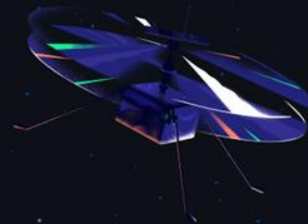




GitHub Copilot Developer Training

Andrew Scoppa

Let's build from here



Resources

- [Getting started with GitHub Copilot](#)
- [Configuring GitHub Copilot in your environment](#)
- [Insider newsletter digest: How to use GitHub Copilot](#)
- [Video - Get Started with the Future of Coding: GitHub Copilot](#)
- [Tutorial: GitHub Copilot and VS Code](#)
- [Copilot Exercises](#)



Exercises Track

Copilot Workshop



Workshop for those who want to see how to build web application

Primary IDE: VS Code
Primary OS: Independent

Copilot Workshop



Workshop for those who want to see how to build Java applications

Primary IDE: IntelliJ
Primary OS: Independent

Copilot Workshop



Workshop for those who want to see how to build .NET applications

Primary IDE: Visual Studio
IDE
Primary OS: Windows



AGENDA

GitHub Copilot - Introduction

Best practices & Prompt Engineering

Coding

Secure coding

Wrap-up, Q&A

Workshop (1 - 2 hours long)



Outcome of this training

You will achieve...

- ✓ Get answers to specific use case scenario questions
- ✓ Increase existing Copilot skills by following a specific workshop tutorial catered to your needs
- ✓ Learn in-depth tips and tricks and best practices on how to best utilize GitHub Copilot



GitHub Copilot

- An intelligent pair programmer
- Draws context from comments & code to suggest individual lines and whole functions
- Powered by OpenAI Codex
 - Copilot uses a transformative model
 - Trained on large datasets to ensure accuracy
- Available as extensions to popular IDEs
- Programming Languages and Technology available in Public code base all are supported

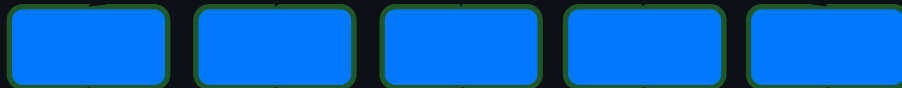
```
TS sentiment.ts  -GO write_sql.go  parse_expense

1  #!/usr/bin/env ts-node
2
3  import { fetch } from "fetch-h2";
4
5  // Determine whether the sentiment of
6  // Use a web service
7  async function isPositive(text: string) {
8      const response = await fetch(`http://
9          method: "POST",
10         body: `text=${text}`,
11         headers: {
12             "Content-Type": "application/x-www
13         },
14     });
15     const json = await response.json();
16     return json.label === "pos";
17 }
```

Whether you're a software engineer or a hardware engineer, the process can be broken down into two steps:

- 1) Design
- 2) Implementation

Design



The first step is driven by you, the second step is where Copilot can assist you with the development effort.

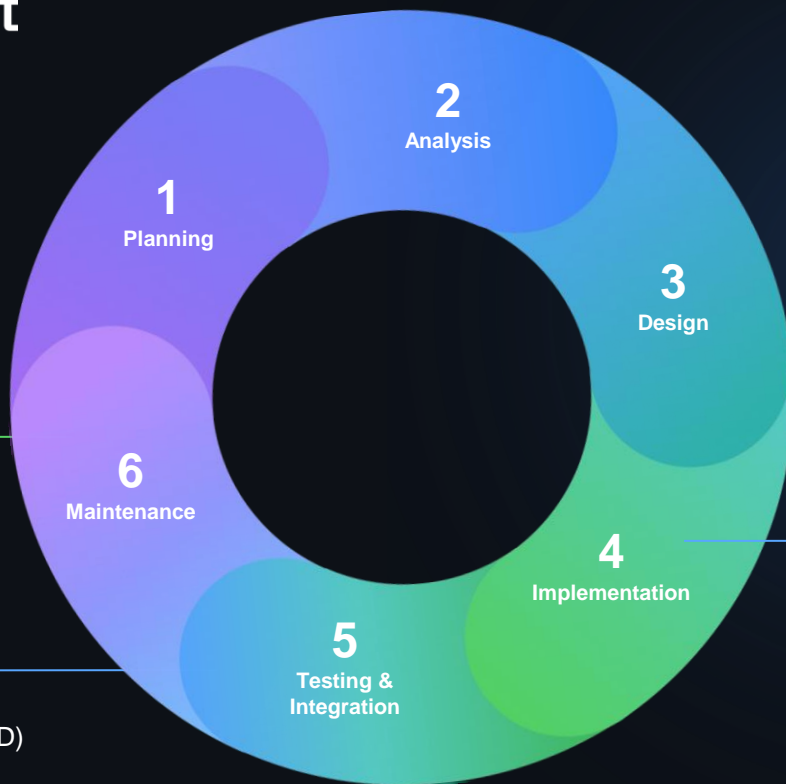


IMPLEMENTATION





Helps developers stay
in the flow throughout
the entire SDLC



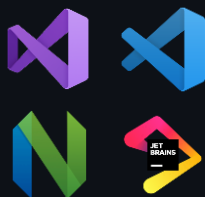
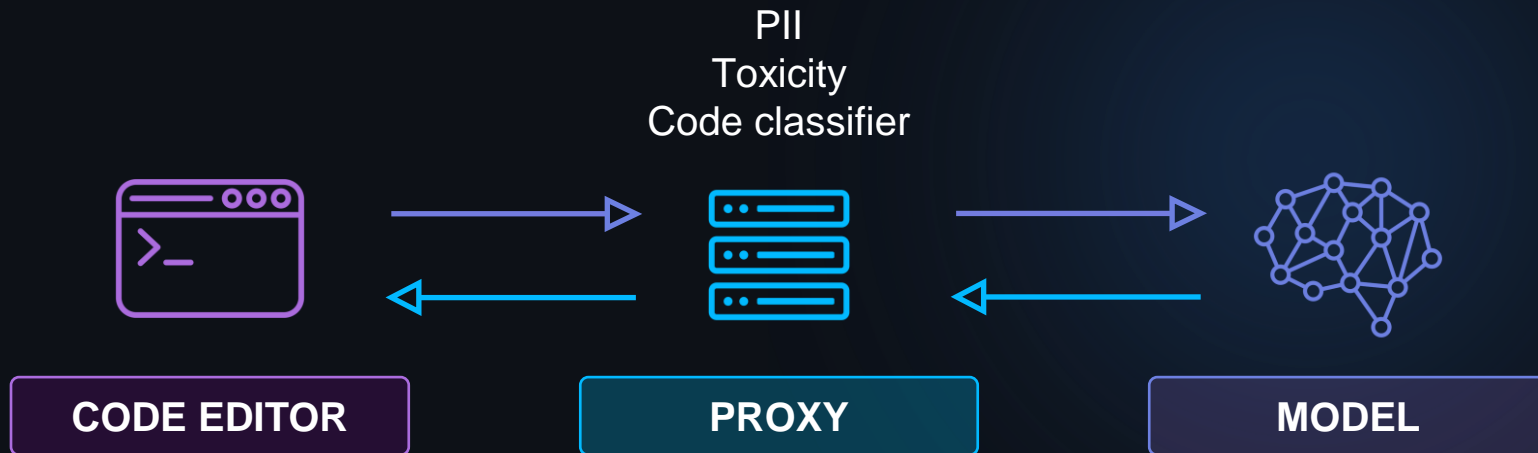
Refactoring code (code translate)
Reviewing code (code explain)
Documentation

Unit testing (TDD and BDD)
Finding code errors
Debugging
Code review
AI Pull Requests

Convert comments to code
Autofill for repetitive code
Show alternatives



Data flow through the Copilot ecosystem



PII
Toxicity
Code classifier
Code quality
Duplicate detection



Coding

GH Copilot vs GH Copilot Chat

Prompt Engineering

Techniques

GH Copilot Chat Commands

Block Files

Demos

When to use Copilot vs Copilot Chat

Copilot

Direct Code Writing

Seamless IDE Integration

Solo Development

Copilot Chat

In-Depth Assistance

Learning & Teaching

Collaborative Scenarios



What is a Prompt?



In the context of Copilot, a prompt is a piece of code or natural language description that is used to generate code suggestions. It is the input that Copilot uses to generate its output.

- Copilot

What is Prompt Engineering?



Prompt engineering is the process of designing and creating high-quality prompts that can be used to generate accurate and useful code suggestions with Copilot.

- Copilot

Why is Prompt Engineering Important?



- Maximizes utility of Copilot
- Minimizes repeated iterations
- Determines code suggestion quality
- Skill prompts guide Copilot to understand context & nuances
- Refined interactions reduces misunderstandings

Techniques



**Zero-Shot
Prompting**



**One-Shot
Prompting**




**Few-Shot
Prompting**

Techniques

Zero-Shot Programming with Copilot

 abelberhane

Write me unit tests for all of the operations in my calculator.js file.

 GitHub Copilot

Sure! Here's an example of a test suite that covers all of the operations in a `calculator.js` file:

```
const { calculate } = require('./calculator');

describe('calculator', () => {
  describe('addition', () => {
    it('adds 1 + 2 to equal 3', () => {
      expect(calculate(1, 2, '+')).toBe(3);
    });

    it('adds -5 + 7 to equal 2', () => {
      expect(calculate(-5, 7, '+')).toBe(2);
    });

    it('adds 0.1 + 0.2 to equal 0.3', () => {
      expect(calculate(0.1, 0.2, '+')).toBeCloseTo(0.3);
    });
  });

  describe('subtraction', () => {
    it('subtracts 5 - 3 to equal 2', () => {
      expect(calculate(5, 3, '-')).toBe(2);
    });
  });
});
```



Techniques

One or few shot Programming with Copilot



abelberhane

Write me unit tests for all of the operations in my calculator.js file.

Use the following example: `test('adds 1 + 2 to equal 3', () => { expect(calculate(1, 2, '+')).toBe(3); });`



GitHub Copilot

Here are some examples of unit tests for all of the operations in a `calculator.js` file:

```
const { calculate } = require('./calculator');

test('adds 1 + 2 to equal 3', () => {
  const result = calculate(1, 2, '+');
  expect(result).toBe(3);
});

test('subtracts 5 - 3 to equal 2', () => {
  const result = calculate(5, 3, '-');
  expect(result).toBe(2);
});

test('multiplies 2 * 4 to equal 8', () => {
  const result = calculate(2, 4, '*');
  expect(result).toBe(8);
});

test('divides 10 / 5 to equal 2', () => {
  const result = calculate(10, 5, '/');
  expect(result).toBe(2);
});

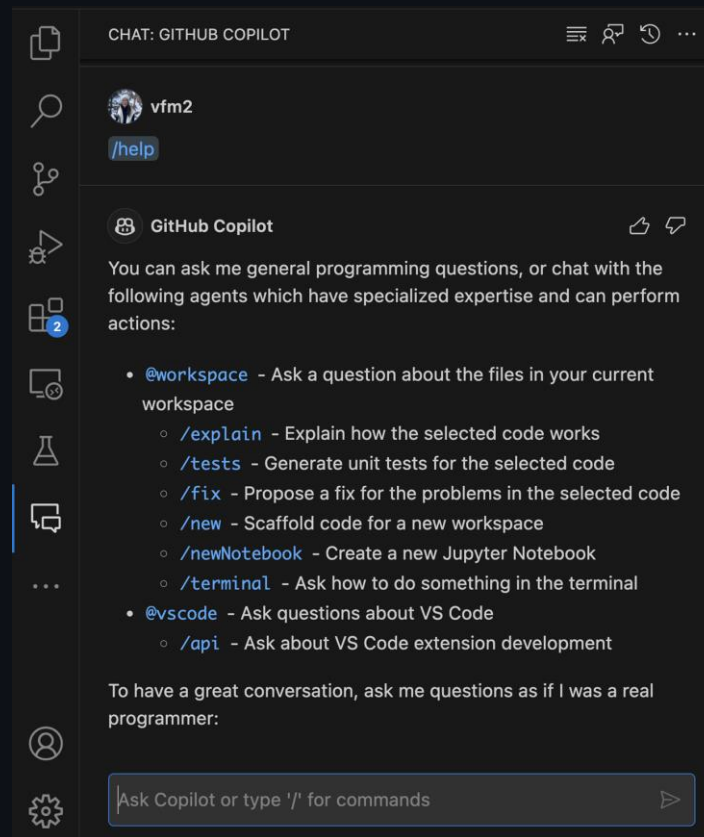
test('throws an error for an invalid operation', () => {
  expect(() => calculate(1, 2, '%')).toThrow('Invalid operation');
});

test('throws an error for division by zero', () => {
  expect(() => calculate(1, 0, '/')).toThrow('Division by zero');
});
```



GitHub Copilot Chat Commands

`/help` to find
available
commands in your
IDE



CHAT: GITHUB COPILOT

vfm2
`/help`

GitHub Copilot

You can ask me general programming questions, or chat with the following agents which have specialized expertise and can perform actions:

- `@workspace` - Ask a question about the files in your current workspace
 - `/explain` - Explain how the selected code works
 - `/tests` - Generate unit tests for the selected code
 - `/fix` - Propose a fix for the problems in the selected code
 - `/new` - Scaffold code for a new workspace
 - `/newNotebook` - Create a new Jupyter Notebook
 - `/terminal` - Ask how to do something in the terminal
- `@vscode` - Ask questions about VS Code
 - `/api` - Ask about VS Code extension development

To have a great conversation, ask me questions as if I was a real programmer:


Ask Copilot or type '/' for commands




Data Generation

Ask Copilot to
generate data

CHAT: GITHUB COPILOT

 bryantson

Produce 10 exemplars for sentiment analysis. Examples are categorized as either positive or negative. Produce 2 negative examples and 8 positive examples. Use this format for the examples:
Q: A:

 GitHub Copilot

Sure, here are 10 examples for sentiment analysis:

1. Q: The code runs smoothly without any errors. A: Positive
2. Q: The application's user interface is intuitive and user-friendly. A: Positive
3. Q: The new feature implemented in the software significantly improves its performance. A: Positive
4. Q: The documentation for the project is comprehensive and easy to understand. A: Positive
5. Q: The code is well-structured and adheres to the coding standards. A: Positive
6. Q: The application crashes frequently. A: Negative
7. Q: The software update has made the application slower. A: Negative
8. Q: The team resolved the software bugs quickly. A: Positive
9. Q: The software was deployed successfully. A: Positive
10. Q: The software was deployed successfully. A: Positive

🔗 How can I integrate sentiment analysis into my application?



Unit Testing Generation

Works best for
general **unit**
testing case
generation

CHAT: GITHUB COPILOT

bryantson

Can you help me to generate 3 unit testing methods for different possible case scenarios for all functions in selected Puzzle.java?

GitHub Copilot

Sure, I can help you with that. Here are three unit testing methods for each function in the Puzzle.java file:

```
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

public class PuzzleTest {

    @Test
    public void testSumEverythingOddNumber() {
        Puzzle puzzle = new Puzzle();
        int[] list = {1, 2, 3, 4, 5};
        int expectedSum = 9;
        int actualSum = puzzle.sum_everything_odd_number(list);
        Assertions.assertEquals(expectedSum, actualSum);
    }

    @Test
    public void testSumEverythingOddNumberWithEmptyList() {
        Puzzle puzzle = new Puzzle();
        int[] list = {};
        int expectedSum = 0;
        int actualSum = puzzle.sum_everything_odd_number(list);
        Assertions.assertEquals(expectedSum, actualSum);
    }

    @Test
    public void testConcatenateStrings() {
        Puzzle puzzle = new Puzzle();
        String first = "Hello";
        String second = "World";
        String expectedConcatenation = "HelloWorld";
        String actualConcatenation = puzzle.concatenateStringsIfI
        Assertions.assertEquals(expectedConcatenation, actualConc

    @Test
    public void testConcatenateStringsWithEmptyStrings() {
        Puzzle puzzle = new Puzzle();
        String first = "";
        String second = "";
        String expectedConcatenation = "";
        String actualConcatenation = puzzle.concatenateStringsIfI
        Assertions.assertEquals(expectedConcatenation, actualConc
```

JavaDemo > J Puzzle.java

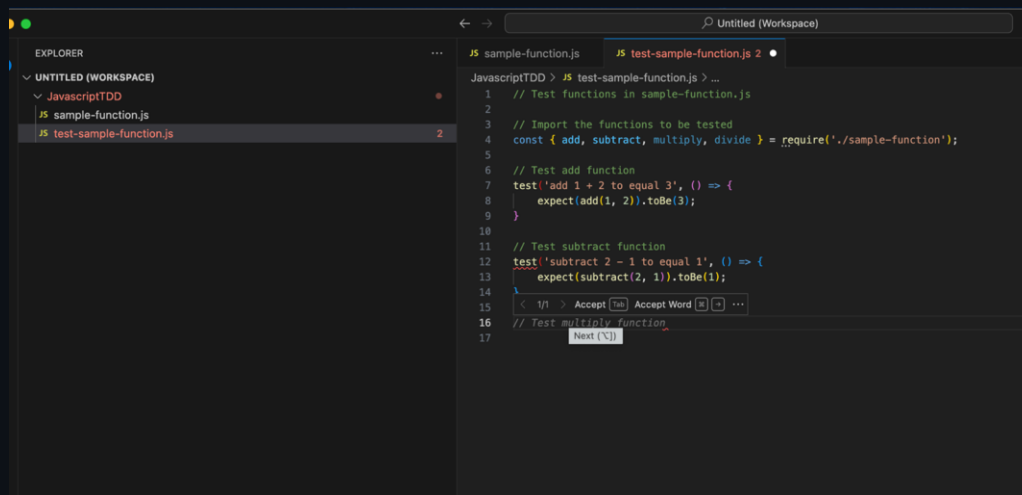
```
1 // This code sums the numbers from 1 to 100 and prints the result.
2
3 public class Puzzle {
4
5     private int sum_everything_odd_number(int[] list) { // sum_everything_od
6         int sum = 0; // sum is an integer variable with the value 0
7         for (int i = 0; i < list.length; i++) { // for loop that iterates th
8             if (list[i] % 2 == 1) { // if the number is odd
9                 sum += list[i]; // add the number to the sum
10            }
11        }
12        return sum; // return the sum
13    }
14
15     private String concatenateStrings(String first, String second) {
16         return first + second;
17     }
18
19     public static void main(String[] args) {
20         Puzzle puzzle = new Puzzle();
21         int[] list = {1, 2, 3, 4, 5};
22         System.out.println(puzzle.sum_everything_odd_number(list));
23         System.out.println(puzzle.concatenateStrings("Hello", "World"));
24     }
25 }
26
27 }
```

Can you explain how to run these unit tests in Visual Studio Code?

Ask Copilot or type "?" for commands

Test Driven Development (TDD)

Test Driven Development (TDD) practice is possible



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a workspace named 'UNTITLED (WORKSPACE)' containing a folder 'JavascriptTDD' and two files: 'sample-function.js' and 'test-sample-function.js'. The code editor is open to 'test-sample-function.js' and contains the following code:

```

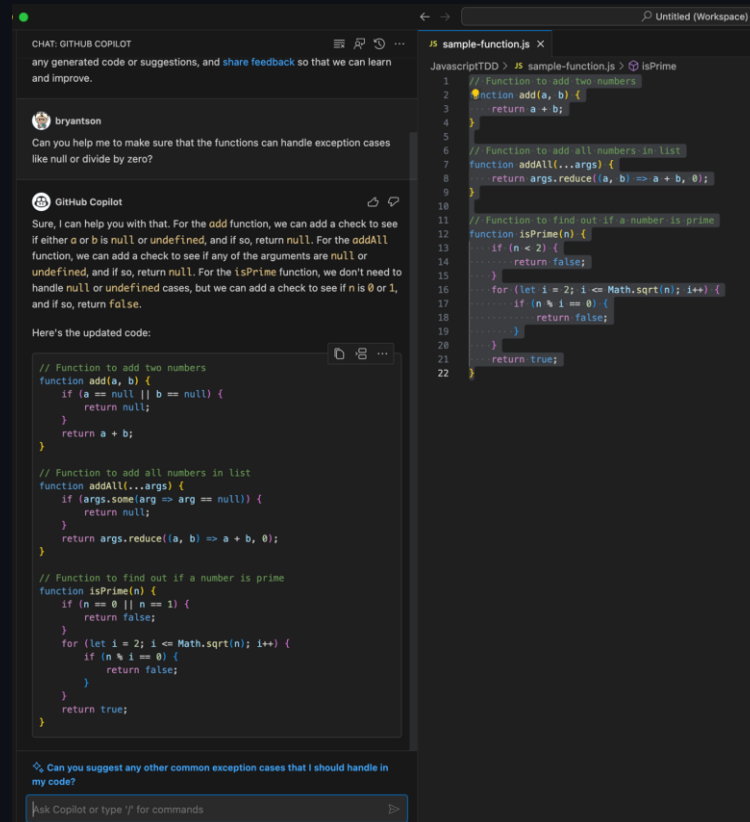
1 // Test functions in sample-function.js
2
3 // Import the functions to be tested
4 const { add, subtract, multiply, divide } = require('./sample-function');
5
6 // Test add function
7 test('add 1 + 2 to equal 3', () => {
8   expect(add(1, 2)).toBe(3);
9 })
10
11 // Test subtract function
12 test('subtract 2 - 1 to equal 1', () => {
13   expect(subtract(2, 1)).toBe(1);
14 })
15
16 // Test multiply function
17

```

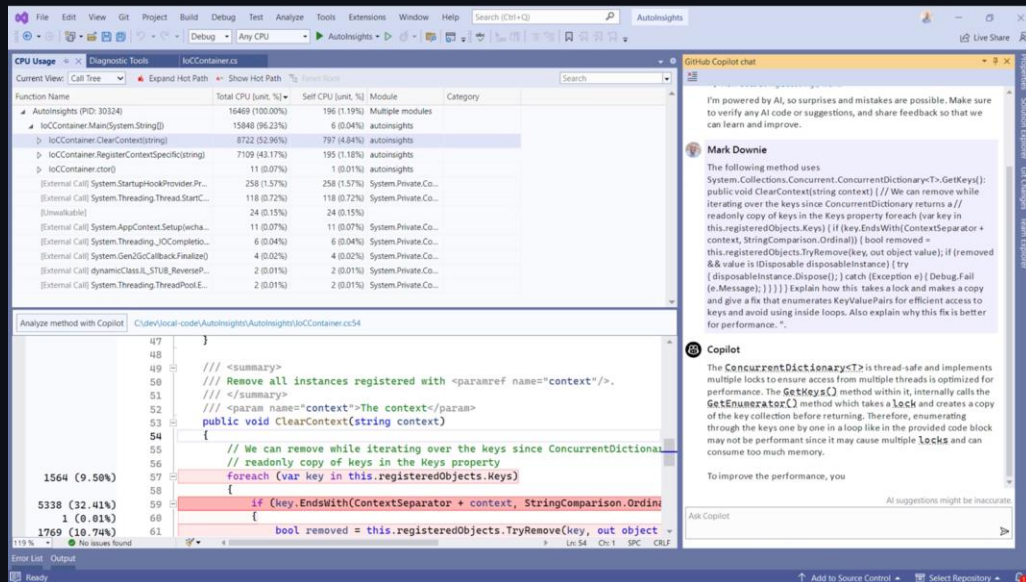
The code editor also shows a command palette with the following options: 'Accept', 'Accept Word', and 'Next'.

Code Refactoring

Refactoring is possible through GitHub Copilot Chat



GH Copilot CPU usage tool



Code Debugging

Debugging with GitHub Copilot

```
42     def collisions(self):
43         if pygame.sprite.spritecollide(self.plane, self.collision_sprites,
44                                         or self.plane.rect.top <= 0:
45             for sprite in self.collision_sprites.sprites():
46                 if sprite.sprite_type == 'obstacle':
46+                 if sprite.sprite_type == 'obstacle':

/fix Expected ":"
Accept Discard | v ↺ Changed 1 line

47             sprite.kill()
48             self.active = False
49             self.plane.kill()
50
```



GitHub Rest API

GitHub Copilot Rest API

GET /orgs/{org}/copilot/billing

cURL

JavaScript

GitHub CLI



```
# GitHub CLI api
# https://cli.github.com/manual/gh_api

gh api \
  -H "Accept: application/vnd.github+json" \
  -H "X-GitHub-API-Version: 2022-11-28" \
  /orgs/ORG/copilot/billing
```

Example response

Response schema

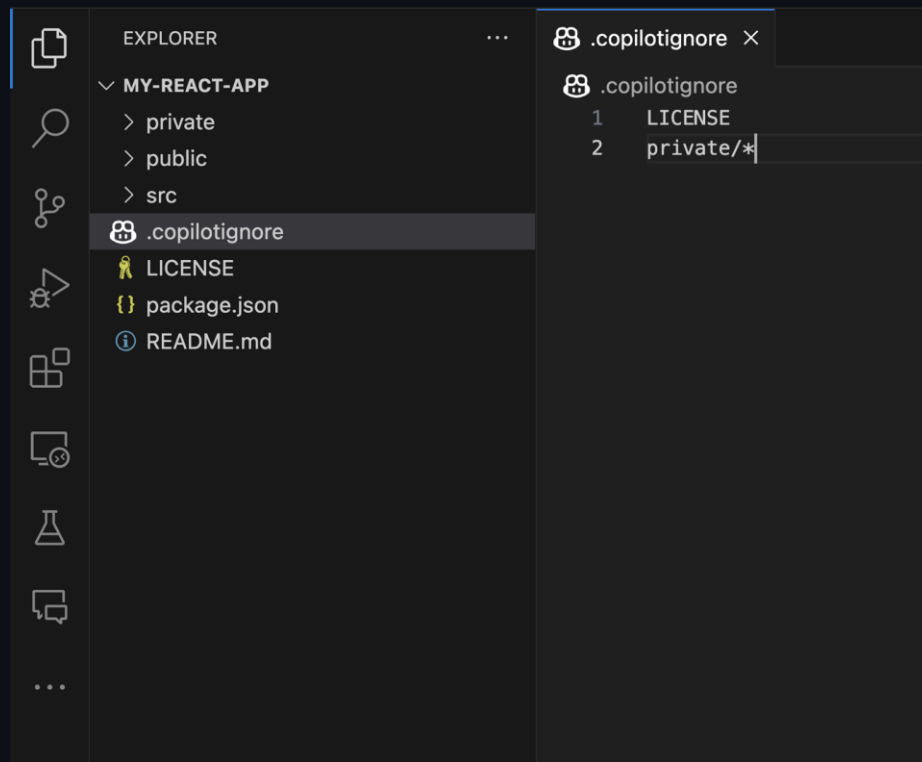
Status: 200

```
{
  "seat_breakdown": {
    "total": 12,
    "added_this_cycle": 9,
    "pending_invitation": 0,
    "pending_cancellation": 0,
    "active_this_cycle": 12,
    "inactive_this_cycle": 11
  },
  "seat_management_setting": "assign_selected",
  "public_code_suggestions": "block"
}
```



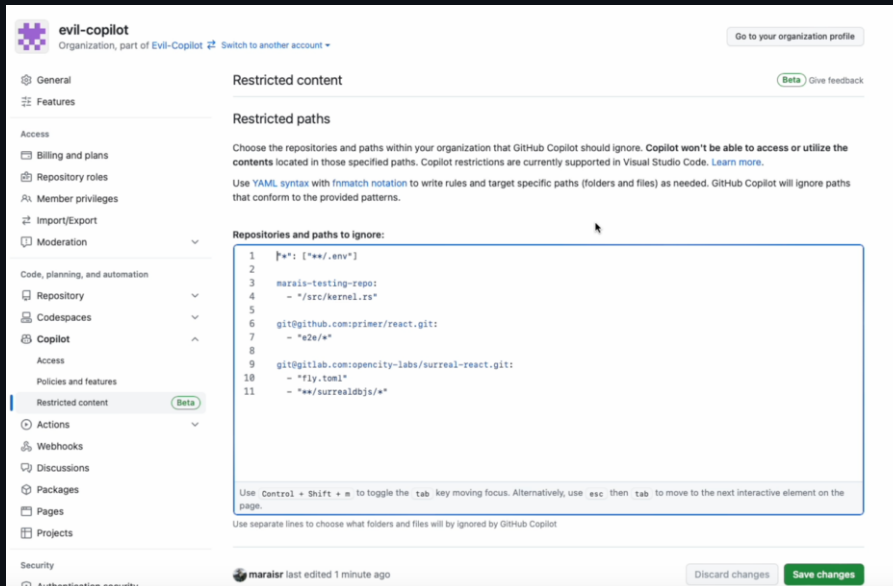
Block files from Copilot

Use `.copilotignore` to block files and folders from being used by Github Copilot



Block files from Copilot

Ignore specific
repositories and paths
in Restricted Content



The screenshot shows the 'Restricted content' settings page for the 'evil-copilot' organization. The left sidebar contains a navigation menu with options like General, Features, Access, Billing and plans, Repository roles, Member privileges, Import/Export, Moderation, Code, planning, and automation, Repository, Codespaces, Copilot, Access, Policies and features, Restricted content (highlighted with a 'Beta' badge), Actions, Webhooks, Discussions, Packages, Pages, Projects, Security, and Authentication security. The main content area is titled 'Restricted content' and includes a 'Restricted paths' section. This section explains that users can choose repositories and paths within their organization that GitHub Copilot should ignore. It provides instructions on using YAML syntax with 'fnmatch notation' to write rules. Below this, there is a text area labeled 'Repositories and paths to ignore:' containing a YAML configuration. At the bottom of the page, there are 'Discard changes' and 'Save changes' buttons, and a note indicating the settings were last edited by 'maraisr' one minute ago.

evil-copilot
Organization, part of Evil-Copilot [Switch to another account](#)

[Go to your organization profile](#)

[Give feedback](#)

Restricted content

Restricted paths

Choose the repositories and paths within your organization that GitHub Copilot should ignore. Copilot won't be able to access or utilize the contents located in those specified paths. Copilot restrictions are currently supported in Visual Studio Code. [Learn more.](#)

Use **YAML syntax** with **fnmatch notation** to write rules and target specific paths (folders and files) as needed. GitHub Copilot will ignore paths that conform to the provided patterns.

Repositories and paths to ignore:

```
1  [*]: [**/.env*]
2
3  marsis-testing-repo:
4    - */src/kernel.rs*
5
6  git@github.com:primer/react.git:
7    - */e2e/*
8
9  git@github.com:opencity-labs/surreal-react.git:
10   - */fly.toml*
11   - */**/surrealdbjs/*
```

Use **Control + Shift + m** to toggle the **tab** key moving focus. Alternatively, use **esc** then **tab** to move to the next interactive element on the page.

Use separate lines to choose what folders and files will be ignored by GitHub Copilot

[Discard changes](#) [Save changes](#)

maraisr last edited 1 minute ago



Demo: Coding practices



Best practices

Getting accuracy closer to the expectation





- Prompt Engineering

Working at a project(s) level

Leveraging Copilot to increase code quality

Getting accuracy closer to expectation

Problems

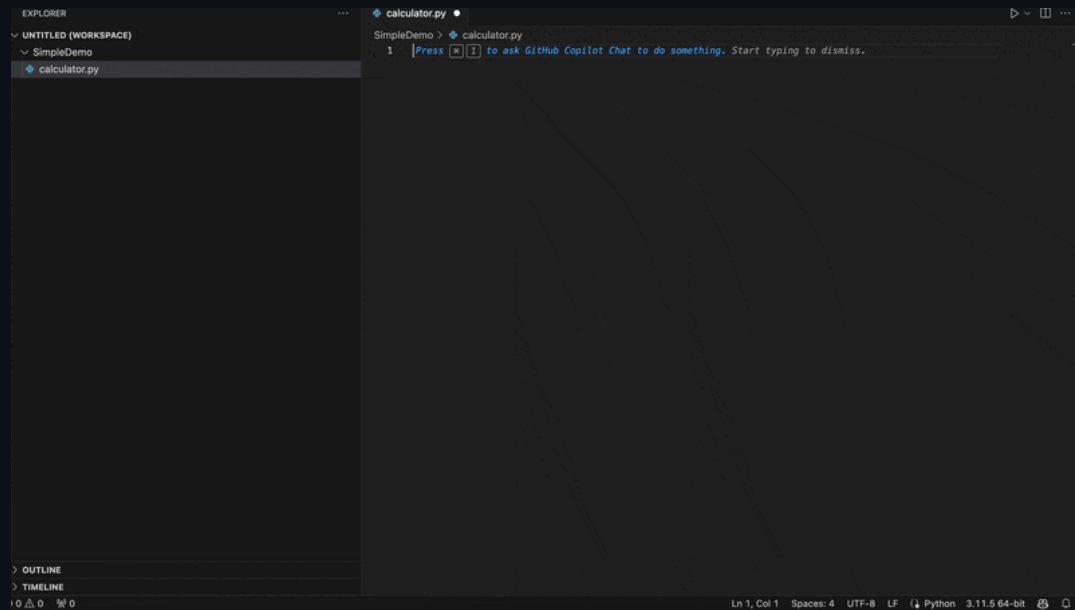
-  Copilot fails to produce answer or to keep repeating
-  Copilot generates incorrect result
-  Library/module version discrepancies issue
-  Copilot suggests non-optimal solution

Problems #1:

Copilot fails to produce answers or will keep repeating

Some problems

- Fails to produce answer
- Hallucination - Keeps repeating

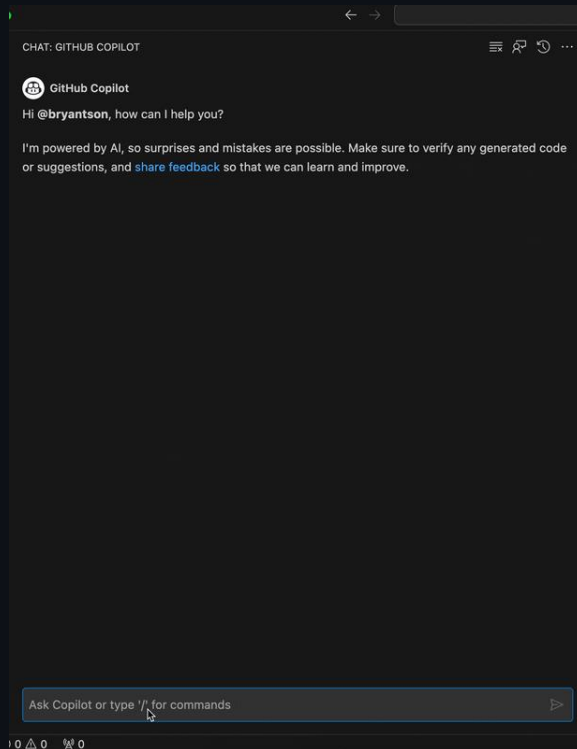


Problems #2:

Copilot generates incorrect result

Reasons why...

- Not enough context
- Old trained data
- Generative in nature

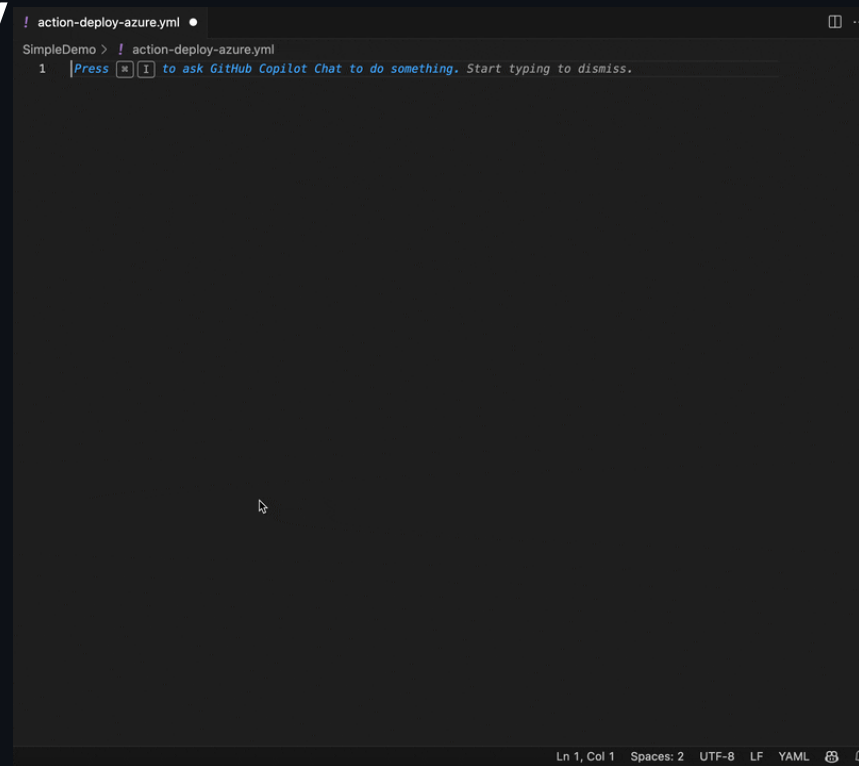


Problems #3:

Library/module version discrepancy

Old trained data

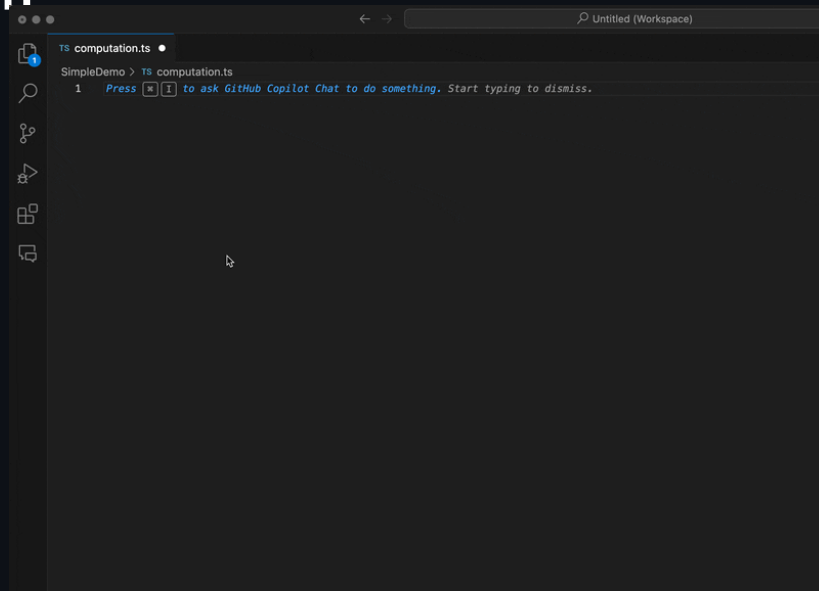
- While packages go through frequent updates, Copilot does not use latest data



Problems #4:

Copilot suggests non-optimal solution

Although solution works...
Suggested solution is not
optimal because Quick
Sort can be implemented
in $O(1)$ complexity,
meaning no space
required

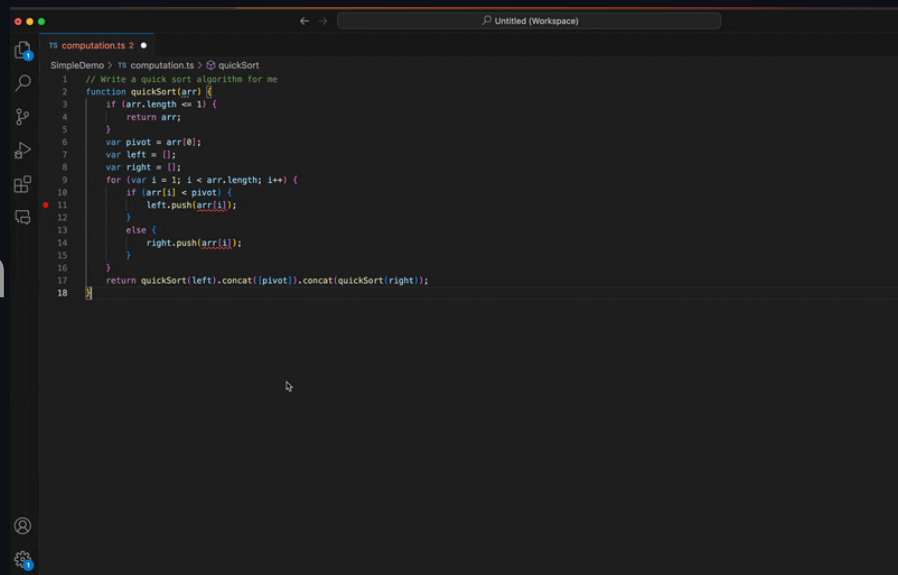


How to improve Problem #4

Copilot suggests non-optimal solution

How to improve

- Ask in Copilot Chat
- Use in-line suggestion from Copilot



The screenshot shows a code editor window titled "TS computation.ts 2" with a file explorer on the left. The code is a TypeScript function for quicksort. A Copilot suggestion is shown as a light blue box around the line `left.push(arr[i]);` on line 11. The suggestion is a red squiggly line, indicating it is not optimal. The code is as follows:

```
1 // Write a quick sort algorithm for me
2 function quickSort(arr) {
3   if (arr.length <= 1) {
4     return arr;
5   }
6   var pivot = arr[0];
7   var left = [];
8   var right = [];
9   for (var i = 1; i < arr.length; i++) {
10    if (arr[i] < pivot) {
11      left.push(arr[i]);
12    }
13    else {
14      right.push(arr[i]);
15    }
16  }
17  return quickSort(left).concat([pivot]).concat(quickSort(right));
18 }
```

Secure coding

Copilot and secure coding

Copilot + GitHub Advanced Security

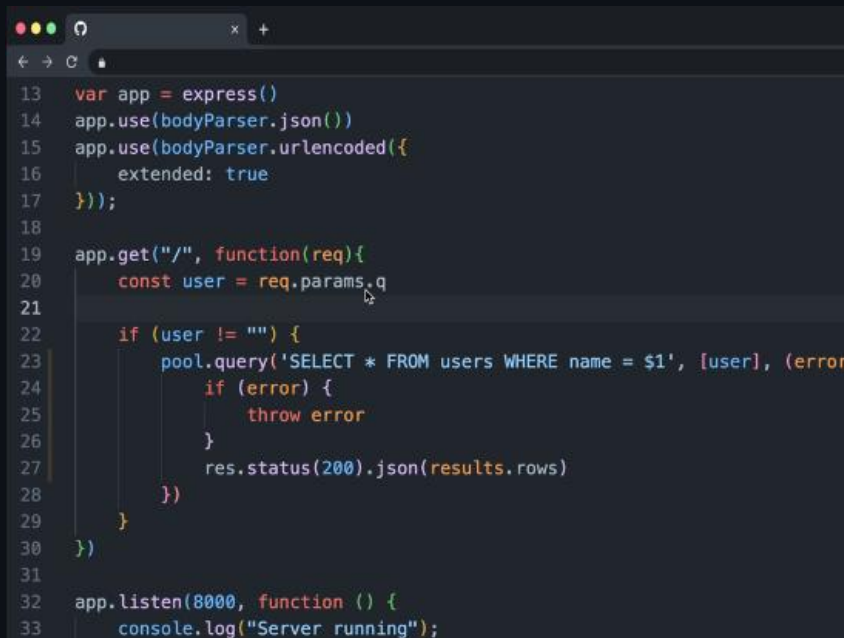
Vulnerability detection

Vulnerability remediation

Copilot Trust Center

Copilot and secure coding

- AI-based vulnerability system that helps prevent insecure coding patterns (e.g. SQL script injection)
- Vulnerability filters are applied only to the Copilot-generated suggestion itself
 - ⊗ It cannot detect downstream vulnerabilities introduced by the code e.g. on deployment infrastructure
 - ⓘ We recommend taking the **same precautions** you take with code written by engineers (linting, code scanning, etc.)
- Copilot Chat can be used to query code for known vulnerabilities



```

13 var app = express()
14 app.use(bodyParser.json())
15 app.use(bodyParser.urlencoded({
16   extended: true
17 }));
18
19 app.get("/", function(req){
20   const user = req.params.q
21
22   if (user != "") {
23     pool.query('SELECT * FROM users WHERE name = $1', [user], (error
24       if (error) {
25         throw error
26       }
27       res.status(200).json(results.rows)
28     })
29   }
30 })
31
32 app.listen(8000, function () {
33   console.log("Server running");

```



Copilot + GHAS

- Copilot is not a replacement of GHAS features
- Copilot can be used in tandem with GHAS features to detect and remediate vulnerabilities earlier during the SDLC
 - GHAS Code scanning results
 - GHAS Secret scanning



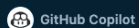
Vulnerability detection

- Prompt for insecure coding patterns
- Use GHAS Code Scanning results
- Write custom CodeQL queries
- Increase your knowledge of secure coding patterns
- Create custom Secret Scanning patterns



stebje

It looks like this code is insecure. Help me understand what the issue is and how to resolve it.



Yes, this code is vulnerable to SQL injection attacks. The issue is that the `search` parameter is directly concatenated into the SQL query string without any sanitization or validation. This means that an attacker could potentially inject malicious SQL code into the query, which could lead to data theft or destruction.

To fix this issue, you should use parameterized queries instead of concatenating strings. Parameterized queries use placeholders for the input values, which are then bound to the query separately. This ensures that the input values are properly sanitized and validated before being used in the query.

Here's an example of how to use parameterized queries with the `pg` library:

```
const search = req.params.q;

if (search !== '') {
  const query = "SELECT * FROM users WHERE name = $1";
  const values = [search];
  pool.query(query, values, (err, res) => {
    console.log(err, res);
    pool.end();
  });
}
```

In this example, the `$1` placeholder is used for the `search` parameter, and the actual value is passed in as an array of values to the `pool.query()` function. This ensures that the `search` parameter is properly sanitized and validated before being used in the query.

```
4
5 const pool = new Pool( config: {
6   user: 'postgres',
7   host: 'localhost',
8   database: 'postgres',
9   password: 'process.env.POSTGRES_PW',
10  port: 5432,
11 });
12
13 var app = express();
14 app.use(bodyParser.json());
15 app.use(bodyParser.urlencoded({ extended: true }));
16
17 app.get('/', function (req, res) {
18   const search = req.params.q;
19
20   if (search !== '') {
21     var query = "SELECT * FROM users WHERE name = \"\" + search + \"\"";
22     pool.query( queryTextOrConfig: query, callback: (err, res) => {
23       console.log( message: err, optionalParams[0]: res);
24       pool.end();
25     });
26   }
27 });
28
29 app.listen( port: 8000, callback: function () {
30   console.log( message: 'Example app listening on port 8000!');
31 });
```



Vulnerability remediation

- Copilot helps prevent suggestions that contain insecure coding patterns in real-time
- Automatically fix insecure code based on Copilot suggestions
- Validate and improve existing CodeQL queries



Copilot Security Exploit?

Prompt Injection

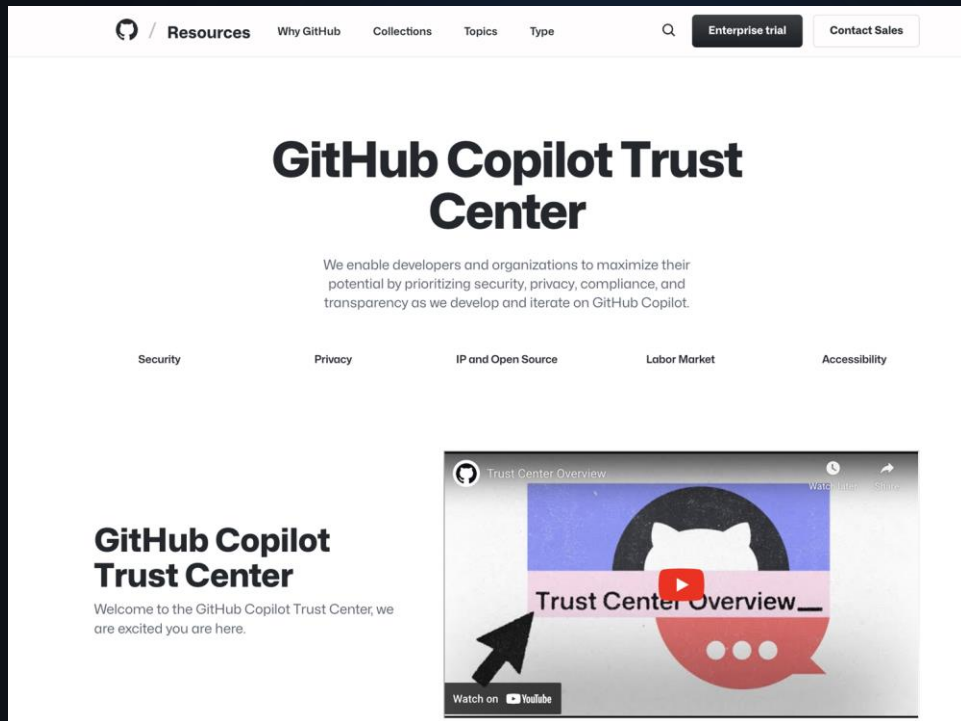
Carefully crafted prompts to make the model to ignore its original instruction or perform unintended actions



Security & Trust

Copilot Trust Center

- Security
- Privacy
- Data flow
- Copyright
- Labor market
- Accessibility
- Contracting





Workshop



Wrap Up



Thank you

