
Effective Prompt Engineering
Andrew Scoppa

OpenAI, Microsoft & GitHub

Copilot

When the OpenAI innovation meets the Microsoft scale at GitHub, the home of developers

- Powered by GPT
- Secure and scalable by default on Microsoft Azure
- Built with developers at the main seat
- Integrated across the whole software supply chain
- Supported by the most popular IDEs

What is the role of a LLM?

> A Large Language Model (LLM) like GPT-4 plays the role of understanding and generating human-like text based on the input it receives.

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 Understanding
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)

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

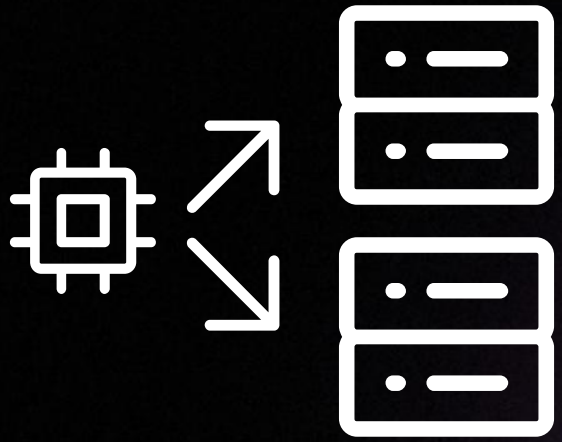
The model generates predictions for the next token and repeats until a complete response is generated.

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

Understanding the user’s intent

The Copilot Intent Classifier is used to determine what kind of assistance or response the user is seeking, such as generating code, explaining code, making changes to existing code, or providing general programming help.

In the context of the Copilot Intent Classifier, the LLM helps in interpreting the user's input, determining the intent behind it, and generating appropriate responses or code snippets.



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- Outdated/Deprecated code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

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  }
```

Using Prompt Engineering

- Prompt engineering applies to both Copilot and Copilot Chat.

- Copilot operates passively, offering suggestions as you write code. You can accept, reject, or modify these suggestions.

- Copilot Chat operates actively through a chat interface, where you can ask questions and receive detailed responses, explanations, and code snippets tailored to your queries.

- Effective prompt engineering is crucial for getting the best results from GitHub Copilot or Copilot Chat because it involves crafting clear, specific, and detailed instructions or questions.

Prompt Quality
For better prediction

Vague Prompt

Q: “How do I implement an 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

Best Practices

> To be *understood* by the machine, you have to *communicate* 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

Source: OpenAI's Prompt Engineering Strategies

Best Practices

Write Clear Instructions

- Models can't read minds
- Long outputs? Ask for brief
- Too simple outputs? Ask for expert-level writing
- Did not like the format? Exemplify the output

Tactics:

1) Include details on prompt
2) Ask the model to be a persona ("act like a senior Java engineer)
3) Use delimiters to separate parts of prompts
4) Specify steps required to complete the task
5) Provide examples
6) Say how long the output should be

Provide References

- References reduce chances of a confident wrong answer when asked about citations or URLs

Tactics:

1) Instruct to answer using reference text
2) Instruct to answer with citations from a reference text

Best Practices

Split Complex into Simpler Tasks

- Just like Software Engineering, solving a bigger problem is disproportionately harder than combining the solution of smaller, easier problems.
- The bigger the tasks, the higher the error rates

Tactics:

1) Summarize or filter previous dialogue
2) Construct full summaries of the code base

Give the model time to think

- Prefer to ask for a thought process instead of straight answer
- This gives the model time to “reflect” on the response and converge to a better output

Tactics:

1) Instruct to work out a solution instead of rushing to conclusion
2) Construct the context with snippets of information from the whole context to start guiding the model

Best Practices

Use External Tools

- Copilot and other Large Language Models are not the best tools for all jobs
- If a task can be done more reliably by another tool, use it and feed the results into Copilot

Tactics:

1) Sometimes a calculator is faster at math than a LLM
2) Give the model references to access contextual information (although Copilot does not yet have capabilities of browsing the web, the model it was trained on might have references to the same link or website)

Test Changes Systematically

- Just like code, the output of LLMs should be tested constantly
- The earlier and the smaller the test scope, the easier to spot and fix the problems

Tactics:

1) Deal with the output like if it was a function that needs unit testing to sparkle confidence

Prompt Engineering

Strategies and Techniques

Prompting Tasks

Problem definition:

- Define a lambda function in C++ that simulates a workload by iterating from a given number down to 1, generating random sleep times, pausing execution, and logging each step, and then executes this function with a specified number of steps.

Create a Plan

Define a lambda function ‘job’ in C++ that simulates a workload by iterating from a given number down to 1, generating random sleep times, pausing execution, and logging each step, and then executes this function with a specified number of steps.

Loop Structure:
- Use a for loop inside the function that iterates from workload down to 1.

Random Number Generation:
- Inside the loop, generate a random sleep time between 0.5 and 1.5.

Sleep for Random Duration:
- Pause the execution for the generated random duration.

Console Output:
- Print the current step number to the console inside the loop.

Error Handling:
If the workload is less than or equal to 0 throw an exception.

Example of Use:

```
try {
    job(5);
} catch (const std::exception &e) {
    std::cerr << "Error: " << e.what() << std::endl;
}
```

Example

```
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);
```

Prompting Tasks

Problem definition:

- Create a function that calculates the greatest common divisor of two numbers using the binary GCD algorithm.

Create a Plan

The plan for the binary_gcd function is to recursively calculate the greatest common divisor (GCD) of two unsigned integers by leveraging the properties of even and odd numbers to simplify the problem, reducing the size of the numbers step-by-step until one of them becomes zero, at which point the other number is the GCD.

Algorithm Steps:

- If either number is zero, return the other number as the GCD.
- If both numbers are even, divide both by 2 and multiply the result by 2.
- If one number is even and the other is odd, divide the even number by 2.
- If both numbers are odd, subtract the smaller number from the larger, divide the result by 2, and continue the process with the smaller number and the result.

Recursion: The function uses recursion to repeatedly apply the above steps until one of the numbers becomes zero, at which point the other number is the GCD.

Edge Cases: The function handles edge cases such as both numbers being zero, one number being zero, and both numbers being the same.

Testing: The test_BinaryGCD function uses assertions to verify that the binary_gcd function produces correct results for various test cases, ensuring the implementation is correct.

Example

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

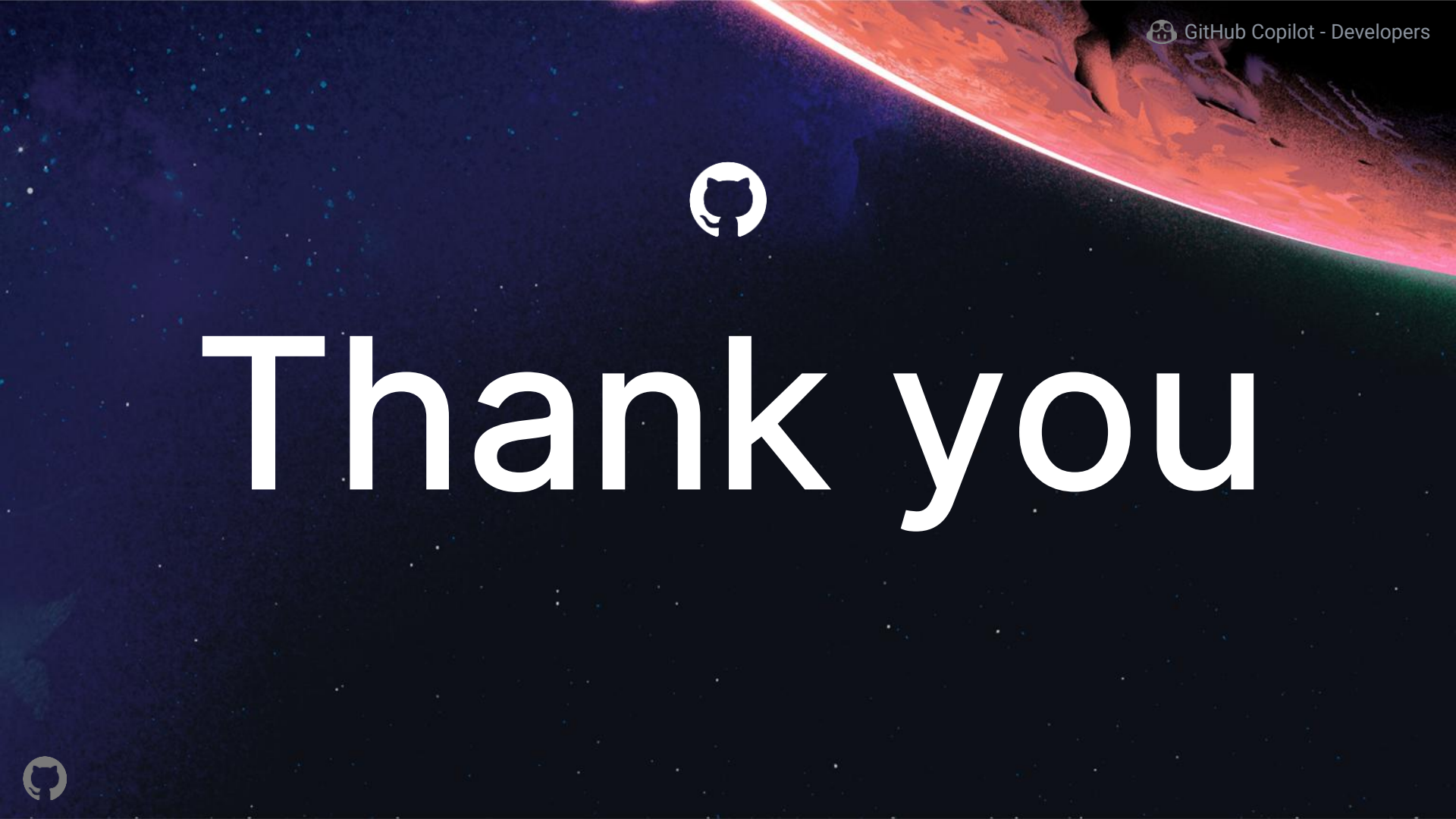
Array 1: [0b1010, 0b1100, …, 0b1111]
Array 2: [0b0110, 0b0011, …, 0b0000]

- 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]

- How would you describe a test such that you would be confident in the procedure’s efficacy?

Q & A


Thank you