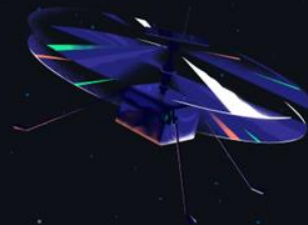




GitHub Copilot Developers Training



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](#)



Choose Workshop Track

Copilot Workshop



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

Primary IDE: VS Code
Primary OS: Independent

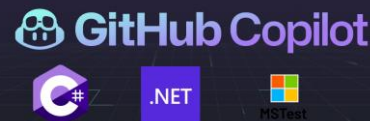
Copilot Workshop



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
Primary OS: Windows



AGENDA

GitHub Copilot - Introduction

Coding

Best practices & prompt engineering

Workshop (1 - 2 hours long)

Secure coding

Wrap-up, Q&A



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



Covered in Copilot Fundamentals

We will not talk about...

- ✗ Statistics around Copilot usage and satisfaction
- ✗ Successful customer case study
- ✗ Enterprise & Organization administrator interface





GitHub Copilot Fundamentals Recap



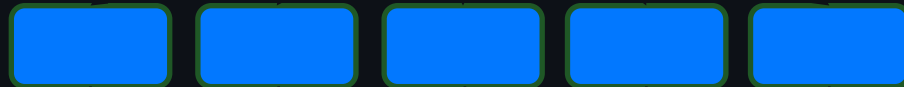
The software process can be broken down into two steps:

- 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.

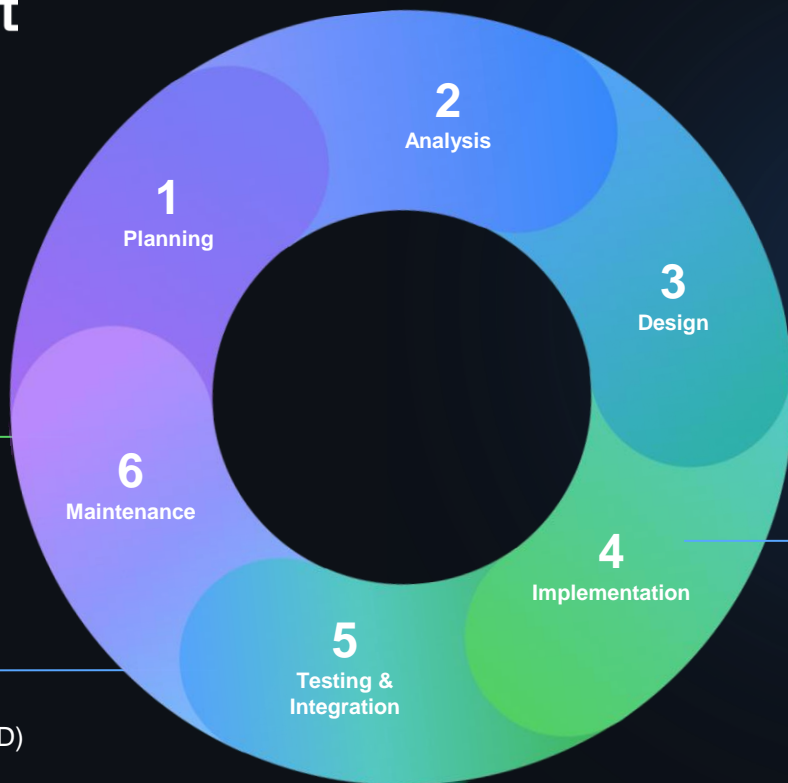


IMPLEMENTATION





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




GitHub Copilot

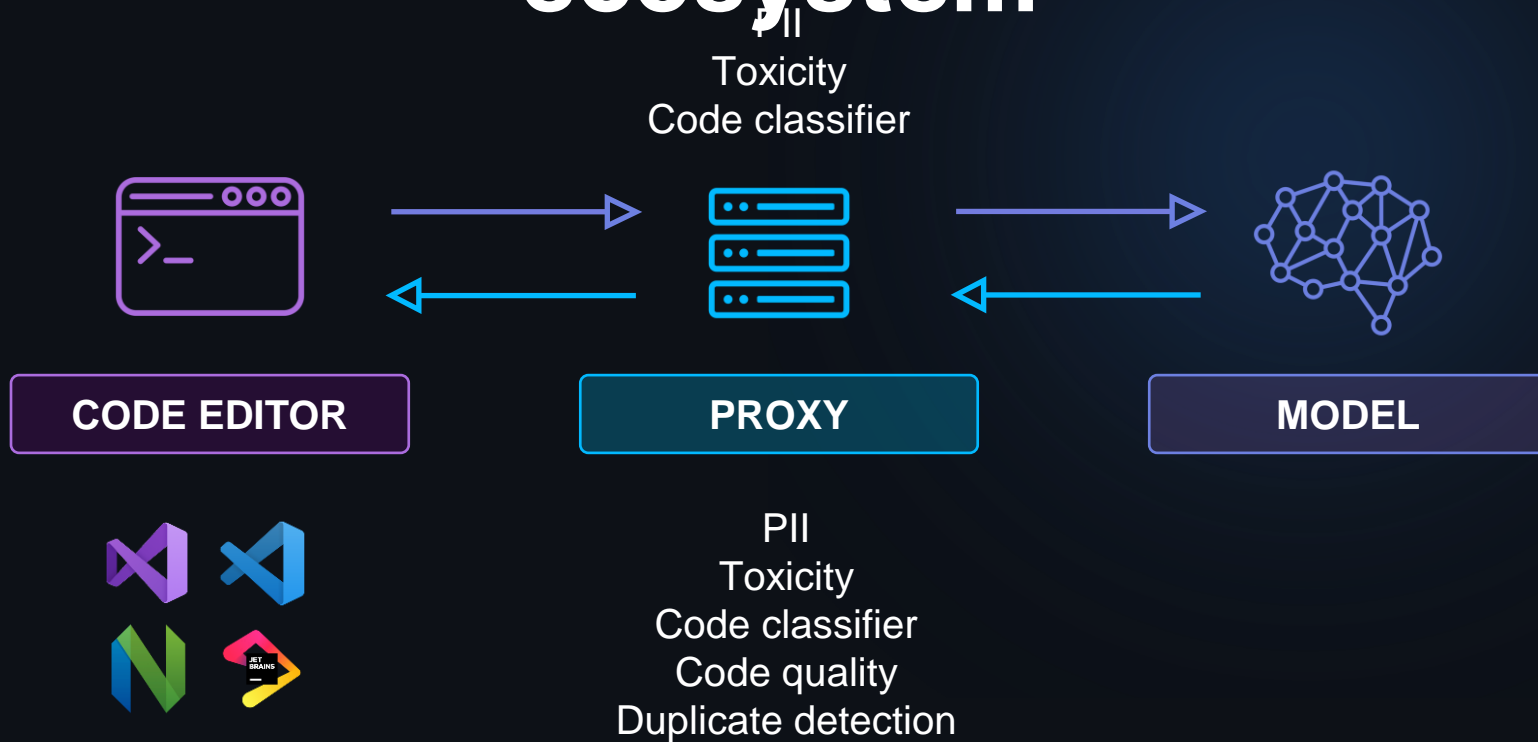
- An intelligent pair programmer
- Draws context from comments & code in open tabs to suggest individual lines and whole functions
- Powered by OpenAI Codex
 - Copilot uses a transformative model
 - Trained on large datasets to ensure accuracy
- Available as extensions to popular IDEs
- Programming Languages 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 }
```

 Copilot

Data flow through the Copilot ecosystem

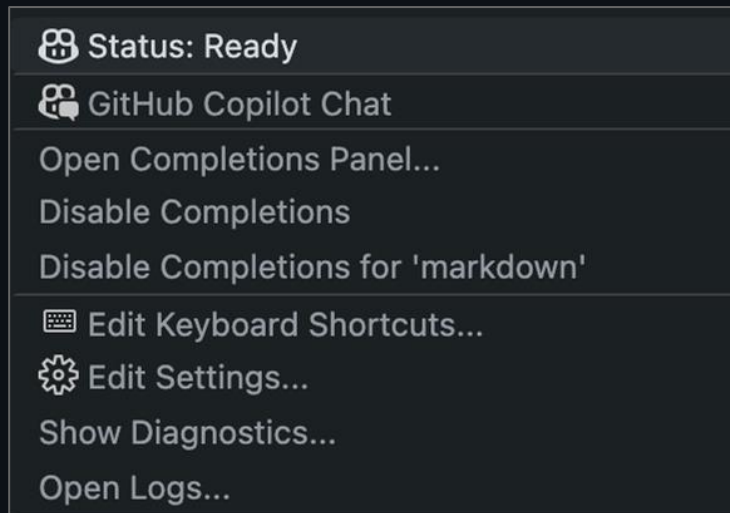


Coding



Advanced Settings

Configuration Options



[Configuring GitHub Copilot in your environment](#)



Copilot vs Copilot Chat

Copilot

Direct Code Writing

Seamless IDE Integration

Solo Development

Copilot Chat

In-Depth Assistance

Learning & Teaching

Collaborative Scenarios



Using Copilot

GitHub Copilot

- ☐ Code Driven Development
- ☐ Multilingual
- ☐ Offering Alternative Results
- ☐ Create Unit Tests
- ☐ Documentation



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



Providing Context

To help Copilot generate accurate suggestions:

- Add a top-level comment block describing the purpose of the file
- Front load as many header files with `#include`
- Use descriptive function / method / type names
- Use sample code as a starting point
- Have related content open in other tabs



Helpful Patterns

- * Use descriptive variable names to make your intentions clear.

```
totalSampleCount = 1000
```

- * Define method signatures with unambiguous parameter names and types.

```
double calculateAverageSampleSize(unsigned long samples[], size_t size)
```

- * Show examples of how to use the code.

```
int samples[] = {1, 2, 3, 4, 5};  
double average = calculateAverageSampleSize(samples, 5);  
printf("Average: %f\n", average); // Average: 3.0
```

- * Maintain consistent naming conventions for variables and functions.

i.e. using camelCase for variable names consistency



Helpful Patterns

- * Use comments to explain the purpose of the code.

```
This function 'palindrome' takes an array of strings  
as input and returns true if a palindrome is found.
```

- * Specify error handling scenarios.

```
Exit where the input is NULL and throw and  
exception if the input out of range [min, max] inclusive.
```

- * Describe control flow structures.

```
Write a while loop to print the first 'n' values in the fibonnaci sequence.
```

- * Test your code.

```
Write a test suite to verify the correctness of the code.
```



Using Copilot Chat

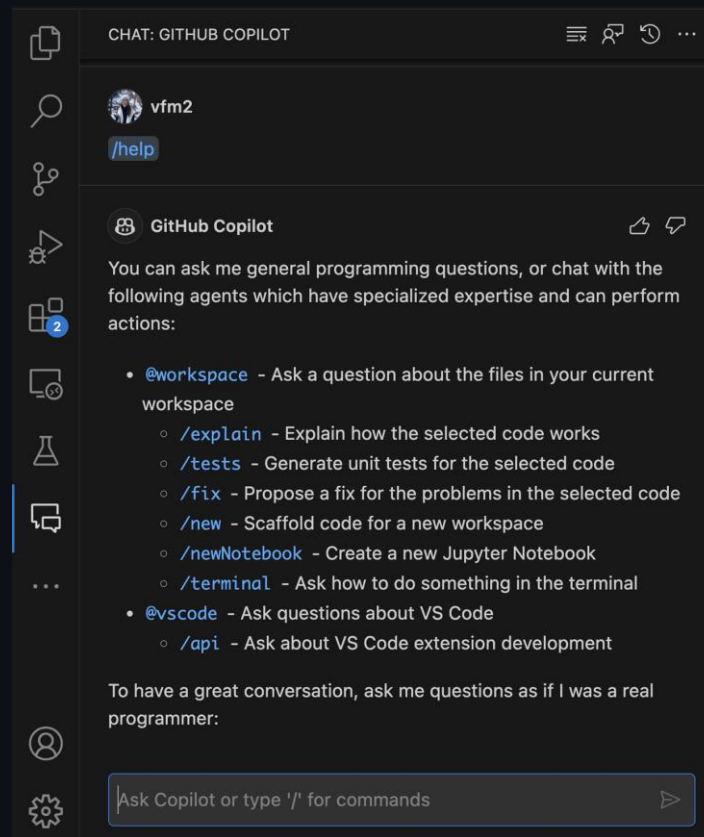
GitHub Copilot Chat

- ☐ Refactor Code
- ☐ Generate Tests
- ☐ Debug
- ☐ Create Workspace
- ☐ Documentation



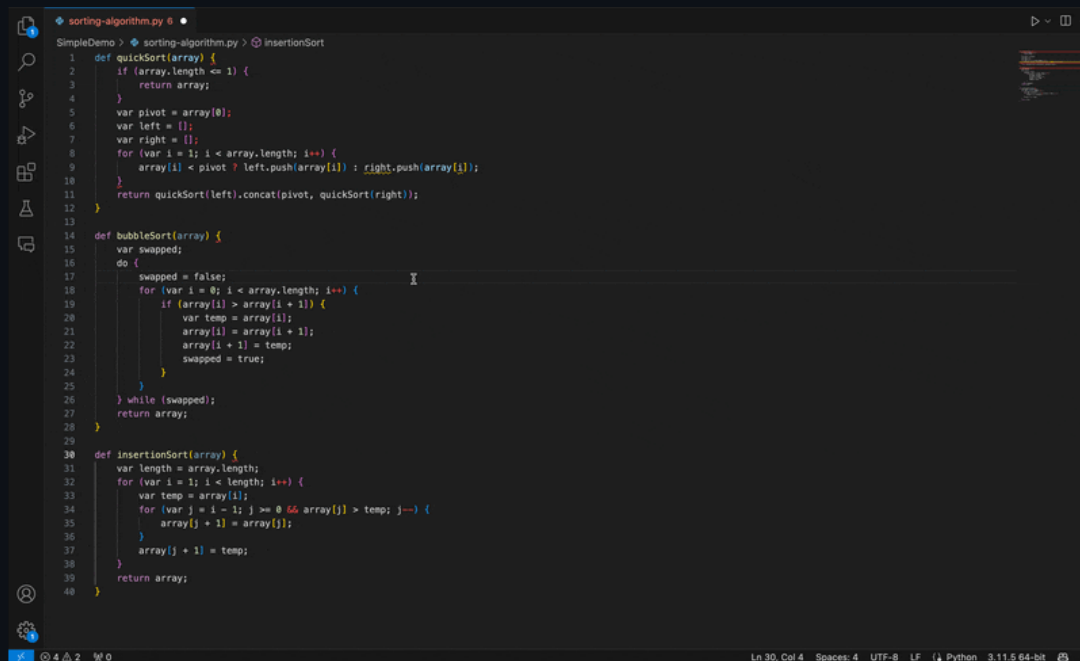
Copilot Chat: Slash Commands

`/help` to find
available
commands in your
IDE



In-file Copilot Chat

Copilot offers
in-file Copilot
feature to
selectively
improve



The screenshot shows a code editor with a file named 'sorting-algorithm.py'. The code contains three sorting functions: quickSort, bubbleSort, and insertionSort. The quickSort function is on lines 1-13, bubbleSort on lines 14-28, and insertionSort on lines 30-39. The Copilot Chat interface is visible on the right side of the editor, showing a chat window with a message from the user and a response from Copilot. The chat window is titled 'SimpleDemo > sorting-algorithm.py > InsertionSort'. The response from Copilot is a code snippet that improves the bubbleSort function by adding a 'swapped' variable to track whether any elements were swapped in the current pass, which can help optimize the algorithm by stopping early if the array is already sorted.

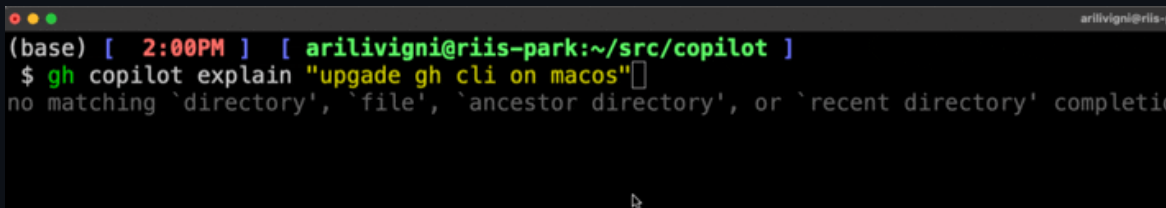
```
1 def quickSort(array) {  
2   if (array.length <= 1) {  
3     return array;  
4   }  
5   var pivot = array[0];  
6   var left = [];  
7   var right = [];  
8   for (var i = 1; i < array.length; i++) {  
9     array[i] < pivot ? left.push(array[i]) : right.push(array[i]);  
10  }  
11  return quickSort(left).concat(pivot, quickSort(right));  
12 }  
13  
14 def bubbleSort(array) {  
15   var swapped;  
16   do {  
17     swapped = false;  
18     for (var i = 0; i < array.length; i++) {  
19       if (array[i] > array[i + 1]) {  
20         var temp = array[i];  
21         array[i] = array[i + 1];  
22         array[i + 1] = temp;  
23         swapped = true;  
24       }  
25     }  
26   } while (swapped);  
27   return array;  
28 }  
29  
30 def insertionSort(array) {  
31   var length = array.length;  
32   for (var i = 1; i < length; i++) {  
33     var temp = array[i];  
34     for (var j = i - 1; j >= 0 && array[j] > temp; j--) {  
35       array[j + 1] = array[j];  
36     }  
37     array[j + 1] = temp;  
38   }  
39   return array;  
40 }
```



Run in Terminal

Ask in GitHub
Copilot CLI

GitHub Copilot Chat (default)



```
(base) [ 2:00PM ] [ arilivigni@riis-park:~/src/copilot ]  
$ gh copilot explain "upgrade gh cli on macos"  
no matching `directory`, `file`, `ancestor directory`, or `recent directory` completi
```



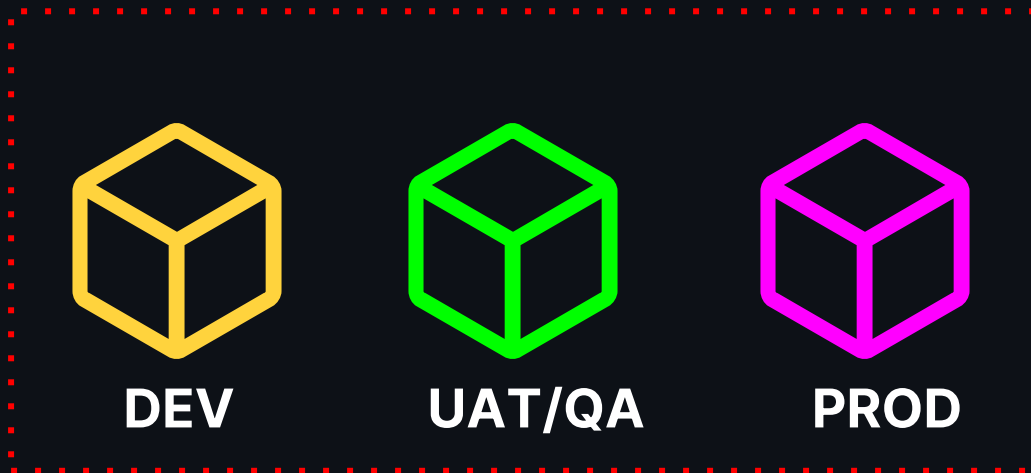
What about my production layer?

Local



Current Copilot

Cloud/Server



Copilot Enterprise







Best practices



Getting accuracy closer to expectation

Problems

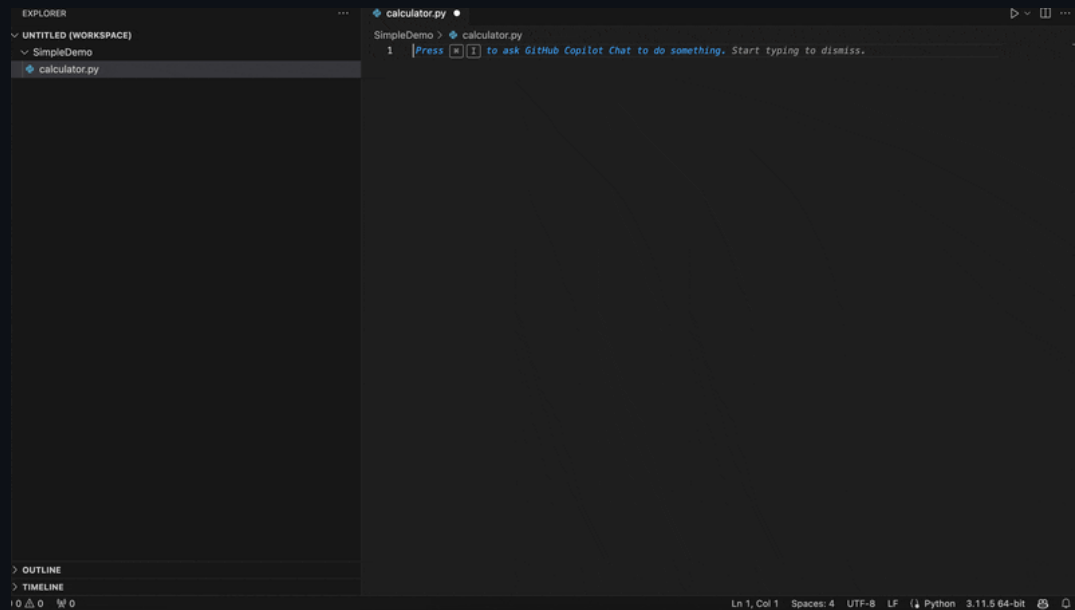
-  Copilot fails to produce answer or to keep repeating
-  Copilot generates incorrect result
-  Library/module version discrepancies issue
-  Copilot suggests non-optimal solution

Problems #1:

Copilot fails to produce answers or will keep repeating

Some problems

- Fails to produce answer
- Hallucinations
- Keeps repeating

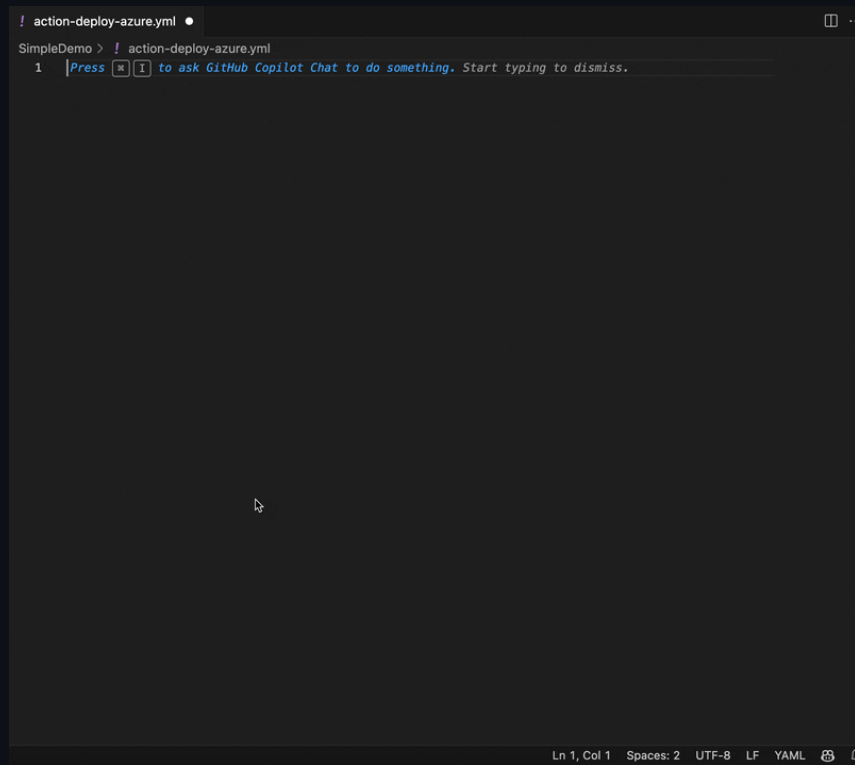


Problems #2:

Library/module version discrepancy

Old trained data

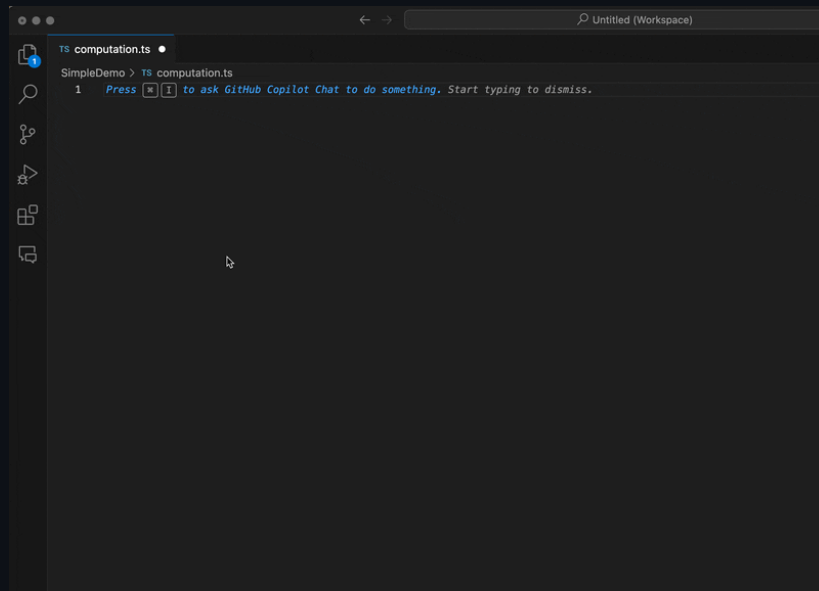
- While packages go through frequent updates, Copilot does not use latest data



Problems #3:

Copilot suggests non-optimal solution

Although solution works...
It is not optimal space
complexity



Problems #4:

Copilot Token Limit

Strategies to overcome the Token Limit

- Break Down Tasks
- Be Concise
- Iterative Development
- Post-Processing
- Stay Updated
- Feedback Loop

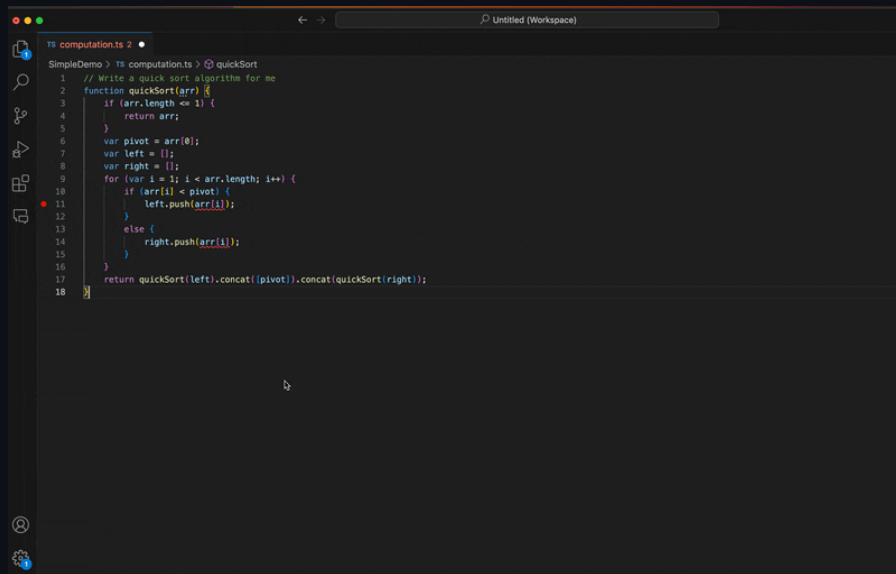


Solution: Prompt Engineering

Ways to improve Copilot Results

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'. The code is a TypeScript implementation of a quicksort algorithm. A Copilot suggestion is visible on line 11, where the text 'left.push(arr[i]);' is highlighted in blue, indicating it was suggested by the AI. 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 }
```

Copilot Prompt Engineering

What

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.

Why

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

How

- Neighboring Tabs
- Zero-Shot Prompting
- One-Shot Prompting
- Few-Shot Prompting
- Let's Think Step by Step



Prompting Best Practices

Improving results



These are best practices, now let's see how those can be used in different ways



Provide references

Improve relevance of the response by providing an example and context



Write clear instructions

Refine your prompt, provide context, write clearly, and give Copilot ample input for better results



Split up big tasks

Breaking down complex tasks minimizes errors and utilizes previous outcomes for efficiency



Allow time to think

Requesting Copilot's thought process will enhance Copilot accuracy, but it may prolong wait times.

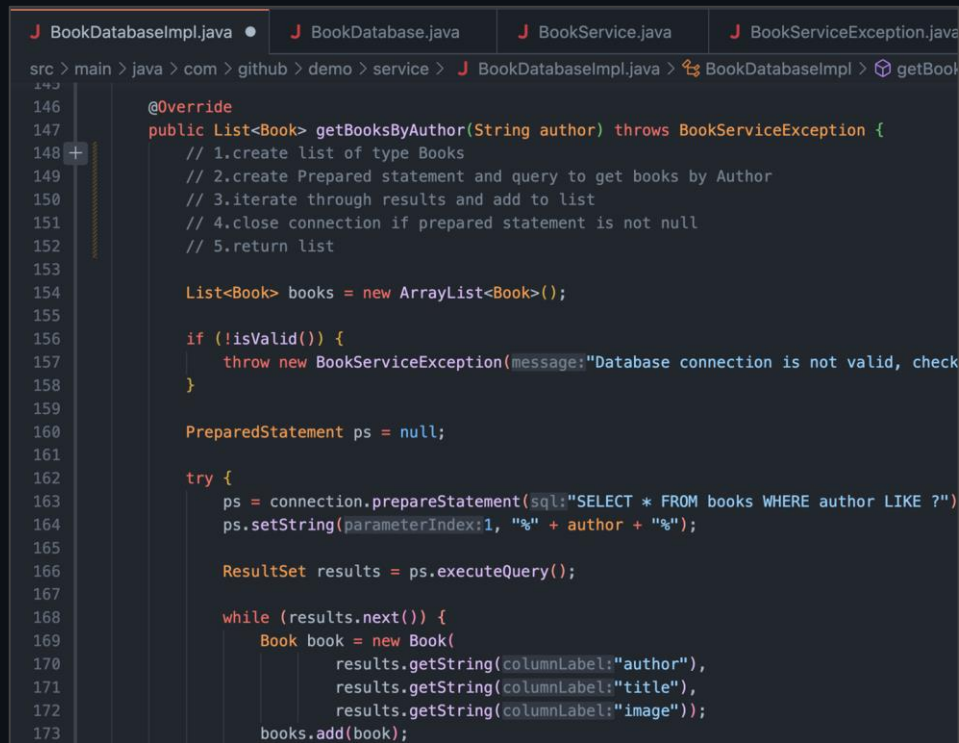


Test changes systematically

Measure performance, watch for prompt changes' side effects, and use test suites for implementation

Neighboring Tabs

Keep relevant files open, things related to your current code



```

J BookDatabaseImpl.java • J BookDatabase.java J BookService.java J BookServiceException.java
src > main > java > com > github > demo > service > J BookDatabaseImpl.java > BookDatabaseImpl > getBook
146 @Override
147 public List<Book> getBooksByAuthor(String author) throws BookServiceException {
148 // 1.create list of type Books
149 // 2.create Prepared statement and query to get books by Author
150 // 3.iterate through results and add to list
151 // 4.close connection if prepared statement is not null
152 // 5.return list
153
154 List<Book> books = new ArrayList<Book>();
155
156 if (!isValid()) {
157     throw new BookServiceException(message:"Database connection is not valid, check
158 }
159
160 PreparedStatement ps = null;
161
162 try {
163     ps = connection.prepareStatement(sql:"SELECT * FROM books WHERE author LIKE ?")
164     ps.setString(parameterIndex:1, "%" + author + "%");
165
166     ResultSet results = ps.executeQuery();
167
168     while (results.next()) {
169         Book book = new Book(
170             results.getString(columnLabel:"author"),
171             results.getString(columnLabel:"title"),
172             results.getString(columnLabel:"image"));
173         books.add(book);

```



If at first you
don't succeed,
iterate!

Zero Shot Prompt

```
1 //Write a JavaScript function that finds
2 //the maximum value in an array.
3 function max(array) {
4     return Math.max.apply(null, array);
5 }
```

One-Shot Prompt

```
1 ✓ //Create a JavaScript function, `findMax`, that takes an array of
2 //and returns the string with the maximum length.
3 //Ensure that the function works correctly for arrays that are em
4 //as well as arrays that contain both strings and numbers.
5 ✓ function findMax(array) {
6     var max = array[0].length;
7     array.map(v => max = Math.max(max, v.length));
8     result = array.filter(v => v.length == max);
9     return result;
10 }
```



Provide a high level description to clarify expectations

```
calculations.py > ...
1  """
2  Create a Python function, `sum_array_values`, that finds
3  and returns the sum of integers in an array.
4
5  1.Verify if the array is empty and exit if it is.
6  2.Throw error if the array contains non-integer values.
7  3.Return the sum of the array values if the array is valid.
8
9  Please consider the following example data:
10 Example 1: [5, 10, 15, 20, 8]
11 Example 2: [2, 4, 6, 1, 9, 12]
12 The generated code should work correctly for arrays like above
13 """
14 #define sum_array_values function
15 def sum_array_values(array):
16     # 1.Verify if the array is empty and exit if it is.
17     if len(array) == 0:
18         return 0
19     # 2.Throw error if the array contains non-integer values.
20     for i in array:
21         if type(i) != int:
22             raise TypeError("The array contains non-integer values.")
23     # 3.Return the sum of the array values if the array is valid.
24     return sum(array)
```

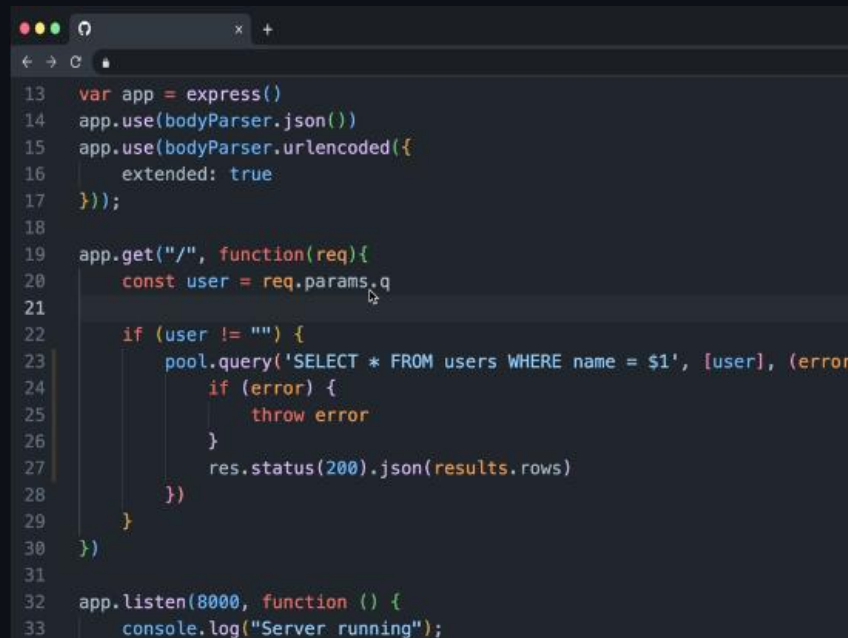


Secure coding



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



Detection & Remediation

- 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.



GitHub Copilot

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 sqquery = "SELECT * FROM users WHERE name = $1";
  const values = [search];
  pool.query(sqquery, 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.

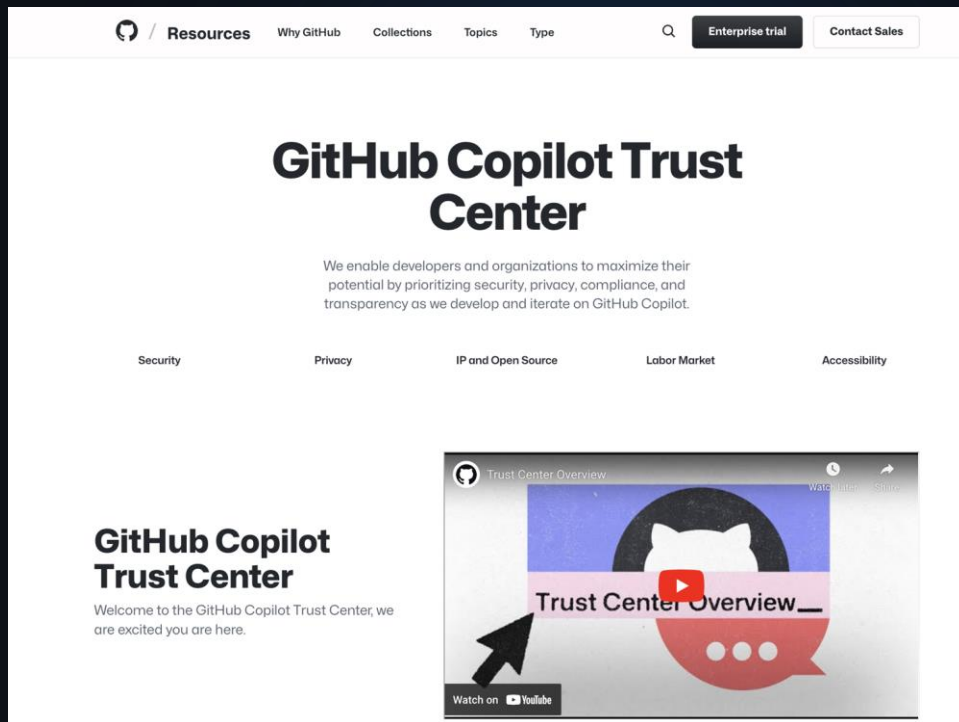
```
4
5 const pool = new Pool( config: {
6   user: 'postgres',
7   host: 'localhost',
8   database: 'postgres',
9   password: 'process.env.POSTGRES_PW',
10  port: 5432,
11 });
12
13 var app = express();
14 app.use(bodyParser.json());
15 app.use(bodyParser.urlencoded({ extended: true }));
16
17 app.get('/', function (req, res) {
18   const search = req.params.q;
19
20   if (search !== '') {
21     var sqquery = "SELECT * FROM users WHERE name = \"\" + search + \"\"";
22     pool.query( queryTextOrConfig: sqquery, 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 });
```



Security & Trust

Copilot Trust Center

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



Wrap Up



Thank you

