

# The 9 rules of object calisthenics

**Guidelines to keep your code maintainable, readable, reusable and testable**

- one level of indentation per method.
- don't use the ELSE keyword.
- wrap all primitives and Strings in classes.
- first class collections.
- one dot per line.
- don't abbreviate.
- keep all classes less than 50 lines.
- no classes with more than two instance variables.
- no getters or setters → Tell. Don’t ask.

https://github.com/copilot-training-2025/tictactoe-python

# Refactoring with GitHub Copilot

## Guidelines to keep your code maintainable, readable, reusable and testable

- ☐ one level of indentation per method.
- ☐ don't use the ELSE keyword.
- ☐ wrap all primitives and Strings in classes.
- ☐ first class collections.
- ☐ one dot per line.
- ☐ don't abbreviate.
- ☐ keep all classes less than 50 lines.
- ☐ no classes with more than two instance variables.
- ☐ no getters or setters → Tell. Don’t ask.

## GitHub Copilot features to use at least once

### Generate suggestions

- ☐ from function signature
- ☐ from in-line or chat prompt
- ☐ from multiple suggestion pane

### Use command from contextual menu

- ☐ Explain This
- ☐ Simplify This
- ☐ Generate Tests
- ☐ Generate Docs

## 04

# GitHub Copilot

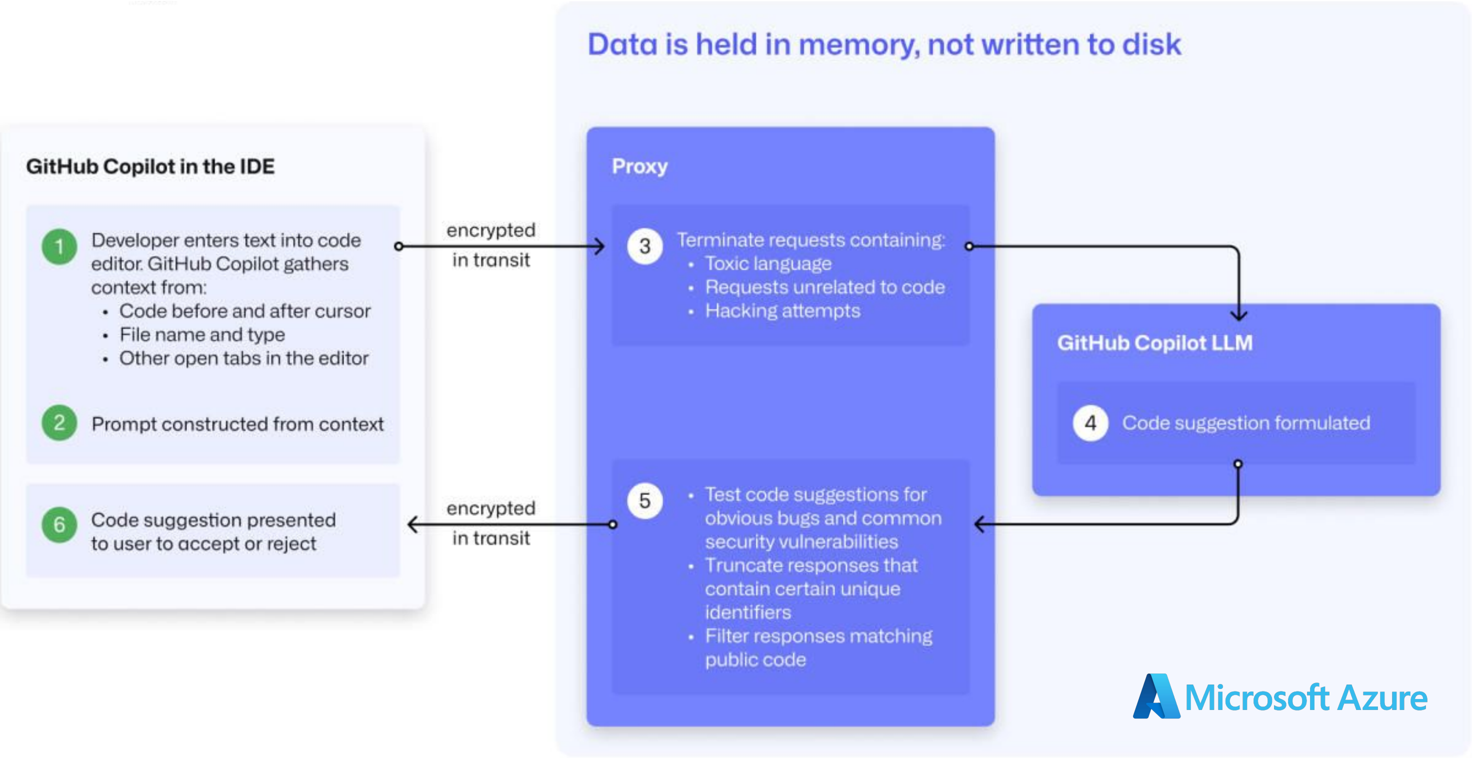
Presentation of GitHub Copilot: capabilities, forces and weakness.

# GitHub Copilot keeps your data secure on trusted Microsoft Azure infrastructure

GitHub
Copilot

*https://resources.github.com/learn/pathways/copilot/essentials/essentials-of-github-copilot*

# Lifecycle of a code suggestion



Data is held in memory, not written to disk

**GitHub Copilot in the IDE**

1. Developer enters text into code editor. GitHub Copilot gathers context from:
    - Code before and after cursor
    - File name and type
    - Other open tabs in the editor
2. Prompt constructed from context

encrypted in transit

**Proxy**

3. Terminate requests containing:
    - Toxic language
    - Requests unrelated to code
    - Hacking attempts

**GitHub Copilot LLM**

4. Code suggestion formulated

5.
    - Test code suggestions for obvious bugs and common security vulnerabilities
    - Truncate responses that contain certain unique identifiers
    - Filter responses matching public code

encrypted in transit

6. Code suggestion presented to user to accept or reject

Microsoft Azure

*https://resources.github.com/learn/pathways/copilot/essentials/how-github-copilot-handles-data*

# How Copilot addresses plagiarism

### Models learn to avoid recitation

Different techniques help models generate new code rather than output code from the training data.

### Duplication detection filter

Checks suggestions against public repositories for matches of 65 or more lexemes.

### Contractual protection

GitHub and Microsoft provide indemnity. But only if duplication detection filter is enabled.

*https://resources.github.com/learn/pathways/copilot/essentials/establishing-trust-in-using-github-copilot/*

# Our findings

**No plan for an on-premises version of Copilot**

But GitHub declares about Copilot that they do not

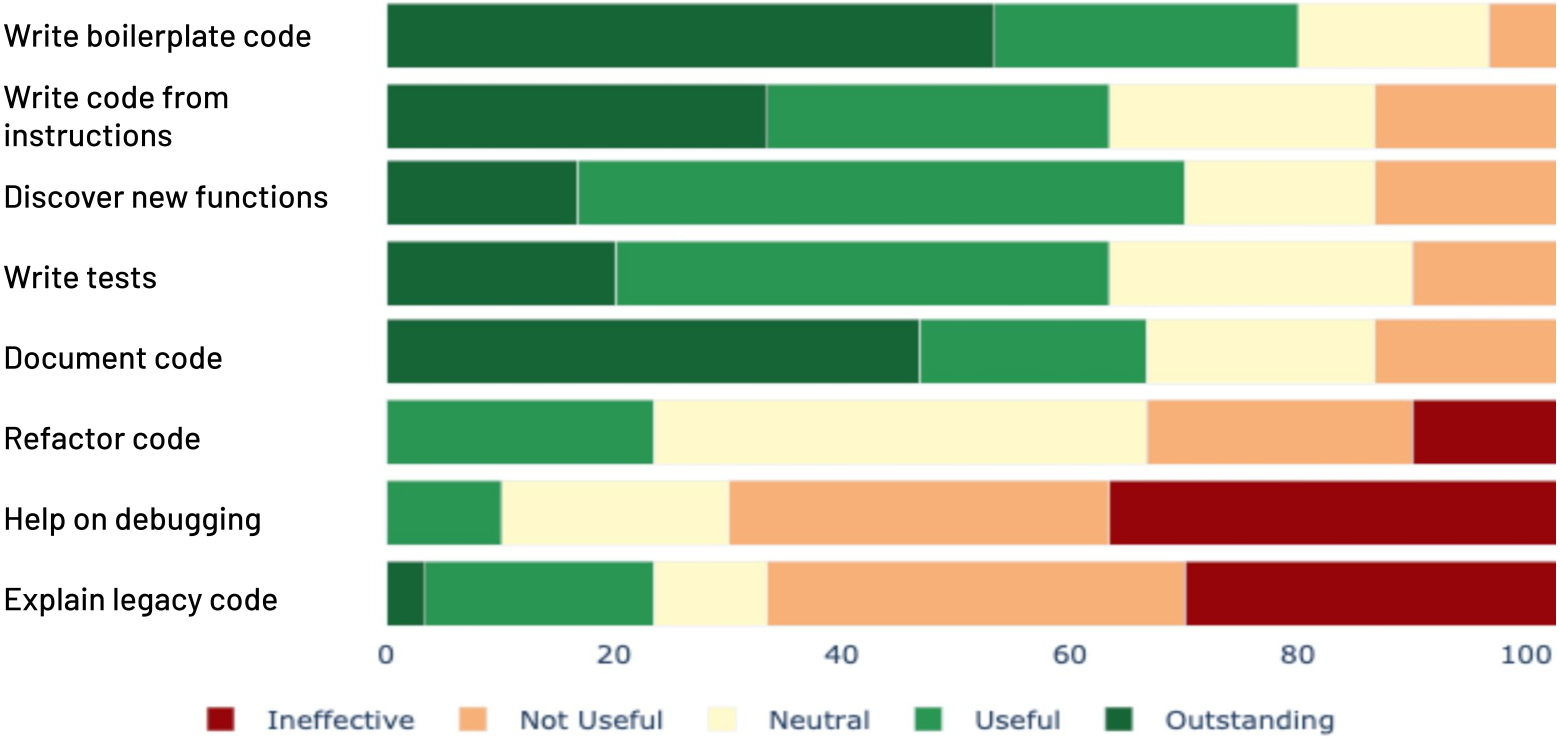
- ❌ retain user content (e.g. code from user's editor)
- ❌ use code in context to improve the model
- ❌ retain suggestions produced by Copilot
- ❌ send information to OpenAI
- ❌ own the code generated by Copilot

# GitHub Copilot handles the tasks developers dislike

**GitHub**
Copilot

*https://resources.github.com/learn/pathways/copilot/essentials/essentials-of-github-copilot*

# Usefulness on different tasks



*Prometeia. Alessandro Benetti & Michele Filannino. GitHub Copilot: a systematic study. 29-30-.05.2024*

# Integration with documentation

"**GitHub Copilot's integration with our documentation** allows our engineers to ask specific questions of our documentation, instead of searching by keyword and scanning the results for what they need. It's a more natural way of interacting with technical content **that's saving our developers time and effort**."

**Jun Li**

Engineering Manager at Lyft, american company offering mobility as a service

*https://resources.github.com/learn/pathways/copilot/essentials/essentials-of-github-copilot*

# Our findings

**Unfortunately Copilot is not going to meetings for us yet**

But GitHub Copilot could be good for

- Repetitive or boilerplate code
- Keeping focus (avoiding context-switch)
- Writing tests (at least the “given” part)
- Explaining error messages and suggesting fixes
- Onboarding with new code libraries
- Generating mockup data

**Try it yourself to find how it can help you**

# GitHub Copilot boosts speed and efficiency

GitHub
Copilot

*https://resources.github.com/learn/pathways/copilot/essentials/essentials-of-github-copilot*

# Feeling versus measure

## 56 min

### Time savings

In a large-scale trial by the UK government, developers answered in a survey that they saved an average of 56 min per day using AI tools.*

## 20 %

### Feel productive

Developers allowed to use AI tools to fix issues believed IA had speed them up by 20%. **

## 19 %

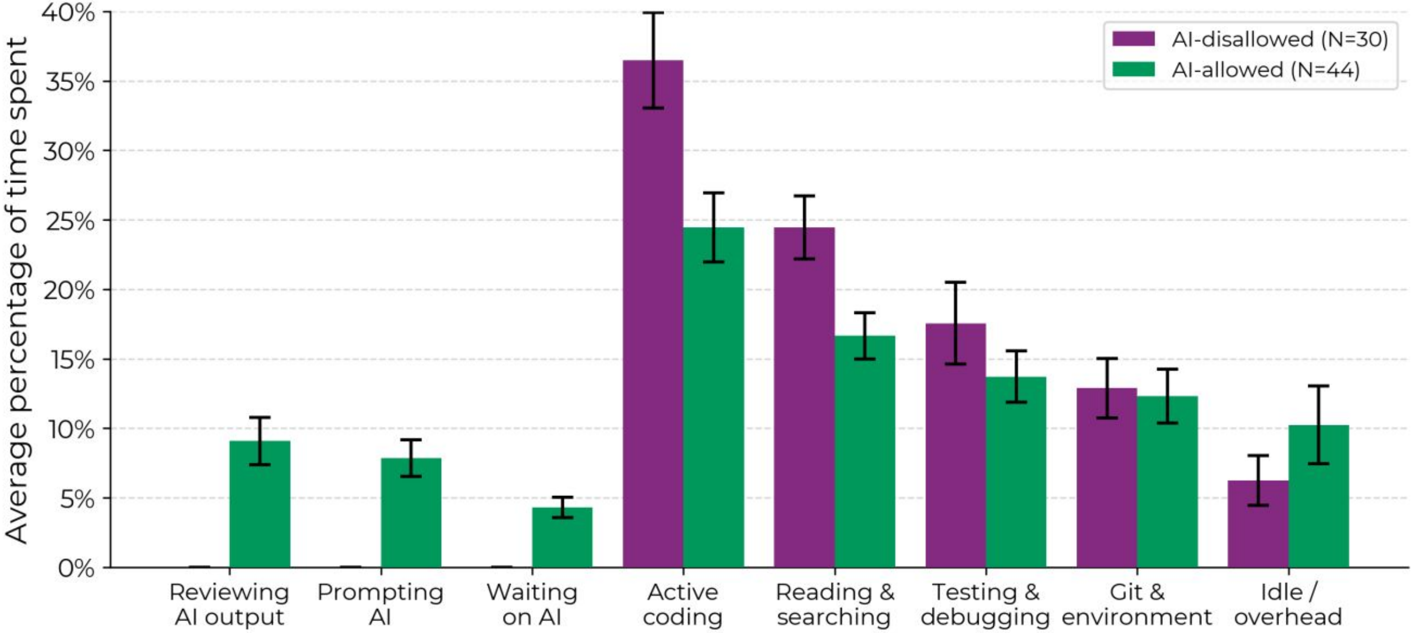
### Longer with IA

... but in comparison with the developers without AI tools, they ultimately take 19% longer to complete issues.**

* "AI coding assistant trial: UK public sector findings report", UK Government, 2025. Available at: https://www.gov.uk/government/publications/ai-coding-assistant-trial/ai-coding-assistant-trial-uk-public-sector-findings-report

** Joel Becker, et al. "Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity", 2025. Available at: https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/

# Average percentage of time spent per activity



Joel Becker, et al. "Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity", 2025. Available at: https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/

# GitHub Copilot’s efficiency

**85%**

**Code is edited**

Only 15% of
AI-generated code used
without edits.*

**<2%**

**Issues resolved**

AI can resolve less than
1.96% of real GitHub
issues.**

*"AI coding assistant trial: UK public sector findings report", UK Government, 2025. Available at: https://www.gov.uk/government/publications/ai-coding-assistant-trial/ai-coding-assistant-trial-uk-public-sector-findings-report

**Carlos E. Jimenez, et al. "Can Language Models Resolve Real-World GitHub Issues?", Princeton University & University of Chicago, 2024. Available at: https://arxiv.org/html/2310.06770v3

# Our findings

When AI is allowed, developers spend less time actively coding and searching for/reading information, and instead spend time prompting AI, waiting on and reviewing AI outputs, and idle

Unfortunately none of the studies uses DORA metrics such as **Lead Time for Changes** or **Deployment Frequency** which seem much more appropriate to measure team performance.

**We can not confirm increase or decrease in productivity.**

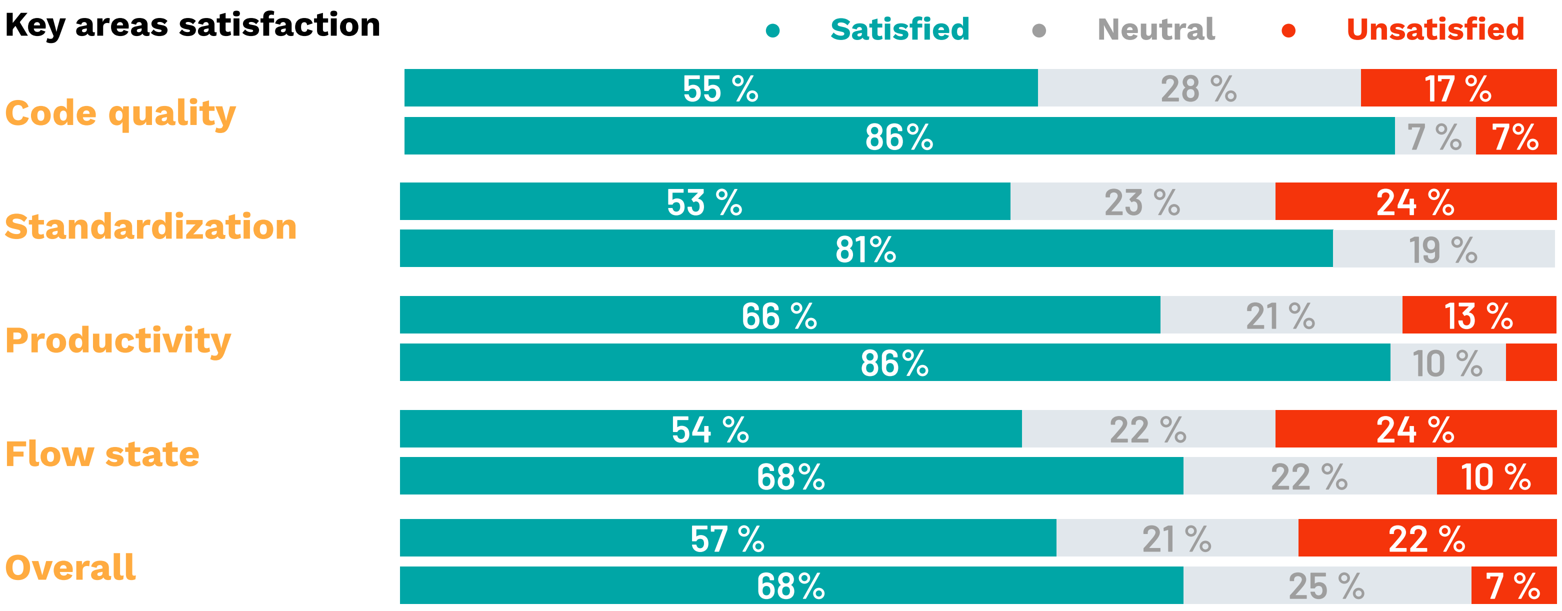
GitHub Copilot can
help ~~attract~~
~~and retain~~ talent
improve coding satisfaction

GitHub
Copilot

*https://resources.github.com/learn/pathways/copilot/essentials/essentials-of-github-copilot*

# Developer satisfaction

## Satisfaction with key areas of job <u>without</u> and <u>with</u> Copilot



*Slalom. Test of productivity with Github Copilot. 2024*

# Our findings

Like every tool, learning to use it **takes some time** and personal investment.

The return on time invested is good for some activities.

- **Might not attract your next genius dev, but can help with developer satisfaction**

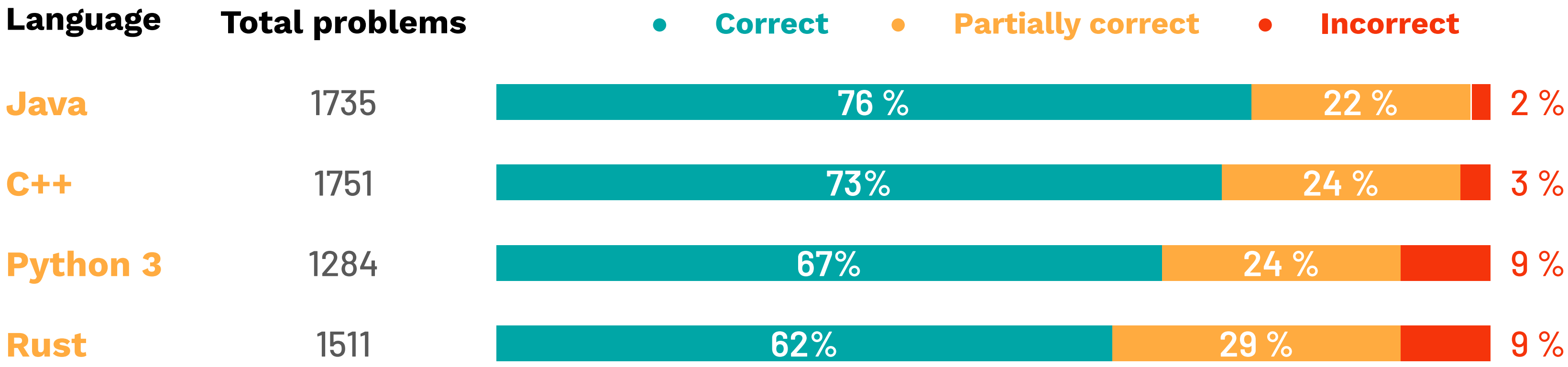
# GitHub Copilot can improve code quality

GitHub
Copilot

*https://resources.github.com/learn/pathways/copilot/essentials/essentials-of-github-copilot*

# Support for programming languages

**Submission results for each language on LeetCode problems**



| Language | Total problems | ● Correct | ● Partially correct | ● Incorrect |
|---|---|---|---|---|
| **Java** | 1735 | 76 % | 22 % | 2 % |
| **C++** | 1751 | 73% | 24 % | 3 % |
| **Python 3** | 1284 | 67% | 24 % | 9 % |
| **Rust** | 1511 | 62% | 29 % | 9 % |

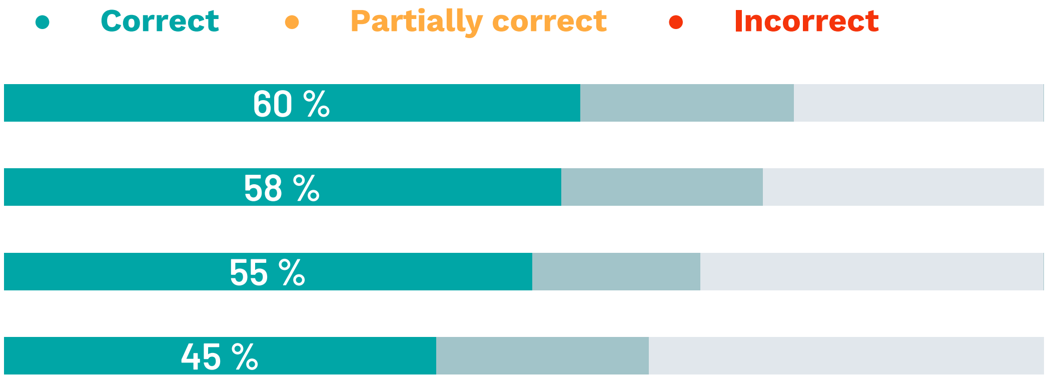
“For each problem, we asked Copilot to generate as many suggestions as possible and tested all of them [...] Here we consider Copilot’s solution for the problem to be Correct if Copilot generated at least one suggestion that passes all the tests“

*Ilja Siros, Dave Singelée & Bart Preneel. GitHub Copilot: the perfect Code compLeeter?. 17.06.2024*

# Support for programming languages

**Top Copilot proposal is not always the best one.**

| Language | Total problems |
|---|---|
| **Java** | 1735 |
| **C++** | 1751 |
| **Python 3** | 1284 |
| **Rust** | 1511 |



---

“We concluded that the code generated by Copilot is more efficient, both time- and memory-wise, than the code **written by the average human** (user on LeetCode platform).”

*Ilja Siros, Dave Singelée & Bart Preneel. GitHub Copilot: the perfect Code compLeeter?. 17.06.2024*

# More time available for quality ?

”Many participants reported reallocating the time gained to enhance the **quality** of their products further by focusing on rigorous **testing**, refining **documentation**, or dedicating effort to areas of the project that could benefit from manual oversight”

**Authors of the study**

People from Prometeia, provider of advisory services, tech solutions and research insights.

*Prometeia. Alessandro Benetti & Michele Filannino. GitHub Copilot: a systematic study. 29-30-.05.2024*

# Our findings

**More time for quality <u>yes but</u>**

- Will not change your company culture (craftsmanship vs assembly line)
- May be a kiss cool effect
- Tends to produce code similar to the existing codebase: good or bad

# GitHub Copilot does not think for you

**GitHub Copilot & LLMs do not understand or interpret meaning** but rather predict words that are likely to follow a given sequence. They operate by recognizing and generating text based on patterns they have learned from vast amounts of data during training.

**GitHub Copilot focuses on the syntax of the code** rather than the underlying semantics or meaning.

# A limited world

**Chat models:** GPT 5.1, Claude Sonnet 4.5, Gemini 2.5 Pro, Grok (preview) etc.

**Code completion model:** GPT-4.1 Copilot

The codebase knowledge of Copilot is limited to the training date at June 2025

**This might cause the suggestion of deprecated methods**
for libraries that change significantly over time.

Java SE 22 – Mars 2024 → Last Java's version: SE 25

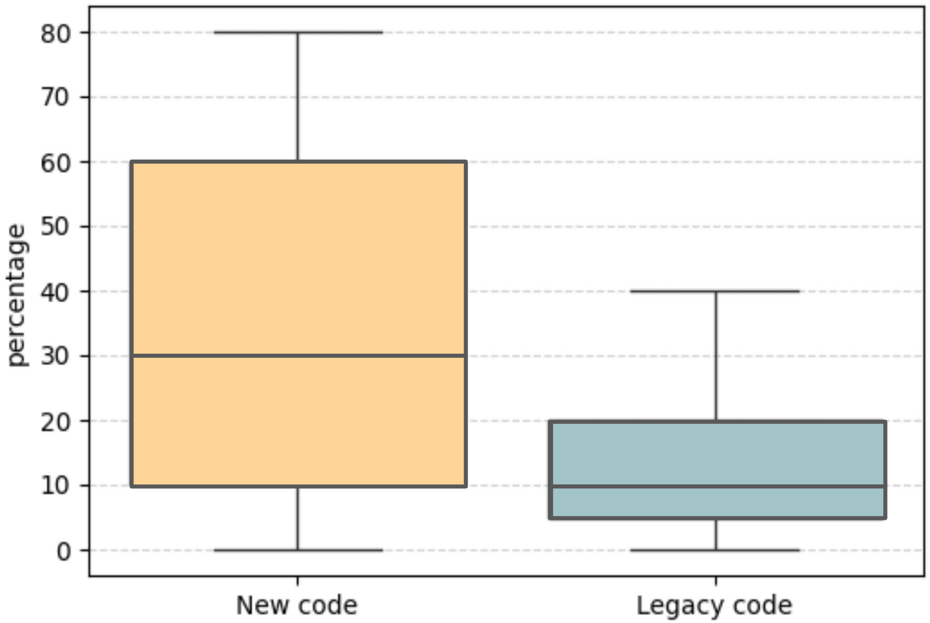
Python 3.12 – October 2023 → Last Python's version: Python 3.14

GitHub Copilot's knowledge base seems to be **updated every 6 months**.

# Main drawbacks

- **Performance with large or legacy codebases presents challenges**
- Conflicts between Copilot and IDE support
- Suggests repetitive code.

**Percentage of code lines written by Copilot**



*Prometeia. Alessandro Benetti & Michele Filannino. GitHub Copilot: a systematic study. 29-30-.05.2024*

# Generated code security

**45%**

**OWASP Top 10**

AI introduced an OWASP top 10 security vulnerability in 45% of the test cases.

**72%**

**Java**

AI introduced an OWASP top 10 security vulnerability in 72% of the test cases when writing java code.

**+38%**

**Other languages**

Other major languages didn’t fare much better:

- Python: **38%**
- JavaScript: **43%**
- C#: **45%**

**100B**

**Parameters ?**

Security pass numbers do not change from 20B to 100B parameter models.

J. Wessling, "GenAI Code Security Report", Veracode, 2025. Available at: https://www.veracode.com/blog/genai-code-security-report/

# Our findings

Copilot is trained on public code of varying quality

- **You remain accountable for the final code.**
  Don't blindly trust suggestions made by Copilot

Copilot will imitate your bad patterns and potentially duplicate the vulnerabilities in your code base.

- **Review code regularly**
- **Include security analysis in continuous integration**
- **Train staff, especially juniors**

# 05

# Conclusion

Recommendations to use Copilot.

# Daily routine with GitHub Copilot

### You are  accountable

You should understand code before running it and use Copilot to enhance, not replace, your problem-solving skills.

### Prompt effectively

Ask good question prompts as minor changes can lead to different outcomes, ensuring the input structure is clear to Copilot.

### Iterative use

Employ Copilot iteratively by creating small code snippets and gradually building on them for comprehensive solutions.

### Copilot is not an architect

Define design by yourself. Use Copilot only to write more quickly the code of your predetermined solution.

### Rubber duck on steroïds

Try to engage Copilot in discussions to refine your ideas and benefit from creative solutions and suggestions.

### Keep related files open in editor

Keep only relevant files open to help Copilot in understanding context due to its limited context window capability.

# Training conclusion

**Who here doesn't use Google or Stack Overflow ?**

- Will not replace human
  → Just a new tool in our development toolbox
- Provides enough value for its price
  → Copilot deserves to be experimented
- Everybody uses it differently and
  Everybody has an opinion
  → Try it by yourself
- Many different integrations getting better
  → Not yet on the full scope of software lifecycle

**A revolution may be underway**

# What next ?

## MCP

Model Context Protocol - Allows AI applications to connect to external systems providing *tools*, *resources* and *prompts*.

## Copilot Agent mode

New agent mode capable of iterating on its own code, detecting errors and correcting them.
(only on VS Code)

## Local config file

Add local configuration file in your project to give general instructions to Copilot.

## IDE Popularity for IA

1. VS Code + Copilot
2. Cursor (VS Code fork)
3. JetBrains + Copilot

*https://github.blog/news-insights/product-news/universe-2024-previews-releases*

# What use should you make of Copilot
# in your context ?

jgobet@craftslab.ch

jdesroches@kleis.ch

# R.O.T.I

**Return on time invested**

From 1 → To 5

# Annexes

# Alternatives

**Local IA model for autocompletion** available in following IDE

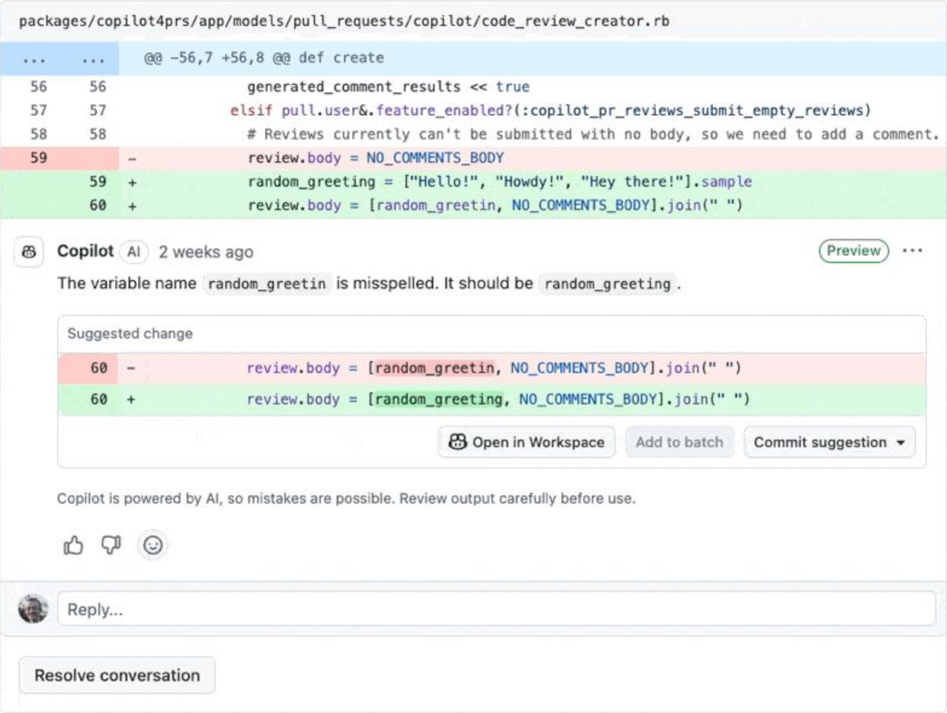
- **JetBrains In-IDE AI Assistant** and AI Service
- **Cursor IA IDE**

**Other IA Assistants** available on JetBrains Marketplace

- Blackbox
- Tabnine
- Sourcery

# GitHub Copilot in code reviews

- **Request review by Copilot**
  or configure automatic reviews
  for every pull-request
- **Copilot attach its comments**
  to specific lines of your code,
  including one-click fixes
- **Copilot Workspace**
  allows to refine and validate
  Copilot’s suggestions in the context
  of the pull request
- **Copilot describes pull request**
  with more details and highlight
  the most important areas for review



`packages/copilot4prs/app/models/pull_requests/copilot/code_review_creator.rb`

```
@@ -56,7 +56,8 @@ def create
              generated_comment_results << true
            elsif pull.user&.feature_enabled?(:copilot_pr_reviews_submit_empty_reviews)
              # Reviews currently can't be submitted with no body, so we need to add a comment.
-             review.body = NO_COMMENTS_BODY
+             random_greeting = ["Hello!", "Howdy!", "Hey there!"].sample
+             review.body = [random_greetin, NO_COMMENTS_BODY].join(" ")
```

**Copilot** AI 2 weeks ago

The variable name `random_greetin` is misspelled. It should be `random_greeting`.

Suggested change

```
-             review.body = [random_greetin, NO_COMMENTS_BODY].join(" ")
+             review.body = [random_greeting, NO_COMMENTS_BODY].join(" ")
```

Copilot is powered by AI, so mistakes are possible. Review output carefully before use.

*https://github.blog/changelog/2024-10-29-github-copilot-code-review-in-github-com-public-preview, 29.10.2024*