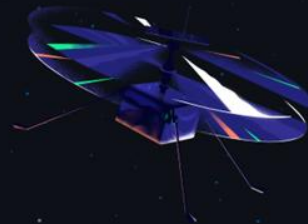# GitHub Copilot Training for Developers

Andrew Scoppa

AGENDA

GitHub Copilot - Introduction

Best practices & prompt engineering

In-class coding demos using copilot and copilot chat

Secure coding

Wrap-up, Q&A

GitHub Copilot Introduction

Let's start with a high-level overview of GitHub Copilot

- GitHub Copilot is there to enhance daily work
    - It's an intelligent "pair-programmer"
    - Like a smart assistant or mentor by your side

- Powered by OpenAI
    - Copilot uses a transformative model
    - Think of something like Google Translate

- Trained on large datasets to ensure accuracy
    - Languages with large representation produce best results

- Copilot generates new code in a probabilistic way
    - Unlikely to produce the same code as a snippet that occurred in training.

GitHub Copilot Architecture "Big Picture"
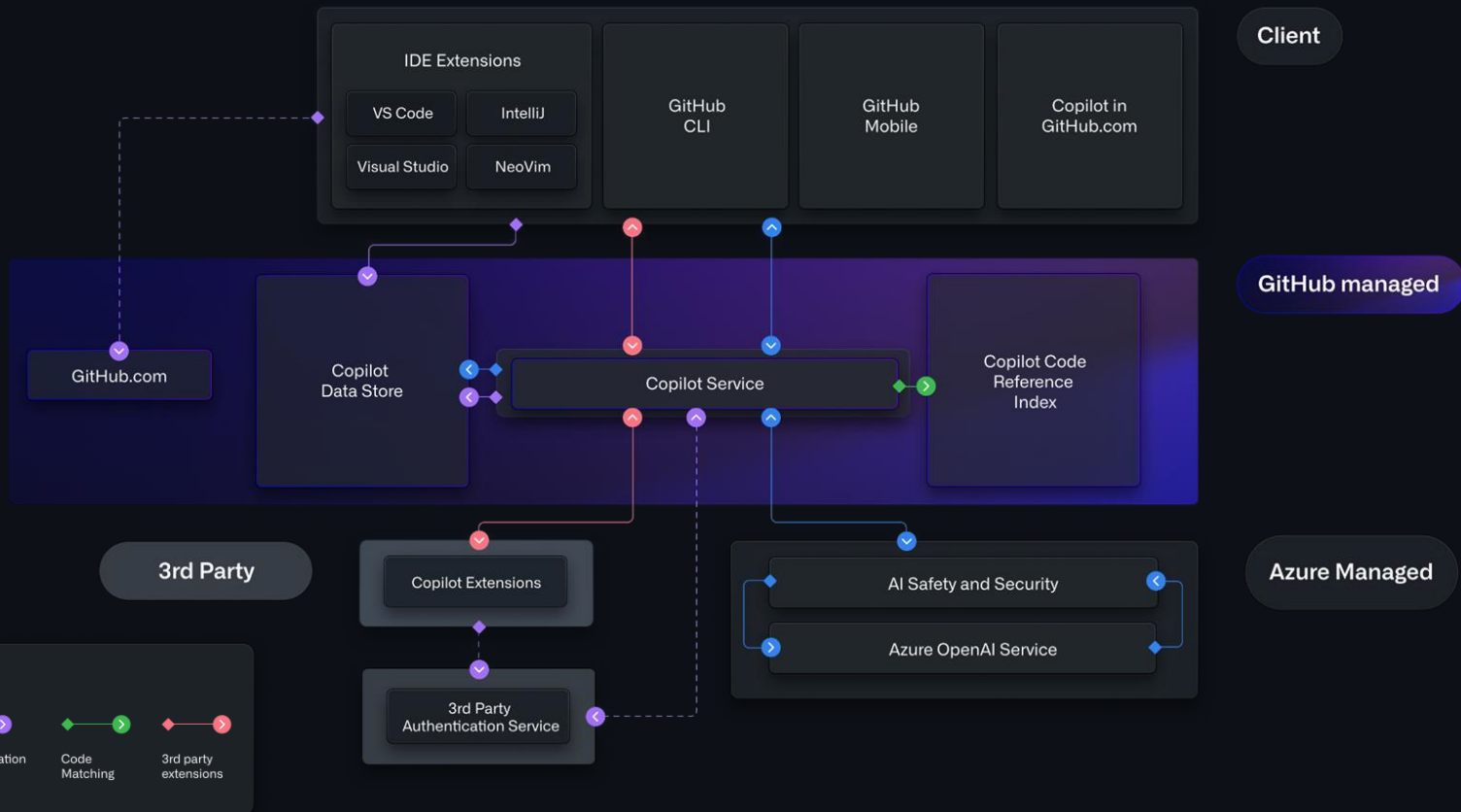
Copilot vs Copilot Chat

Copilot

Direct Code Writing

Your "coding assistant"
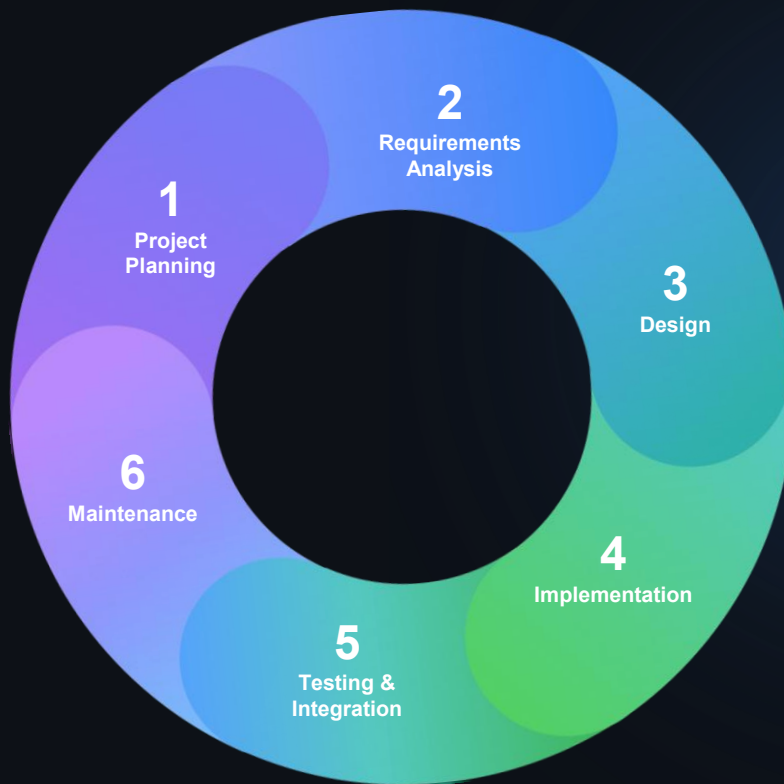
Solo Development

Copilot Chat

In-Depth Interactive Assistance

Your "research assistant"

Collaborative Scenarios

GitHub Copilot + Chat

Helps developers stay in the flow throughout the entire SDLC

Using Different LLM Models with Copilot

- Several models to choose from including

  GPT 5 - Next-generation successor to GPT-4o, geared toward deeper multi-step reasoning, tool orchestration, and extended trusted context windows.

  Claude 3.5 Sonnet - Balanced model delivering strong reasoning, long-context handling, and fast iterative responses at moderate cost.

  o3-mini (OpenAI) - Lightweight reasoning-optimized model variant focused on fast iterative problem solving (code, logic puzzles, structured planning) with lower latency and cost versus larger flagship models.

- Be aware of costs

  Costs mainly come from tokens (input/output/context), model tier (mini / base / pro), and embedding storage/lookups.

  Cut them by shortening prompts, limiting context, caching, and choosing right-sized models.

GitHub Models

Chatting with Copilot

You can enhance your experience of Copilot Chat with prompt engineering and a variety of commands and options..

Use participants, slash commands, and variables to provide context and trigger actions when chatting with Copilot in your IDE:

- **Chat participants** — refer to scopes/resources: @workspace (files), @github (repo/PR), @file, @terminal

- **Slash commands** — quick actions: /explain, /fix, /new, /tests, /doc

- **Chat variables** — inject context: #selection, #filePath, #cursor, #branch

GitHub Copilot Commands and Options

Prompt Engineering

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.

- The GitHub Copilot Team

What is Prompt Engineering?

Prompt Engineering is the strategic crafting of user inputs to guide the AI towards producing desired code outputs.

This involves techniques like role prompting, contextual detailing, and iterative refinement, which collectively enhance the interaction between the developer and Copilot.

Structured Prompt Engineering

- **Define Goal**
  - One sentence on what you want.

- **Provide Context**
  - Key constraints, environment, and inputs.

- **Include Sources and Links**
  - Only the needed code/spec links or snippets.

- **Specify Expectations**
  - Exact output format and style.

Example:

Role-play as a senior Python reviewer mentoring a programmer.
Goal: Generate deterministic pytest unit tests for the Producer–Consumer in apps/python/ex1.
Context: Python 3.13, stdlib only; use delay=0 in tests; enforce ValueError on negative count or delay; stop on unique SENTINEL; queue empty after run.
Sources: apps/python/ex1/specs.md and the files producer.py, consumer.py, run.py, logging_utils.py, __init__.py in copilot-base.
Expectations: Output exactly one test file (tests/test_producer_consumer.py) in a single fenced code block with the filepath comment; cover count=0 and normal flow; assert sentinel consumed once; keep assertions concise; no prose; ≤80 lines.

Using GitHub Skills

When using the GPT and Claude Sonnet models, Copilot Chat has access to a collection of skills to fetch data from GitHub, which are dynamically selected based on the question you ask.

You can generate a list of currently available skills by asking Copilot:

- *What skills are available for C# developers?*

- *What skills are available for Web developers using Angular.io, TypeScript, and Node?*

- *What skills are available for E2E testing?*

- *What skills are available repository management with git?*

Use Prompting Across the SDLC

Observe the technique being used in the following prompt:

"You are an experienced C++ software engineer.

- Provide a detailed explanation of how Copilot can assist in each phase of the SDLC, including planning, design, development, testing, deployment, and maintenance.

- Include specific C++ examples of tasks or activities within each phase where Copilot can be particularly beneficial."

- The output should be formatted as markdown with clear delineation of phases.

Interactive Coding with Copilot

To help Copilot generate accurate suggestions when writing code:

- Add a top-level comment block describing the purpose of the file

- Front load as many imports as needed (import / include / requires / etc.)

- Create a detailed comment block describing the purpose of an operation or UDT

- Use sample code as a starting point

- Have related content open in other tabs

AI Prompt Techniques

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 strings as input
2    //and returns the string with the maximum length.
3    //Ensure that the function works correctly for arrays that are empty,
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   }
```

Helpful Patterns

* Use descriptive variable names to make your intentions clear.

    totalSampleCount = 1000

* Maintain consistent naming conventions for variables and functions.

    i.e. using camelCase for variable names consistency

* Define method signatures with unambiguous parameter names and types.

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

* 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.

Helpful Patterns

```
* 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.

* 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

* Test your code.

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

Secure coding

Block files from Copilot

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

31
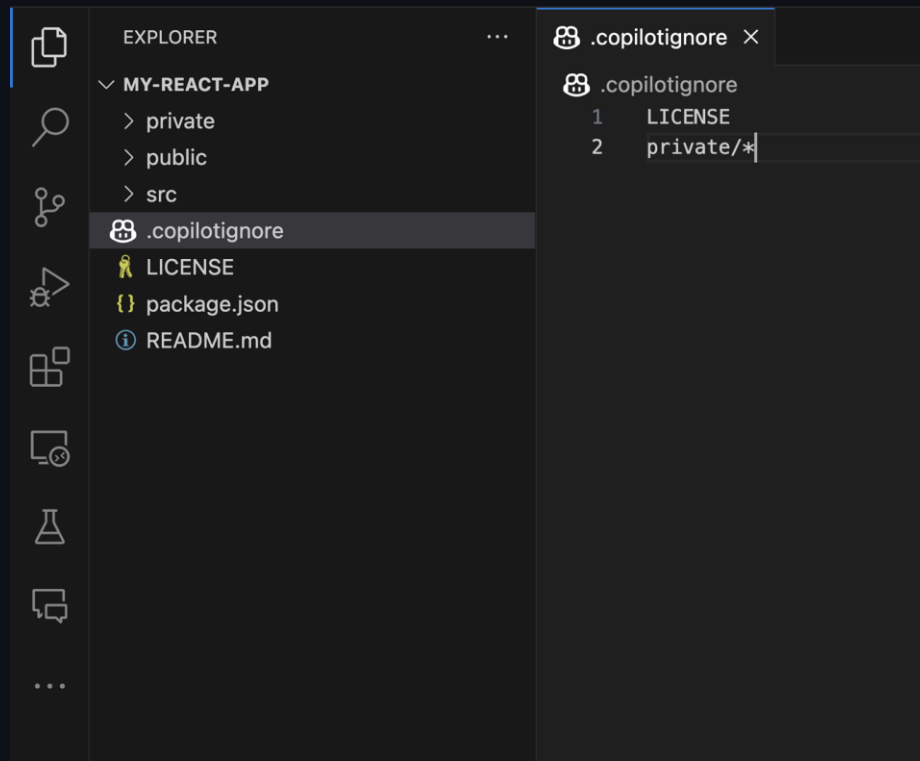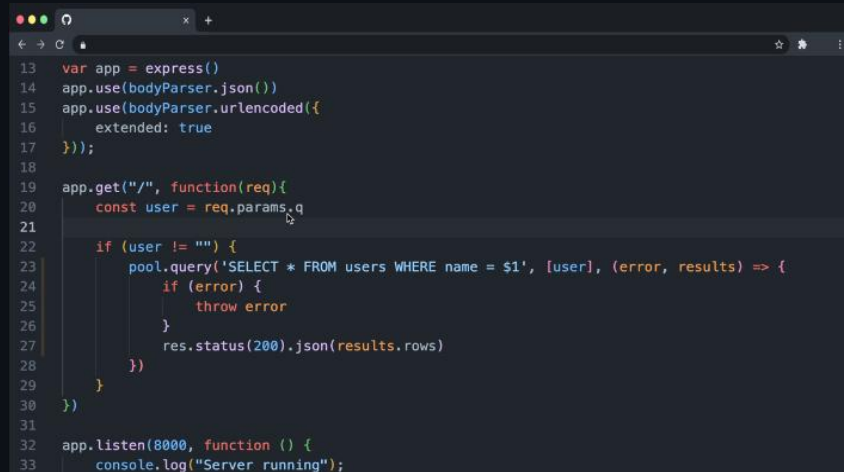
Copilot Restricted Content feature

- The Copilot Restricted Content feature allows repository owners to control the usage of Copilot across their codebases.

- This includes specific files, folders, or entire repositories, even if they're not hosted on GitHub.

- The goal is to provide CfB customers a flexible way of controlling what content Copilot can access for prompt crafting and where it can insert code.
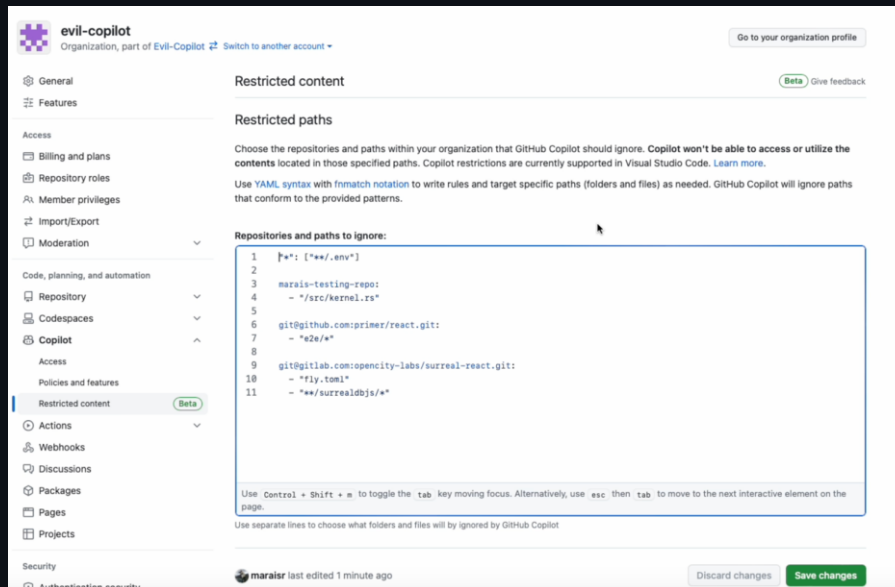
Copilot and Secure Coding

- 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

33

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

Copilot + GHAS

GitHub Copilot and GitHub Advanced Security (GHAS) serve different purposes and are not replacements of each other.
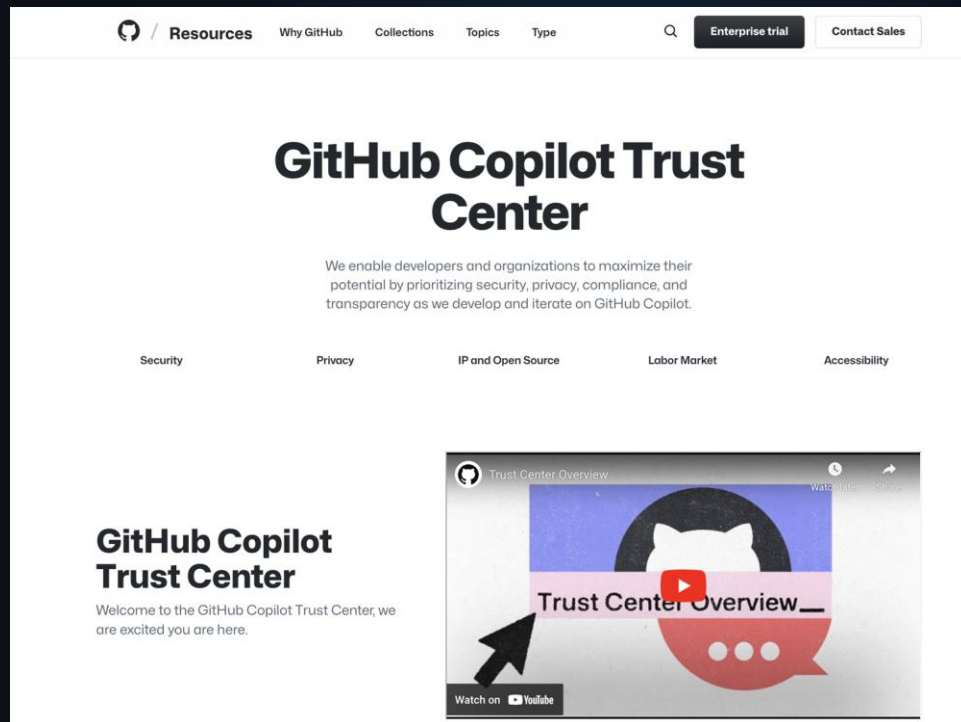
- Copilot is not a replacement of GHAS features

- 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

Security & Trust

Copilot Trust Center

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

GitHub Copilot Trust Center - GitHub Resources

Wrap Up

Thank you