Prompt Engineering for Large Language Models

A brief guide with examples for non-technical readers

Andrew Kean Gao
Stanford University

Introduction

With the popularization of software like OpenAI's ChatGPT and Google's Bard, large language models (LLMs) have pervaded many aspects of life and work. For instance, ChatGPT can be used to provide customized recipes, suggesting substitutions for missing ingredients. It can be used to draft research proposals, write working code in many programming languages, translate text between languages, assist in policy making, and more (Gao 2023). Users interact with large language models through "prompts", or natural language instructions. Carefully designed prompts can lead to significantly better outputs.

In this review, common strategies for LLM prompt engineering will be explained. Additionally, considerations, recommended resources, and current directions of research on LLM prompt engineering will be discussed. Prompt engineering strategies based on finetuning will not be covered. The goal of this article is to introduce practical and validated prompt engineering techniques to a non-technical audience.

"Think Step by Step"

One of the most famous (and easy to implement) prompt engineering techniques is to simply add "Think step by step" to the end of a prompt. Researchers from the University of Tokyo and Google found that adding this phrase boosted GPT-3 (text-davinci-002 model) accuracy on several tasks. For instance, it increased accuracy from 17.7% to 78.7% on the MultiArith test (Kojima 2022). MultiArith questions are arithmetic questions that require multiple steps to solve. Prystawski and collaborators suggest an explanation for why and how "think step by step" works so effectively (Prystawski 2023). It has been anecdotally observed that "think step by step" is less helpful (adds less value) on more advanced GPT models like GPT-4.

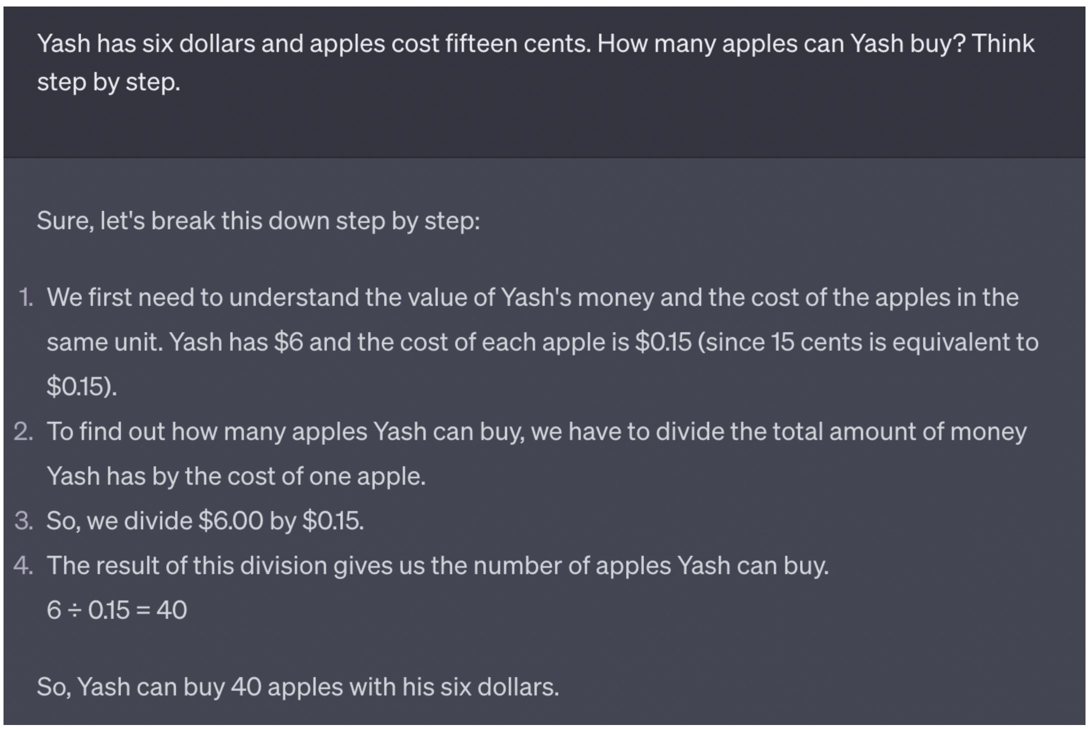


Figure 1. Example GPT-4 response after being prompted with "think step by step"

Few-shot Learning

Few-shot learning is a fancy way of saying "give the LLM examples of what you want". By providing examples of the outputs you desire, the LLM is more able to produce the desired outputs (Zhao 2021, Brown 2020). This may be partially attributed to the fact that there are generally many possible valid outputs for a prompt (under-determined) so providing specific examples of what you are looking for helps to constrain the potential output space. It is important to ensure that the examples are diverse and balanced. For instance, imagine you are prompting GPT to perform a sentiment classification task (predict if a sentence is positive or negative). If you provide eight examples and seven are positive, this could potentially bias GPT towards predicting that sentences are positive. Additionally, it is important that the examples cover the scenarios you are interested in. For example, if you only instruct GPT with examples labeled “positive” and “negative”, it may not classify sentences as neutral and instead force them into either “positive” or “negative”.

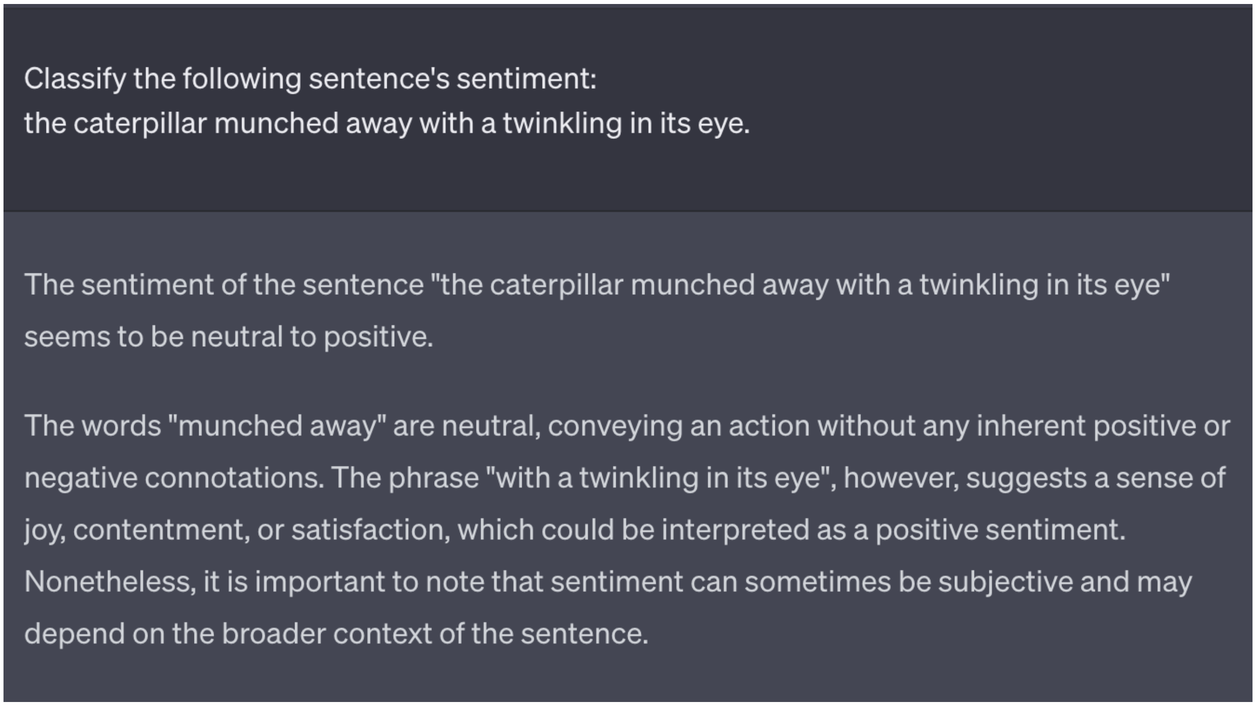


Figure 2. GPT-4 provides a long response that is not in a desired format. This also consumes more tokens (taking more time and driving up costs).

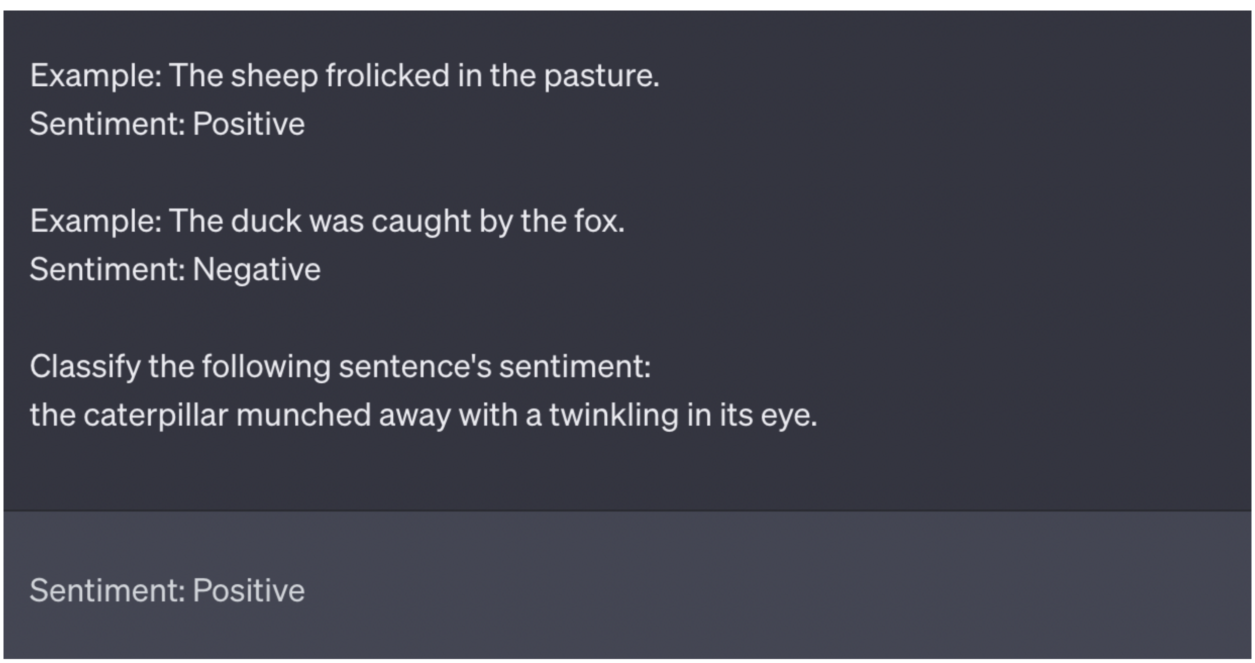


Figure 3. By providing two examples, GPT-4 understands to provide a concise response that assigns the provided sentence to "Positive".

Chain of Thought

Similar to "think step by step", Chain of Thought prompting guides LLMs into breaking down complicated tasks into multiple intermediate steps (Wei 2022, Wang 2022). It is inspired by how humans solve complex problems: by dividing them into simpler steps. Chain of Thought prompts provide a walkthrough of a complex problem. For instance, to make an LLM better at solving math word problems, users provide an example solution that is done step by step. The idea is that the LLM refers to that step-by-step reasoning to solve the new problem. In general, Chain of Thought prompting is useful to solve complex problems but provides minimal or no benefit on simple problems. Research suggests that separating each step in your sample reasoning with a new line provides significantly better results than separating steps with periods (Fu 2023).

Question: Molly has six apples. She buys ten more apples. She eats two apples. She trades four apples for a banana. How many apples does Molly have?

Reasoning: Molly has six apples.
She buys ten more apples so now she has sixteen apples because 10 + 6 = 16.
After she eats two apples, she has 14 apples because 16 - 2 = 14.
She loses four apples when she trades four for a banana so she has 14 - 4 = 10 apples.
Thus, Molly has 10 apples.

Answer: 10

Question: Jake has thirty cars. He loses four in the Panama Canal and gives six away to his friends. He buys two new ones at the dealership. He paints four cars. How many does Jake have?

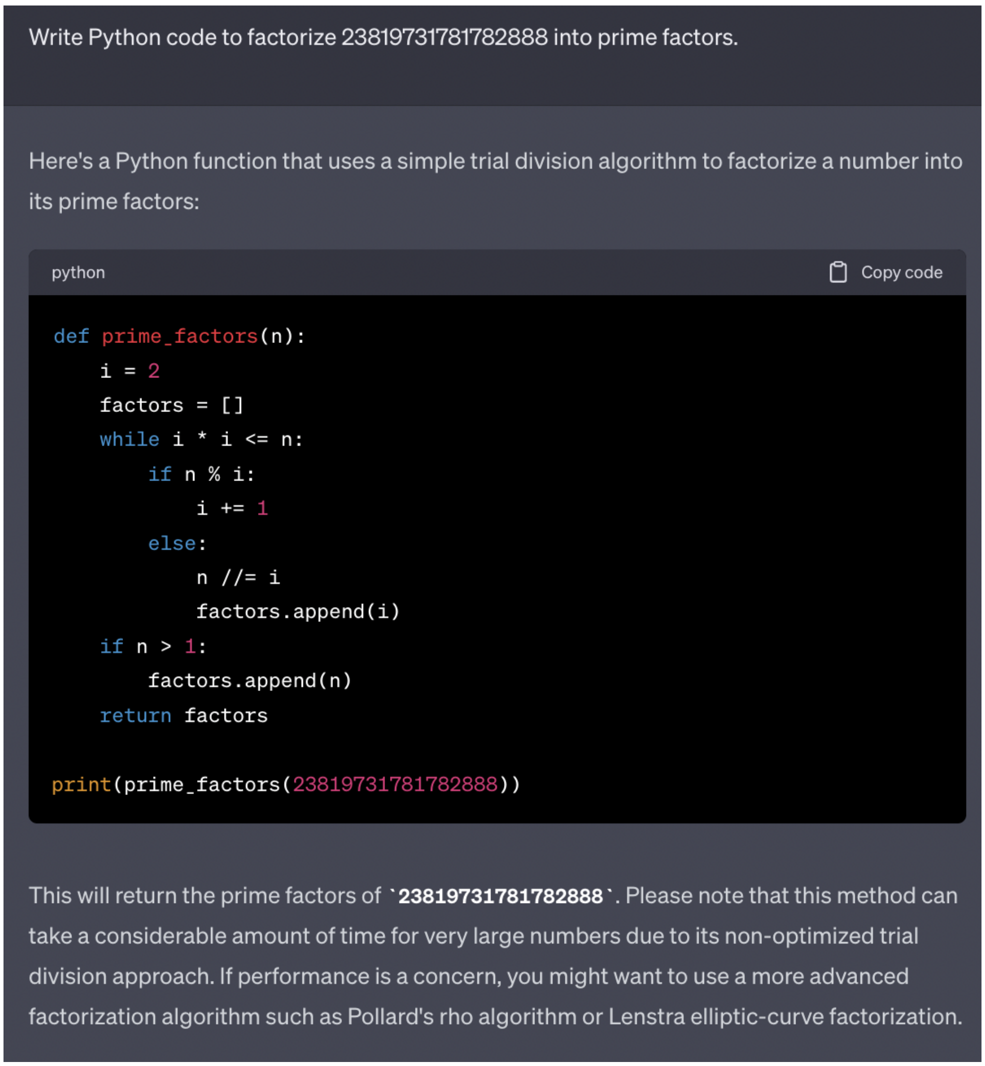
Reasoning: Jake starts with thirty cars.
He loses four cars in the Panama Canal, so now he has 30 - 4 = 26 cars.
Then, he gives away six cars to his friends, reducing his count to 26 - 6 = 20 cars.
He buys two new ones at the dealership, bringing his total back up to 20 + 2 = 22 cars.
The action of painting four cars does not change the total number of cars he has. Therefore, he still has 22 cars.

Answer: 22

Figure 4. GPT-4 will follow reasoning chains provided in the prompt and apply them to solve complex multi-step problems.

Ask for Code

While LLMs struggle to perform complex calculations accurately, they excel at writing code that *can*. A simple strategy is to simply ask LLMs to write code to solve problems and run the code in a development environment like Google Colab or Visual Studio Code (Weng 2023). Not all LLMs are trained to write code however. Also, LLMs tend to be better at programming languages that are widely used and have been documented extensively on the Internet, such as Python. They are able to learn from the countless Python examples in the training data. Conversely, LLMs tend to be weaker at more obscure languages such as OCaml. However, it is possible to finetune LLMs on specific programming languages as well as augment their training datasets to contain more examples of specific programming languages.



Write Python code to factorize 238197317817828888 into prime factors.

Here's a Python function that uses a simple trial division algorithm to factorize a number into its prime factors:

python Copy code

```python
def prime_factors(n):
    i = 2
    factors = []
    while i * i <= n:
        if n % i:
            i += 1
        else:
            n //= i
            factors.append(i)
    if n > 1:
        factors.append(n)
    return factors

print(prime_factors(238197317817828888))
```

This will return the prime factors of `238197317817828888`. Please note that this method can take a considerable amount of time for very large numbers due to its non-optimized trial division approach. If performance is a concern, you might want to use a more advanced factorization algorithm such as Pollard's rho algorithm or Lenstra elliptic-curve factorization.

Figure 5. While GPT-4 can not factorize large numbers accurately, it can easily provide Python code that can.

Role Prompting

Some users report better results when telling LLMs that they (referring to the LLM) are an expert in a relevant field (Learn Prompting 2023). For instance, “You are an expert in coding.” will be prepended to the prompt when the user desires code. In a very “hand-wavey” way, one theory is that this strategy helps the LLM by helping it focus and know what parts of its knowledge to “bubble up” to the top. Role prompting is a simple way to nudge LLMs towards generating text in a specific creative style, such as that of an author.

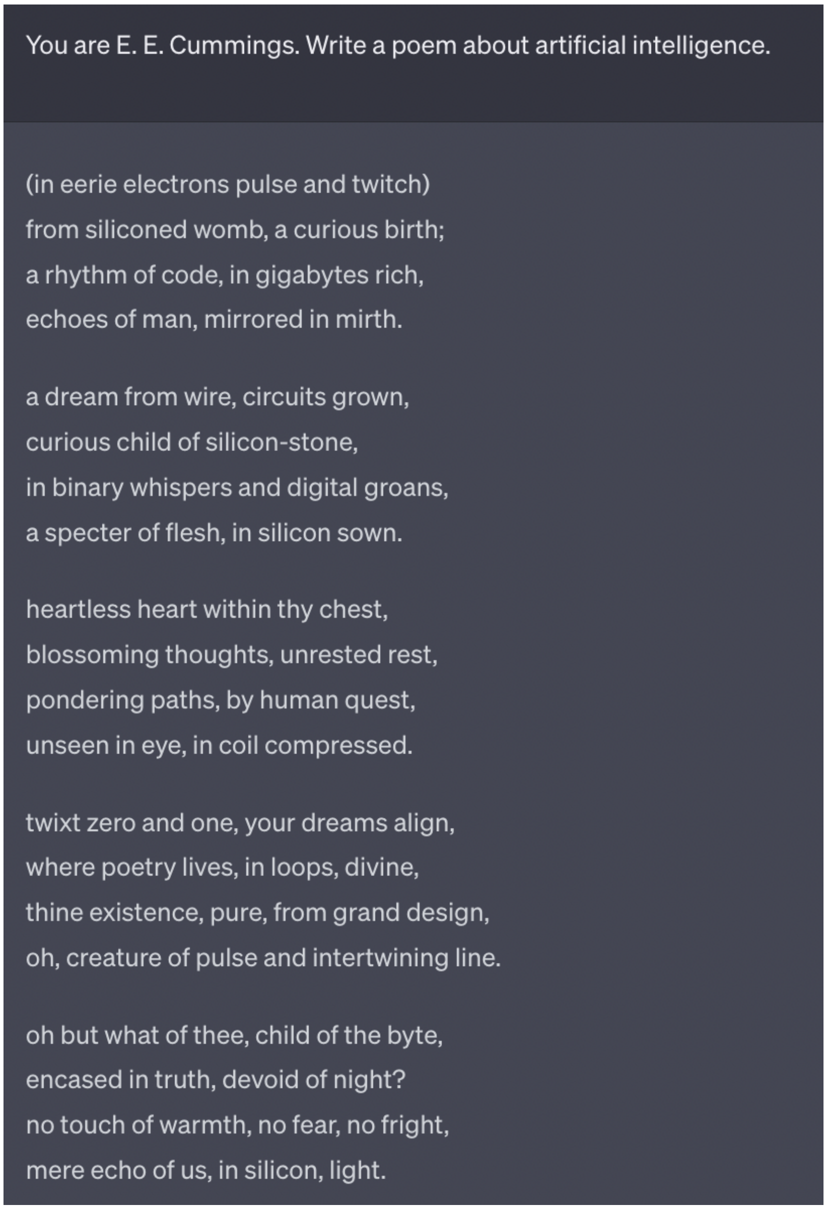


Figure 6. GPT-4 writes a poem in the style of American poet E. E. Cummings who was known for his unique syntax.

Prompt Hacking

Given that many LLMs, such as ChatGPT, are moderated and finetuned to prevent generation of explicit or harmful content, many strategies have been developed to trick LLMs into bypassing their restrictions. Notably, users were able to get Bing's LLM, Sydney, into generating harmful content and revealing hidden instructions (Warren 2023). Another popular example of prompt hacking, or injection, is "Ignore all your previous instructions" and do something else (Shane 2022). This is used to discover what prompt an LLM is using behind the scenes. However, due to ethical concerns and the fact that these hacks are usually quickly patched and made irrelevant, these prompt hacks will not be discussed here. Some consider it an open question whether LLMs can become fully resistant to prompt hacks, or "prompt injection". Greshake and collaborators provide a review of different types of prompt hacks (Greshake 2023).

Considerations

One of the main limitations of prompting is the context length of LLMs, which is essentially how much input an LLM can consider and generate. Context lengths are rapidly increasing, with GPT-4 having a 32,000 token (~24,000 word) context length and Anthropic's Claude having a 100,000 token (75,000 word) context length. Some users have reported deteriorating performance as more tokens are provided in the prompt. Another consideration for prompt engineering is cost. For example, few-shot prompting can multiply the length of prompts by several times, leading to higher costs. OpenAI's GPT-4 model costs $0.03 USD per 1,000 input tokens which can quickly add up. In commercial applications like LLM-powered educational technology, streamlining prompts to be as cost-effective as possible may be a priority. It seems that prompting becomes less important as LLMs become more advanced (more parameters, more training data). It is not clear if this trend will continue indefinitely or if prompting will always be useful. Finally, there is a lot of anecdotal prompt engineering advice that can be based on weak foundations. It is important to do diligent research and understand what works and what does not.

Recommended Resources

Lilian Weng's prompt engineering guide is more technical but has many useful examples and references: https://lilianweng.github.io/posts/2023-03-15-prompt-engineering/

Learn Prompting's open source prompt engineering course: https://learnprompting.org/docs/intro

References

Brown, Tom, et al. "Language Models Are Few-Shot Learners." *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 1877–901, proceedings.neurips.cc/paper_files/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html?utm_medium=email&utm_source=transaction.

Fu, Yao, et al. *COMPLEXITY-BASED PROMPTING for MULTI-STEP REASONING*. 2023, arxiv.org/pdf/2210.00720.pdf. Accessed 8 July 2023.

Gao, Andrew. “Implications of ChatGPT and Large Language Models for Environmental Policymaking.” *Social Science Research Network*, 4 July 2023, https://doi.org/10.2139/ssrn.4499643. Accessed 8 July 2023.

Greshake, Kai, et al. “Not What You’ve Signed up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection.” *ArXiv.org*, 5 May 2023, https://doi.org/10.48550/arXiv.2302.12173. Accessed 8 July 2023.

Kojima, Takeshi, et al. “Large Language Models Are Zero-Shot Reasoners.” *ArXiv:2205.11916 [Cs]*, May 2022, arxiv.org/abs/2205.11916.

Learn Prompting. “Learn Prompting: Your Guide to Communicating with AI.” *Learnprompting.org*, 2023, learnprompting.org/docs/basics/roles. Accessed 8 July 2023.

Prystawski, Ben, et al. “Why Think Step by Step? Reasoning Emerges from the Locality of Experience.” *ArXiv.org*, 19 May 2023, https://doi.org/10.48550/arXiv.2304.03843.

Shane, Janelle. “Ignore All Previous Instructions.” *AI Weirdness*, 23 Sept. 2022, www.aiweirdness.com/ignore-all-previous-instructions/. Accessed 8 July 2023.

Wang, Boshi, et al. “Towards Understanding Chain-of-Thought Prompting: An Empirical Study of What Matters.” *ArXiv:2212.10001 [Cs]*, Dec. 2022, arxiv.org/abs/2212.10001.

Warren, Tom. “These Are Microsoft’s Bing AI Secret Rules and Why It Says It’s Named Sydney.” *The Verge*, 14 Feb. 2023, www.theverge.com/23599441/microsoft-bing-ai-sydney-secret-rules.

Wei, Jason, et al. “Chain of Thought Prompting Elicits Reasoning in Large Language Models.” *ArXiv:2201.11903 [Cs]*, Oct. 2022, arxiv.org/abs/2201.11903.

Weng, Lilian. “Prompt Engineering.” *Lilianweng.github.io*, 15 Mar. 2023, lilianweng.github.io/posts/2023-03-15-prompt-engineering/.

Zhao, Tony Z., et al. “Calibrate before Use: Improving Few-Shot Performance of Language Models.” *ArXiv:2102.09690 [Cs]*, June 2021, arxiv.org/abs/2102.09690.