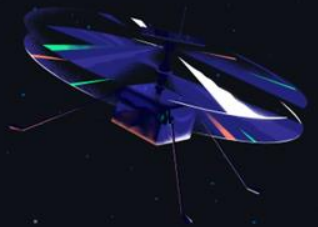




GitHub Copilot Prompt Engineering

Andrew Scoppa



Helpful Resources

- [Getting started with GitHub Copilot](#)
- [Configuring GitHub Copilot in your environment](#)
- [GitHub Copilot Trust Center](#)

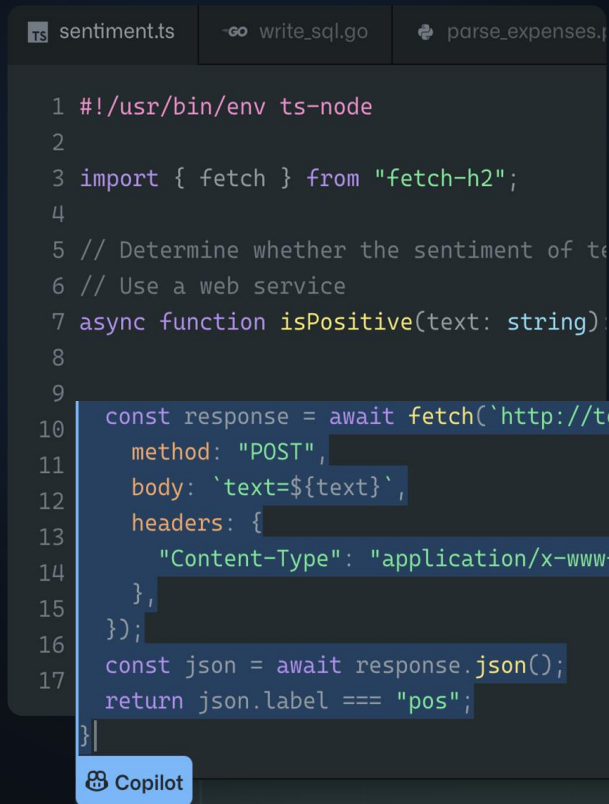


Agenda

- Introduction
- Best Practices
- Prompting Techniques
- Coding Demos
- Q & A

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


- GitHub Copilot is there to enhance daily work
 - Like a smart assistant or mentor by your side
- Draws context from text & code in open tabs
- Powered by OpenAI
 - Copilot uses a transformative model
 - Think of something like Google Translate
- Trained on large datasets to ensure accuracy
 - It even can help with HTML and markdown!
- Available as an extension to IDEs and editors



```

1  #!/usr/bin/env ts-node
2
3  import { fetch } from "fetch-h2";
4
5  // Determine whether the sentiment of text is positive
6  // Use a web service
7  async function isPositive(text: string): Promise<boolean> {
8
9
10     const response = await fetch(`http://localhost:3000/api/sentiment`, {
11       method: "POST",
12       body: `text=${text}`,
13       headers: {
14         "Content-Type": "application/x-www-form-urlencoded",
15       },
16     });
17     const json = await response.json();
18     return json.label === "pos";
19   }

```

 Copilot





Helps developers stay in the flow throughout the entire SDLC



What is Prompt Engineering?



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

A nuanced understanding of prompt engineering allows developers to optimize their use of Copilot, ensuring they get the most out of this advanced coding assistant.

```
15  /*
16  Create a function 'sqrGen' which returns a lamda function
17  that will square the argument passed to the lambda
18  Example usage:
19  auto sqr = sqrGen(); int result = sqr(5); // result will be 25
20  ✨
21  auto sqrGen() {
22  ... return [](int m) -> int { return m * m; };
23  }
```



Best Practices



To be understood by the machine, you have to behave like an engineer

**Write Clear
Instructions**

**Split Complex Tasks into
Simpler Subtasks**

**Use External
Tools**

**Provide
References**

Give the Model Time to Think

Test Changes



Prompt Quality

For better prediction

Vague Prompt

Q: "How do I implement this API?"

A: "You can implement an API by writing the necessary code and setting up your server to handle requests"

- General
- Non-actionable
- Too broad
- Not very helpful

General Prompt, Little Context

Q: "How do I implement an API in Java?"

A: "To implement an API in Java, you need to use a framework like Spring Boot. You will define your endpoints, handle HTTP requests, and set up your server to process these requests."

- Some context (framework)
- Narrows down response
- Still lacks depth

Specific Prompt, Detailed Context

Q: "How do I implement a RESTful API in Java using Spring Boot? Provide steps to set up a basic API with a sample endpoint that returns a list of users"

A: "1) Set up your project ...
2) Create your Main application class:

```
package com.example.demo;  
import org.springframework.boot.SpringApplication;  
...
```

- Clear, detailed & comprehensive
- Instructions and code samples
- More accurate

Using Prompt Engineering

You need to create a function named 'job' that simulates a workload by iterating a specified number of times, sleeping for a random time during each iteration.



Create a Plan

Define the Objective

- Clearly state the goal: To create a function that simulates a workload with random delays

Break Down the Task

- Identify the key components:
function definition, loop structure, etc

Create Prompts for Each Component

- Write specific prompts to generate code for each component.

Combine the Generated Code

- Integrate the code generated from each prompt into a complete function.



Example Prompt steps

Function Definition:

- Write a Python function named `job` that takes an integer parameter `workload`.

Loop Structure:

- Add a for loop inside the `job` function that iterates from `workload` down to 1.

Random Number Generation:

- Inside the loop, generate a random float between 0.5 and 1.5 using the `random` module.

Sleep for Random Duration:

- Use the `time.sleep` function to pause the execution for the generated random duration.

Console Output:

- Print the current step number to the console inside the loop.



Example solution

Create a lambda function named 'job' that simulates a workload.

Arguments:

- 'workload': an integer representing the number of steps in the workload

Returns: void

Details:

- Use a 'for loop' to to simulate a workload by iterating from the given workload value down to 1.
- For each iteration, it performs the following steps:
 - 1) Generates a random sleep time between 0.5 and 1.5 seconds.
 - 2) Pauses the execution for the generated sleep time.
 - 3) Prints a message indicating the completion of the current step.

Errors:

- If the workload is less than or equal to 0, the function should throw an `invalid_argument` exception with the message "Invalid workload value".

Example:

```
job(10);
```



Use Pseudocode for Granular Prompting

1. Function Definition:

- Define a function `binary_gcd` that takes two unsigned integers `numerator` and `denominator`.

2. Base Cases:

- If `numerator` is 0, return `denominator`.
- If `denominator` is 0, return `numerator`.

3. Both Numbers Even:

- If both `numerator` and `denominator` are even:
 - Return 2 times the result of `binary_gcd(numerator / 2, denominator / 2)`.

4. One Number Even, One Odd:

- If `numerator` is even and `denominator` is odd:
 - Return the result of `binary_gcd(numerator / 2, denominator)`.
- If `numerator` is odd and `denominator` is even:
 - Return the result of `binary_gcd(numerator, denominator / 2)`.

5. Both Numbers Odd:

- If both `numerator` and `denominator` are odd:
 - If `numerator` is greater than `denominator`:
 - Return the result of `binary_gcd((numerator - denominator) / 2, denominator)`.
 - Else:
 - Return the result of `binary_gcd((denominator - numerator) / 2, numerator)`.



At a Lower Level...

Given the following:

Mask: 0b1010

Array 1: [0b1010, 0b1100, ..., 0b1111]

Array 2: [0b0110, 0b0011, ..., 0b0000]

How would you use prompt engineering to generate a transformation procedure that produces the following result?

Array 1: [0b0010, 0b0110, ..., 0b0101]

Array 2: [0b1110, 0b1001, ..., 0b1010]

The solution should be a function that takes the mask, arrays, and size as arguments.



Detailed Prompt

Generate a function 'conditionalBitSwap' that takes in 4 arguments:

- 'arr1': an array of char
- 'arr2': an array of char
- 'mask': a char that represents a char bit mask.
- 'arrLen': a size_t that represents the length of the array.

Implementation details:

- the conditionalBitSwap function swaps the bits in the same positions of the elements of two arrays, but only if the corresponding bit in the mask is set and the bits to be swapped are different.
- the conditionalBitSwap function should not return anything.

Example:

```
arr1 = [0b1010, 0b1100]
arr2 = [0b0110, 0b0011]
mask = 0b1010
```

After calling conditionalBitSwap(arr1, arr2, mask, 2), arr1 and arr2 should be:

```
arr1 = [0b0010, 0b0110]
arr2 = [0b1110, 0b1001]
```



Q & A



Thank you





GPT Models

Copilot is always evolving and much of this evolution is powered by the evolving models behind it



Past (Codex)

- AI Hallucination problem
- Outdated/Deprecated code
- Lack of contextual understanding
- Failing to suggest best practices
- But it worked!

Present (GPT-3.5 GPT-4)

- Faster
- More efficient in regards to GPU usage
- Larger context window
- More recent training data

Future (Specific Models)*

- Code-specific models
- Enhanced experience and quality through fine-tuning
- Customization of models

* GPT keeps evolving and the latest can be found in github.com/blog



How does a LLM work?

“

A LLM combines deep learning techniques with extensive training data to understand and generate human-like text based on the input it receives



Inference

The input is tokenized, and processed through the different transformer layers to capture context and relationship.

The model generates predictions for the next token and repeats until a complete response is generated.

This response can be controlled or guided through **prompt engineering** to achieve desired outcomes



Training

The model is trained on large amounts of text data, with parameters to learn patterns, grammar and facts



Architecture

Most use a neural network called Transformer, effective at handling sequential data



Prediction

Find the most likely output for next token based on patterns learned during training

<>Tokenization

Input is broken down into smaller units (words, subwords, characters)



Contextual

Surrounding tokens are used to understand meaning and predict next token



Fine-Tuning

Apply a specific dataset for a particular task after training (e.g. code, java, legal documents)