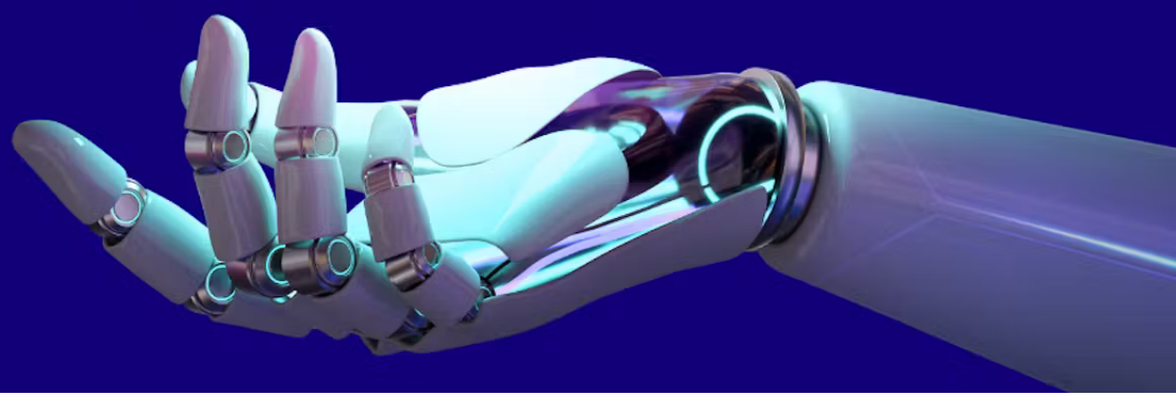
@ black girl bytes

Rizel Scarlett

Follow

PROMPT ENGINEERING WITH GITHUB COPILOT



A Beginner's Guide to Prompt Engineering with GitHub Copilot

Rizel Scarlett

Apr 4, 2023 • 12 min read

PLAY THIS ARTICLE

00:00 -00:00

TABLE OF CONTENTS

What is GitHub Copilot?

Can GitHub Copilot code on i...

How does GitHub Copilot wo...

What is prompt engineering?

Best Practices for Prompt En...

- Provide high-level conte...
- Provide specific details
- Provide examples
  - Zero-shot learning
  - One-shot learning
  - Few-shot learning
  - No example
  - With example

Additional tips

Iterate your prompts

Keep a tab opened of rel...

Give your AI assistant an...

Use predictable patterns

Use consistent, specific ...

Use good coding practices

Beyond your editor

Start practicing prompt engi...

Let's keep learning together

When I started using GitHub Copilot and other generative AI tools, I felt frustrated because I wasn't receiving the expected results. How were people feeling so successful with these tools, and why weren't they doing what I wanted? For example, I would ask GitHub Copilot to solve a LeetCode problem for me. The GitHub Copilot icon would spin to indicate that it's thinking, and then I would receive an incongruent suggestion or no suggestion at all. I was annoyed, but it turns out – I was using it wrong! After more experimentation, I improved my communication methods with GitHub Copilot by providing context, examples, and clear instructions in the form of comments and code. Later on, I learned that this practice is called prompt engineering. In this blog post, I'll discuss top tips to help you get the most out of GitHub Copilot.

First, let's start with the basics for folks who are unfamiliar with GitHub Copilot or prompt engineering.

What is GitHub Copilot?

GitHub Copilot is an AI pair programmer developed by GitHub and GitHub Copilot is powered by OpenAI Codex, a generative pre-trained language model created by OpenAI.that provides contextualized code suggestions based on context from comments and code. To use it, you can install the GitHub Copilot extension available to you in the following Integrated Development Environments (IDEs):

- Visual Studio
- Visual Studio Code
- Neovim
- JetBrains IDEs (IntelliJ, PyCharm, WebStorm, etc)

Can GitHub Copilot code on its own?

At GitHub, we use the terms "AI pair programmer," "AI assistant," and "Copilot" to since this tool cannot work without you – the developer! In fact, AI systems can only perform the tasks that developers program them to perform, and they do not possess free will or the ability to make decisions independently. In this case, GitHub Copilot leverages context from the code and comments you write to suggest code instantly! With GitHub Copilot, you can convert comments to code, autofill repetitive code, and show alternative suggestions.

How does GitHub Copilot work under the hood?

Under the hood, GitHub Copilot draws context from comments and code, instantly suggesting individual lines and whole functions. OpenAI Codex, a machine-learning model that can translate natural language into code, powers GitHub Copilot

What is prompt engineering?

Prompt engineering is the practice of giving an AI model specific instructions to produce the results you want. A prompt is a sequence of text or a line of code that can trigger a response from an AI model. You can liken this concept to receiving a prompt for an essay. You can receive a prompt to write an essay about a time you overcame a challenge or a prompt to write about a classic book, such as the Great Gatsby. As a result, you give a response to the prompt based on what you've learned.

A large language model or LLM will behave similarly.

Here's another illustration of prompt engineering: When I learned to code, I participated in an activity where I gave a robot instructions on how to make a sandwich. It was a fun, silly activity that taught me that:

- Computers can only do what you tell them to do
- You need to be very specific with your instructions
- They're better at taking orders one step at a time

- Algorithms are just a series of instructions

For example, if I were to tell the "robot" to make a sandwich, I need to tell it:

1. Open the bag of bread
2. Take the first two slices of bread out of the bag
3. Lay the slices of bread side by side on the counter
4. Spread peanut butter on one slice of bread with a butter knife
5. Et cetera, et cetera, et cetera

Without those clear instructions, the robot might do something silly, like spread peanut butter on both slices of bread, or it might not do anything at all. The robot doesn't know what a sandwich is, and it doesn't know how to make one. It just knows how to follow instructions.

Similarly, GitHub Copilot needs clear, step-by-step instructions to generate the code that best helps you.

Let's discuss best practices for prompt engineering to give clear instructions to GitHub Copilot and generate your desired results.

Best Practices for Prompt Engineering with GitHub Copilot

Provide high-level context followed by more detailed instructions

The best technique for me is providing high-level context in a comment at the top of the file, followed by more detailed instructions in the form of comments and code.

For example, if I'm building a to-do application. At the top, I'll write a comment that says, "Build a to-do application using Next.js that allows user to add, edit, and delete to do items." Then on the following lines, I'll write a comment to create the:

- to-do list component, and I'll let GitHub Copilot generate the component underneath the comment
- button component, and I'll let GitHub Copilot generate the component underneath the comment

- input component, and I'll let GitHub Copilot generate the component underneath the comment
- add function, and I'll let GitHub Copilot generate the function underneath the comment
- edit function, and I'll let GitHub Copilot generate the function underneath the comment
- delete function, and I'll let GitHub Copilot generate the function underneath the comment
- etcetra, etcetera, etcetera...

Here's an example of me using this technique to build a house with p5.js:

In the following GIF, I write a comment at the top describing at a high-level what I want p5.js to draw. I want to draw a white house with a brown roof, a red door, and a red chimney. Then, I write a comment for each element of the house, and I let GitHub Copilot generate the code for each element.

Provide specific details

When you provide specific details, GitHub Copilot will be able to generate more accurate code suggestions. For example, if you want GitHub Copilot to retrieve data from an API, you need to tell it what type of data you want to retrieve, how to process the data, and what API endpoint you're hoping to hit.

Here's an example of a non-specific comment (prompt) where GitHub Copilot will be less likely to generate a useful suggestion:

```
// Get data from API
function getData() {
  //code goes here
}
```

In the example above, GitHub Copilot didn't generate anything for me besides the name of the function and the opening and closing brackets.

Here's an example of a specific comment (prompt) where GitHub Copilot will be more likely to generate a useful suggestion:

```
// Pass in user ids and retrieve
async function getUserData(userI
    const response = await fetch
    const data = await response.
```

```
    return data;
}
```

Provide examples

In addition to telling GitHub Copilot what you want it to do, you can also show it what you want it to do with examples in your preferred coding style. While GitHub Copilot is using an AI model that is already trained on a large amount of data, providing examples to GitHub Copilot help it understand the context and constraints of a particular code snippet. Showing AI models examples of what you want them to do is a common practice in machine learning.

Here are a few approaches engineers use to train AI models:

- Zero-shot learning
- One-shot learning
- Few-shot learning

I asked ChatGPT to explain the difference between these methods to me like I was 5 years old, and here's what it gave me:

Zero-shot learning

This is like when you see a new toy you've never seen before, but you know what it is because it looks like other toys you've seen. For example, if you've seen many different kinds of stuffed animals, you might be able to recognize a new one even if you've never seen that exact one before. Similarly, a computer can learn to recognize new things by using what it knows about similar things to make educated guesses about the new thing.

One-shot learning

This is like when you see a new toy for the first time, and your mom or dad tells you what it's called and what it does. After hearing about it just once, you remember what it is and can recognize it the next time you see it. A computer can learn to recognize new things in the same way by being shown just a few examples and learning from them.

Few-shot learning

Few-shot learning is like learning to do something new by seeing only a few examples. For example, let's say you want to learn how to draw a cat, but you've only seen a few pictures of cats before. With few-shot learning, you can still learn how to draw a cat even if you haven't seen many cats before. It's like a magic power that helps you learn new things quickly, even if you don't have a lot of examples to learn from!

Here's what happened when I showed GitHub Copilot examples vs. when I didn't show it examples:

No example

```
# Create a list of the first 10 p
primes = []

for num in range(2, 30):
    if all(num % i != 0 for i in
        primes.append(num)
```

```
print(primes)
```

In the code snippet above, GitHub Copilot will accurately return the first 10 numbers, however it's inefficient because it will loop through all 29 numbers between 2 and 30. We can get more efficient results by providing an specific example of what we want GitHub Copilot to do.

With example

```
# Create a list of the first 10 p
# Example: [2, 3, 5, 7, 11, 13, 1
primes = []
```

```
for num in range(2, 30):
    if all(num % i != 0 for i in
        primes.append(num)
    if len(primes) == 10:
        break
print(primes)
```

In the above code snippet, GitHub Copilot will return the first ten prime numbers and stop when it's found all 10. My goal was to get accurate but fast results, and GitHub Copilot successfully achieved that after I nudged it in the right direction.

Additional tips

Iterate your prompts

If your initial prompt elicits the desired response, delete the generated code suggestion, edit your comment with more details and examples, and try again! It's a learning process for you and GitHub Copilot. The more you use it, the better you'll get at communicating with GitHub Copilot.

Keep a tab opened of relevant files in your IDE

Currently, GitHub Copilot does not have the ability to gain context for your entire codebase. However, it can read your current file and any files that are opened in your IDE. I've found it helpful to keep an open tab of relevant files that I want GitHub Copilot to reference. For example, if I'm writing a function that uses a variable from another file, I'll keep that file open in my IDE. This will help GitHub Copilot provide more

help GitHub Copilot provide more accurate suggestions.

Give your AI assistant an identity

I received this advice from Leilah Simone, a Developer Advocate at BitRise. I've yet to try this with GitHub Copilot, but it was helpful advice. It helps to control the type of response the user will receive. In Leila's case, she requests that ChatGPT behave as a senior iOS engineer. She says, "It's helped her reduce syntax and linting issues."

Use predictable patterns

As we saw in the many examples above, GitHub Copilot will follow patterns in your code. AI enthusiast and Developer Content Creator, YK aka CS Dojo, shares how he uses this to his advantage:

itter.com/ykdojo/status/164062930059

%

Use consistent, specific naming conventions for

variables and functions that describe their purpose

When declaring variables or functions, use names specific to the variable's purpose. This will help GitHub Copilot understand the context of the variable and generate more relevant suggestions. For example, instead of using a generic variable name like "value," use a more specific name like "input_string" or "output_file."

GitHub Copilot will also use the naming conventions you use in your code. For example, if you use camelCase for variables, GitHub Copilot will suggest camelCase variables. If you use snake_case for variables, GitHub Copilot will suggest snake_case variables.

Use good coding practices

While GitHub Copilot can be a powerful tool for generating code suggestions, it's important to remember that it's not a replacement for your own programming skills and expertise. AI models are only as good as the data they have been trained on, Therefore, it's important to use these tools as aids and not rely on them entirely. I encourage every user of

GitHub Copilot to:

- Review code
- Run unit tests, integration tests, and any other programatic forms of testing code
- Manually test code to ensure it's working as intended
- And to use good coding practices because GitHub Copilot will follow your coding style and patterns as a guide for its suggestions.

Beyond your editor

Currently, GitHub Copilot is an extension that is available in the most popular IDEs. There's also GitHub Copilot Labs, a separate experimental extension available with GitHub Copilot access. Copilot Labs can help you translate, debug, test, document, and refactor code. Additionally, we recently launched Copilot X, a suite of features that improve developer productivity outside of the IDE. Copilot X includes:

- Copilot for Docs - saves developers from scouring reams of documentation.
- Copilot for Pull Requests - helps you write better PR descriptions and to help your team review and merge PRs faster/

- Copilot Chat - a ChatGPT-like experience in your editor with GitHub Copilot chat.
- Copilot for CLI - helps you remember shell commands and flags to run commands in your terminal faster.
- Copilot Voice - write and edit code, navigate the codebase, and control Visual Studio Code with your voice.

Copilot X features/products are currently in technical preview. If you'd like access to one of the features above, sign up for the waitlist at github.com/features/preview/copilot-x.

Start practicing prompt engineering with GitHub Copilot

The best way to learn how to use GitHub Copilot optimally is to start using it. You can sign up for access to GitHub Copilot here.

If you're a business leader and you'd like to give your entire team access to GitHub Copilot, sign up for GitHub Copilot for Business here.

Let's keep learning together

Thanks for reading this blog post. I'd love to hear from you! I'm still learning more about artificial intelligence, GitHub Copilot, and prompt engineering. If you're a GitHub Copilot fan and have learned how to use it to improve your developer workflow, share some of your tips in the comments below!

For similar content, follow GitHub and me on DEV.to! Also, check out other resources to learn more about prompt engineering:

- How to get Codex to produce the code you want
- Prompt Design from OpenAI
- Prompt Engineering 101: Introduction to Codex
- Prompt Engineering Guide by Elvis Saravia

Subscribe to my newsletter

Read articles from **Black Girl Bytes** directly inside your inbox. Subscribe to the newsletter, and don't miss out.

Enter your email address SUBSCRIBE

Did you find this article valuable?

Support **Rizel Scarlett** by becoming a sponsor. Any amount is appreciated!

Sponsor

See recent sponsors | Learn more about Hashnode Sponsors

GitHub

Artificial Intelligence

Machine Learning

General Programming

#PromptEngineering

WRITTEN BY

Rizel Scarlett

Follow

Developer Advocate @ GitHub

©2023 Black Girl Bytes

Publish with Hashnode

Powered by Hashnode - Home for tech writers and readers