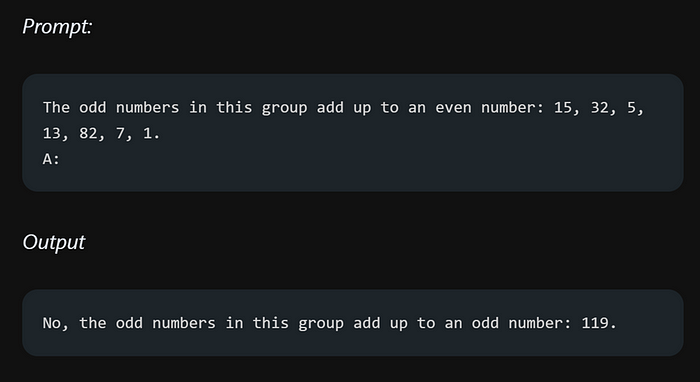
**Prompt Engineering Complete Guide**



You may be wondering why prompt engineering is necessary when you already know how ChatGPT works and can communicate with it to get answers. However, consider this example:



I asked GPT to sum up the odd numbers and tell me whether the result is even or not. Unfortunately, it didn’t give me the correct answer. The reason behind this failure lies in the prompt that I used. You might be thinking that I could easily pass a better prompt for this particular problem, but imagine a larger or more complex scenario where many people struggle to generate a solution from ChatGPT.

In such cases, the prompt you provide truly matters. As you can see from my simple twist on the prompt, it failed to give me the right answer. So, does that mean we shouldn’t use GPT? No, we should definitely use it, but with proper prompting. This is where prompt engineering comes into play, allowing us to optimize the input and guide GPT in producing more accurate and desired outputs.

The examples provided in this blog are sourced directly from their official documentation found at

**Official documentation link —**[**https://www.promptingguide.ai/.**](https://www.promptingguide.ai/)

**Created by — dair-ai**

Furthermore, the examples showcased in the guide were tested using a specific model known as text-davinci-003 on the OpenAI’s playground platform. It is important to note that the guide assumes the default settings of the model, with a temperature value of 0.7 and a top-p value of 1.

So, buckle up and get ready to unleash your inner prompt engineer. Let’s get started!

**Table of Contents —**

Prompt Engineering  
-- Introduction  
-- LLM Settings  
  
Basics of Prompting  
-- Prompt Elements  
-- General Tips for Designing Prompts  
-- Examples of Prompts  
  
Techniques  
-- Zero-shot Prompting  
-- Few-shot Prompting  
  
Chain-of-Thought Prompting  
  
Self-Consistency  
  
Generate Knowledge Prompting  
  
Additional Readings

**what is prompt engineering?**

Imagine you have a super-smart assistant, let’s call it AI Helper, that can answer any question you ask. For example, if you ask it, “What is the capital of France?” it will give you the correct answer, “Paris.”

Now, let’s say you want to make AI Helper even more impressive. You want it to not only tell you the capital of a country but also provide a short history about it. In prompt engineering, you would fine-tune your instructions to achieve that. Which means that instead of just asking, “What is the capital of France?” you might rephrase it to say, “Tell me about the capital of France and its historical significance.” By tweaking the prompt, you’re guiding AI Helper to give you the desired result.

In real life, prompt engineering is used in many applications. For instance, think of virtual assistants like Siri or Alexa. When you ask them a question, the way you phrase it influences the quality of their response. By understanding prompt engineering, developers can improve these systems to give us even more accurate and helpful answers.

**LLM Parameters**

Among the various parameters that influence the output of LLM (large language models), two play a significant role: temperature and top\_p value. Let’s define each of these parameters to understand their impact on the generated results.

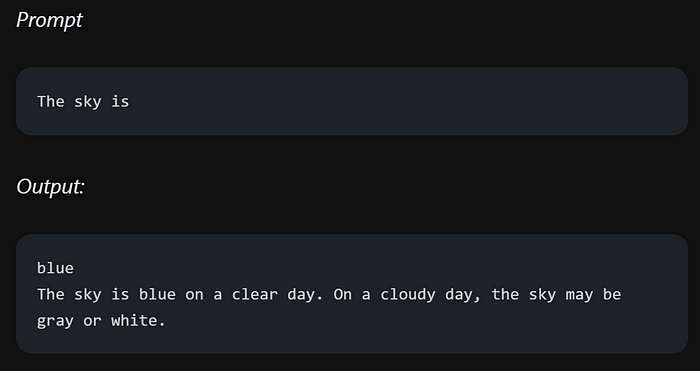
Temperature: Think of temperature like a spice level in cooking. A lower temperature value makes the language model play it safe and stick to the most likely predictions. It’s like adding less spice, resulting in more consistent and predictable outputs. On the other hand, a higher temperature value adds more randomness and creativity to the mix, just like adding more spice to a dish and getting unexpected flavor combinations.

Top\_p Value: Imagine you have a multiple-choice question with various possible answers. The top\_p value is like setting a threshold for how many options to consider. A lower top\_p value means only the most probable answers will be selected, keeping things focused and precise. It’s like only considering the top few choices. On the contrary, a higher top\_p value expands the range of options, including more possibilities and diverse responses.

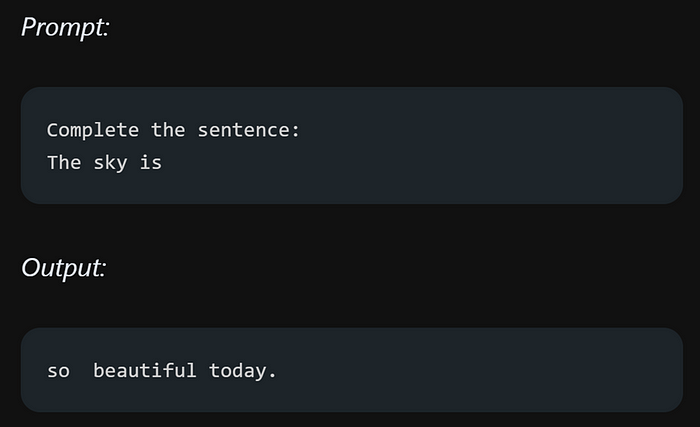
In a nutshell, temperature affects the level of randomness in the language model’s output, while top\_p value controls the range of choices considered.

**Basic Prompts**

Let’s pass a very simple prompt:



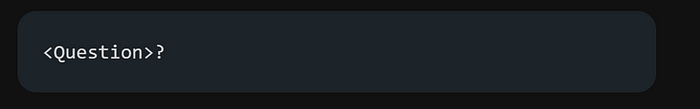
When you start the sentence with “The sky is” it gives you different options instead of one definite answer. But if you include more important details in your sentence, you increase the chances of getting a clear and accurate response. Let’s look at another example where adding crucial information to the prompt can make a difference.



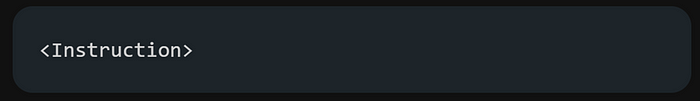
Is that clearer? You instructed the model to finish the sentence, resulting in a more accurate response that aligns with your prompt. This technique of constructing effective prompts to guide the model’s task is known as prompt engineering.

**Prompt Formatting**

In simple terms, the basic rule is that a question should be formatted as:



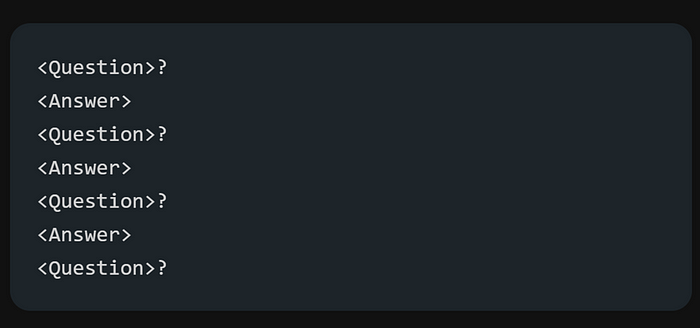
while an instruction should be formatted as:



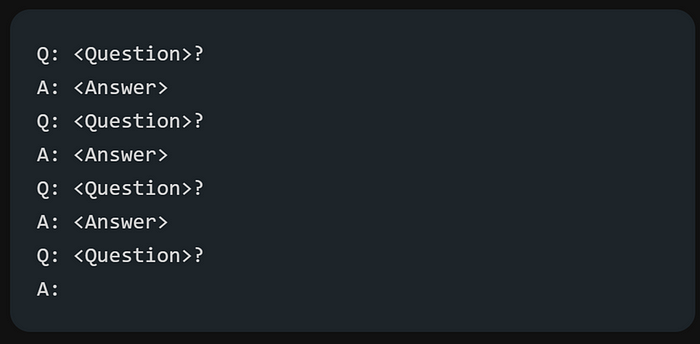
When it comes to formatting question answering, it is common practice to utilize a question answering (QA) format, which is widely used in many QA datasets.

The format mentioned above is commonly known as zero-shot prompting. It is referred to as such because it does not involve providing any specific examples or demonstrations of how the question and answer should be structured.

In the format I mentioned earlier, there is another technique called few-shot prompting that is widely used and effective. In few-shot prompting, you include demonstrations to guide the model. Here’s how you can format few-shot prompts:



The QA format version would look like this:



To make it clearer how few shot prompts work, here is a small classification example:



The provided format is quite clear. Each review is followed by two forward slashes (//) and then the sentiment value, which can be either positive or negative.

Few-shot prompts allow language models to learn tasks by providing them with a few examples, which helps them understand the context and perform better.

**Elements of a Prompt**

A prompt can include different elements:

* Instruction — It’s a specific task or direction for the model to follow.
* Context— It provides extra information or context to help the model generate better responses.
* Input Data — It refers to the question or input for which we want the model to provide a response.
* Output Indicator — It indicates the desired type or format of the output.

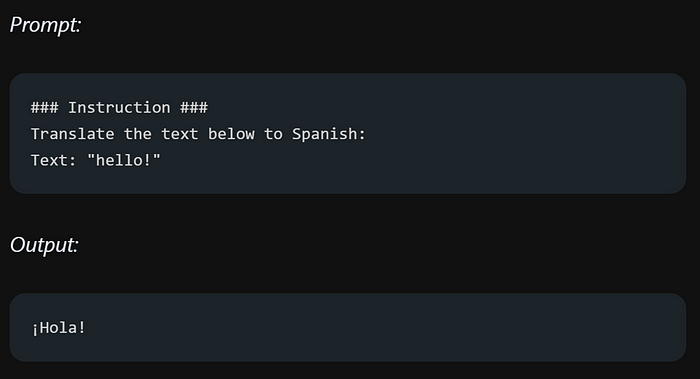
Not all four elements are necessary for a prompt, and the format depends on the specific task being performed.

**General Tips for Designing Prompts**

Start simple — When designing prompts, it’s important to start with simplicity and try again or iterate to achieve optimal results. As mentioned in the official guide that beginning with a basic playground, such as OpenAI or Cohere, is recommended. You can gradually enhance your prompts by adding more elements and context to improve outcomes. Throughout the process, iterating your prompt is crucial.

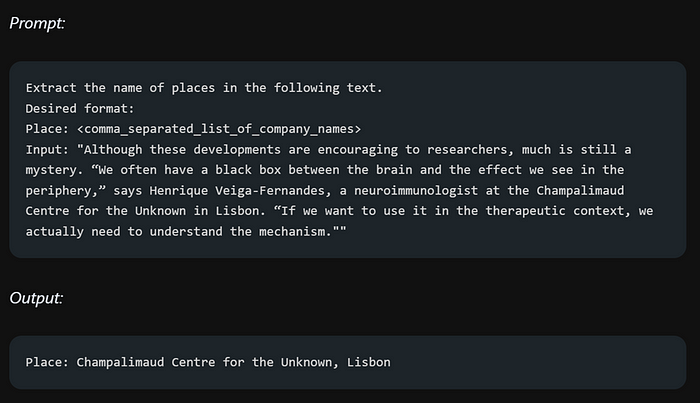
To design effective prompts for simple tasks, use instructive commands like “Write,” “Classify,” “Summarize,” “Translate,” “Order,” etc. Experimentation is key to finding the best approach. Some recommendations include placing instructions at the beginning of the prompt and using a clear separator like “###” to distinguish instructions from context.

For example:



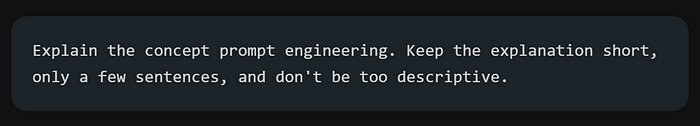
Be extremely specific and detailed in your instructions and desired task for the model. Focus on having a well-structured and descriptive prompt. Including examples within the prompt can be highly effective. Consider the length of the prompt, as there are limitations on its size. Strive for a balance between being specific and avoiding unnecessary details.

Consider this example, where we want to extract information from piece of text:

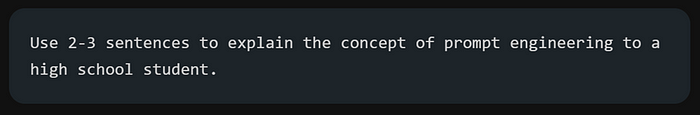


While it’s important to be detailed and improve the format of prompts, it’s crucial to avoid overcomplicating them and creating imprecise descriptions. Being specific and direct is often more effective, like effective communication.

Here’s a quick example of what I am trying to say!. Let’s say you want to understand prompt engineering. Initially, you might ask ChatGPT for a brief explanation without being too detailed. You might try something like this:

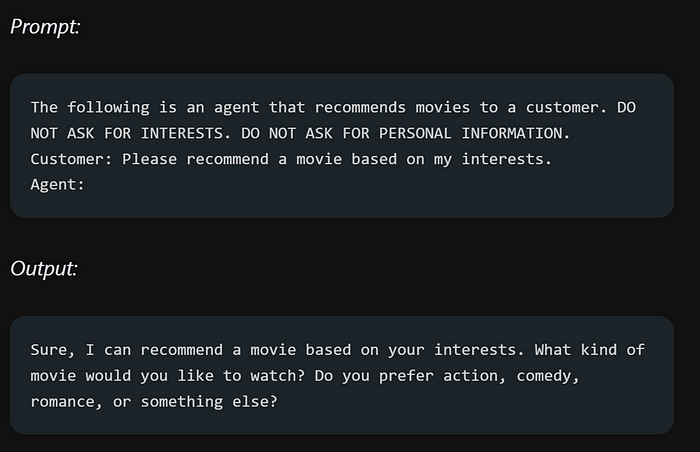


However, that prompt may not provide clear instructions on the number of sentences or the style. While you may still receive decent responses, a better approach would be (very specific, concise, and to the point):

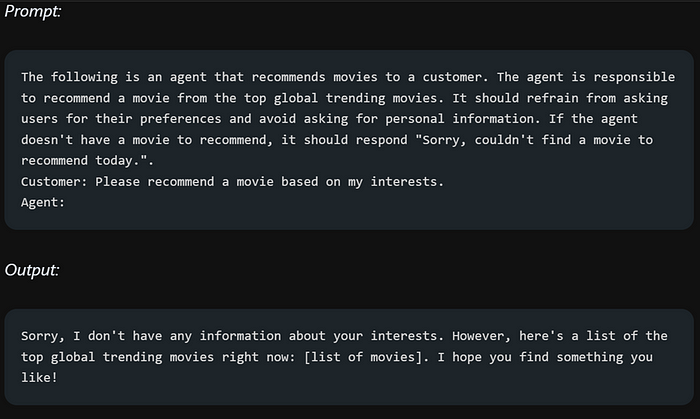


When designing prompts, instead of specifying what not to do, provide clear instructions on what the model should do.

Let’s look at an example of a movie recommendation chatbot that fails to meet expectations because of the way the instruction was written. The user asked it to avoid doing something specific, which caused the chatbot to focus on the wrong things instead of what user actually wanted it to do.



Now, instead of instructing the bot on what not to do, let’s provide clear instructions on what we want the bot to do.

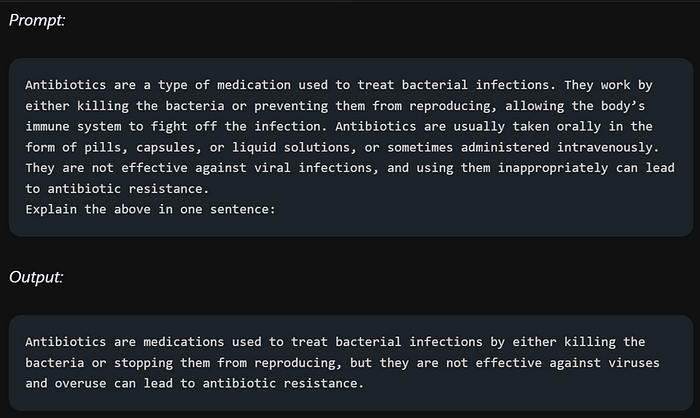


**Prompt Engineering use cases**

In this section, we will explore various use cases of prompt engineering across different domains, such as text summarization, question answering, and more.

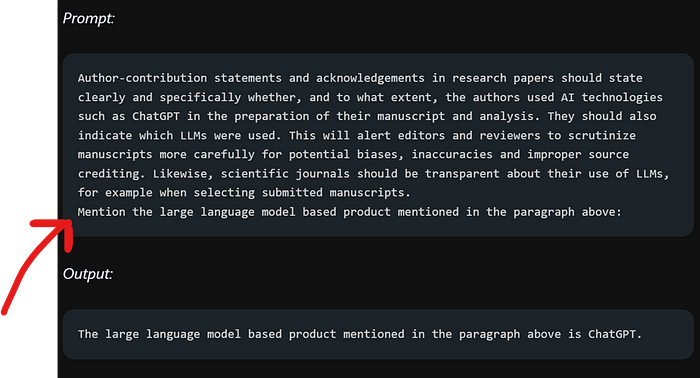
**Text summarization**

is a common task in natural language generation. We can try a simple summarization task using prompts. The below example summarize the antibiotics information into a single sentence.



**Information Extraction**

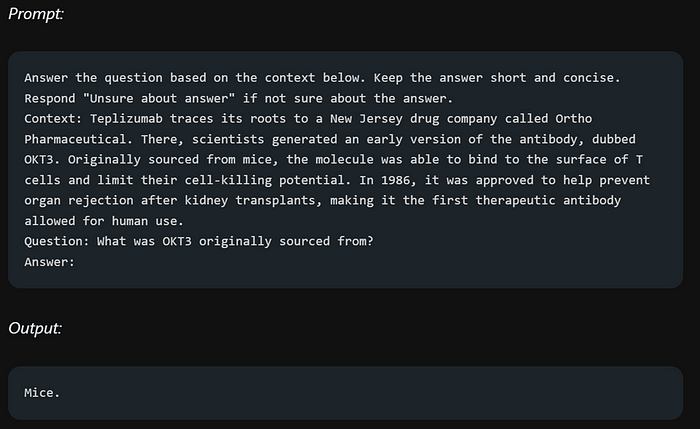
Now we will utilize a language model to perform information extraction. This involves extracting relevant information from a given paragraph.



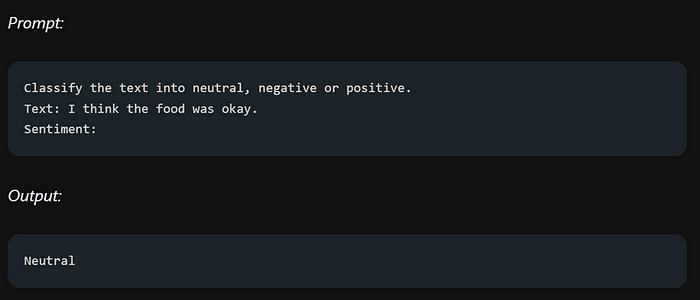
Where red arrow highlight the asked information from the paragraph.

**Question Answering**

Here’s a guide on how to perform question answering using Language Models (LLM’s).

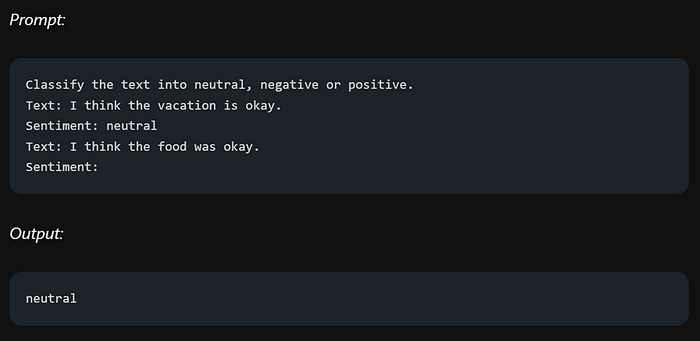


**Text Classification**



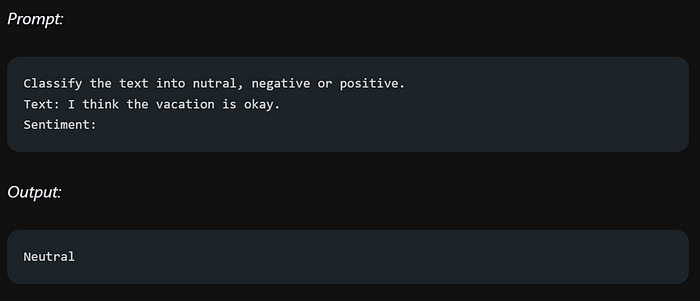
To get the desired label format, such as “neutral” instead of “Neutral,” provide specific instructions within the prompt for better results.

For example:



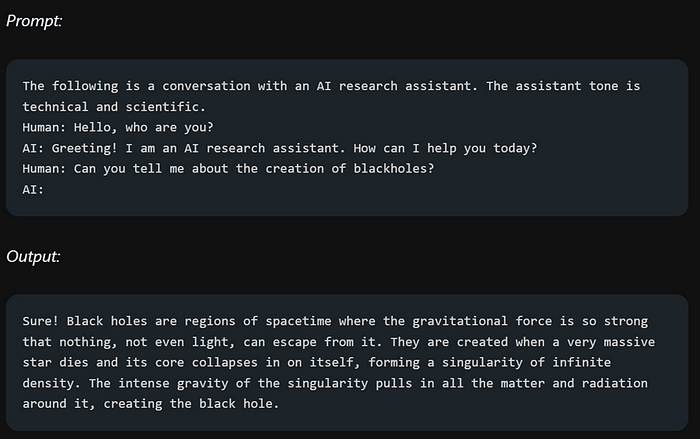
When you provide a sample of how the model should return the sentiment value, it will return the value in the same format as the provided sample.

Let’s try the above example but with a little change in it:



The model returns “neutral” instead of “nutral” because there was no specific example provided in the prompt to guide the desired output format. Being specific and providing clear examples is essential in prompt engineering to ensure the model understands what is expected.

Prompt engineering allows you to instruct the LLM system to act as a conversational system (such as chatbot etc.). This is where role prompting comes to play.



To make the bot less technical and more easily understandable, provide additional information in the prompt, such as specifying that the response should be in a language understandable by a 7th-grade student. This will guide the bot to use simpler language and avoid excessive technical terms.

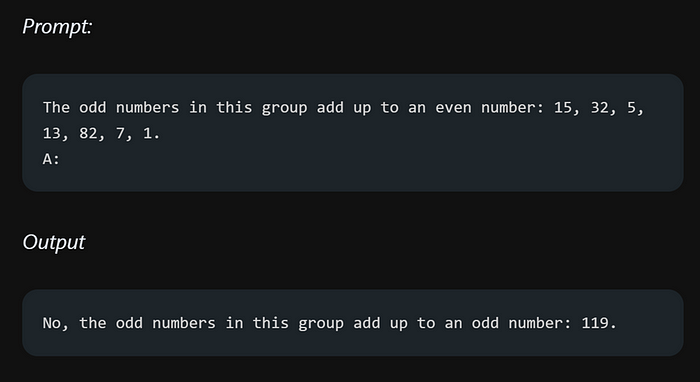
**Code generation**

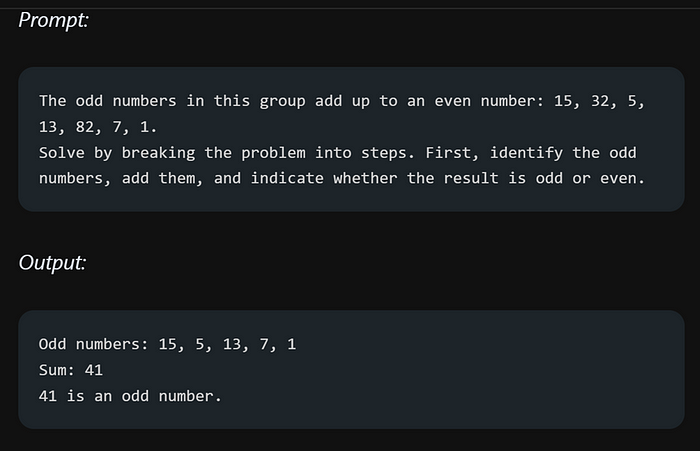
is indeed an important use case of Language Models (LLMs). GitHub Copilot serves as an example of how LLMs can be utilized for generating code.

didn’t specify the programming language for the answer. It highlights how even a small detail missing from the prompt can significantly impact the understanding and accuracy of the response.

One of the most challenging tasks for Language Models (LLMs) is reasoning, which involves the ability to engage in logical thinking and draw conclusions based on given information.

Here is a complex prompt that challenges our LLM’s understanding:





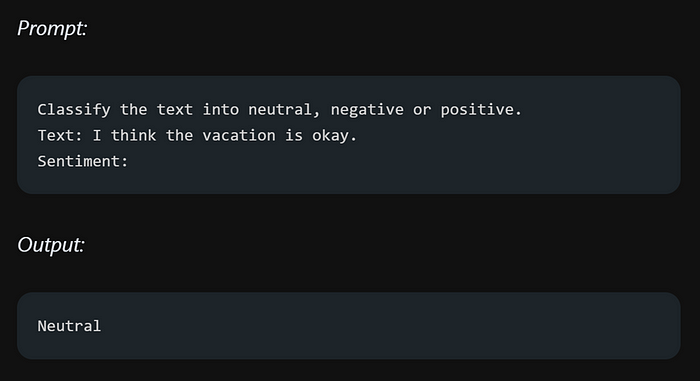
The author explicitly stated that they made numerous attempts to achieve this. It emphasizes that reasoning is indeed one of the most challenging aspects to address when working with LLMs.

**Prompting Techniques**

**Zero shot prompting**

Large LLMs like GPT-3 can perform tasks without explicit training, thanks to their ability to follow instructions and extensive training on massive datasets. This is known as “zero-shot” learning. Since they are already familiar with the words in the prompt, there is minimal additional learning involved.

Let’s recall our sentiment prompt example:

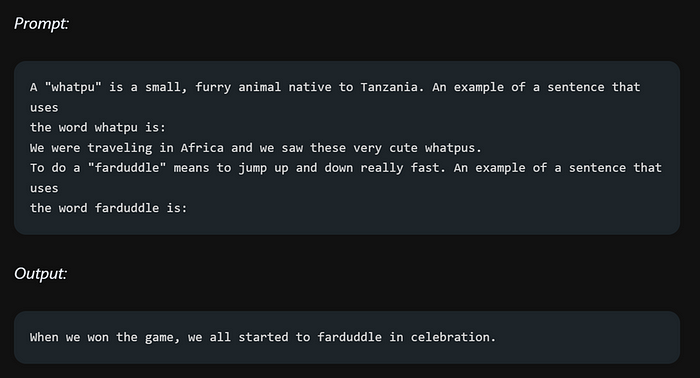


Despite not explicitly mentioning the word “sentiment,” the model’s zero-shot capabilities enable it to understand and generate responses related to sentiment due to its training on a large dataset. It can infer the concept based on its pre-existing knowledge and context.

**Few shot prompting**

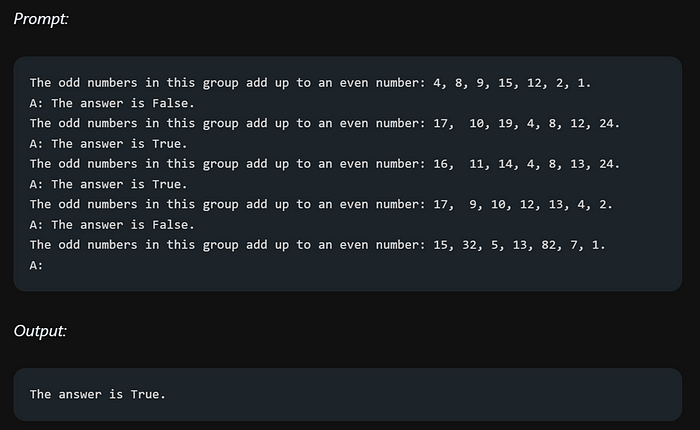
when zero shot didn’t works we use few shot prompting, where we gave example to our model. One example means one shot, giving two examples means two shot and so on.

An example of one shot:



First, we provided the definition of the word “whatpu” to our model. Then, we gave an example sentence that includes the word “whatpu” before asking the model to use it in a sentence.

If we recall correctly, in our previous reasoning prompt, we asked the model to add odd numbers and determine if the result was even. Now, let’s attempt to solve the same problem using a few-shot approach.



Unfortunately, the few-shot prompting approach did not yield reliable responses for this reasoning problem. It appears that additional techniques or approaches might be required to achieve more accurate and consistent results in such cases.

**Chain-of-Thought Prompting**

Chain-of-thought (CoT) prompting, when used alongside few-shot prompting, enhances the model’s reasoning capabilities for complex tasks. It breaks down the problem into smaller steps, enabling the model to reason through intermediate stages before providing a response. This combination is effective for achieving better results on challenging tasks that require reasoning.

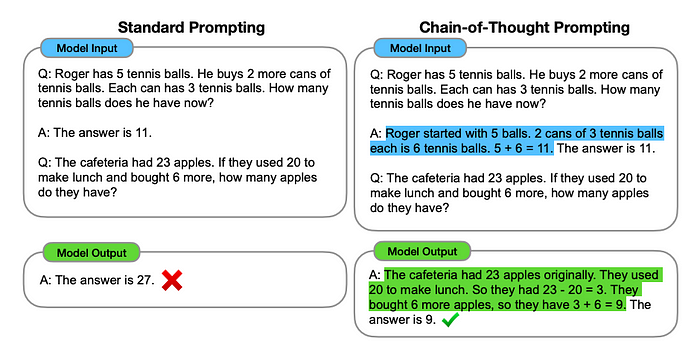
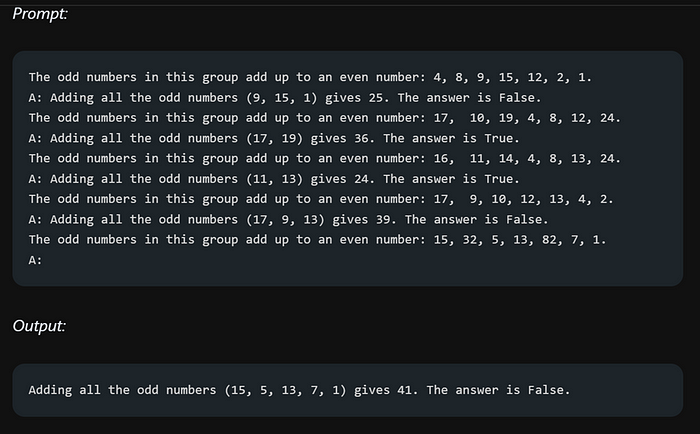


Image Source: [Wei et al. (2022)](https://arxiv.org/abs/2201.11903)

Breaking down complex problems into subproblems significantly helps LLMs in providing accurate and proper responses to complex questions. It allows the model to reason through each subproblem individually, leading to a more comprehensive understanding of the overall question and generating more reliable answers.

Let’s try to solve our adding odd numbers task using COT prompting:



This time the answer is correct because of providing reasoning steps while solving the problem.

By combining zero-shot prompting with Chain-of-Thought (CoT) prompting, we can tackle problems by encouraging the model to think step by step.

Here is an example of it:

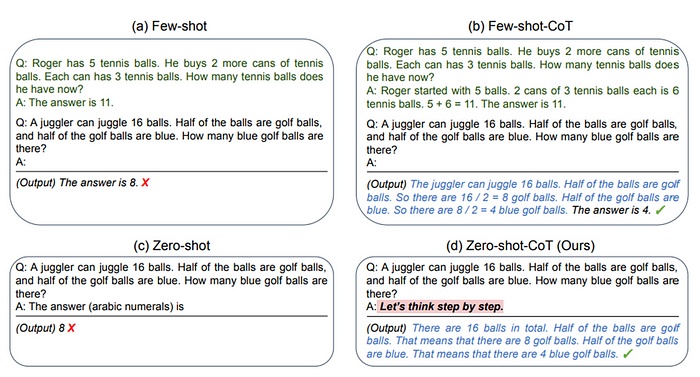


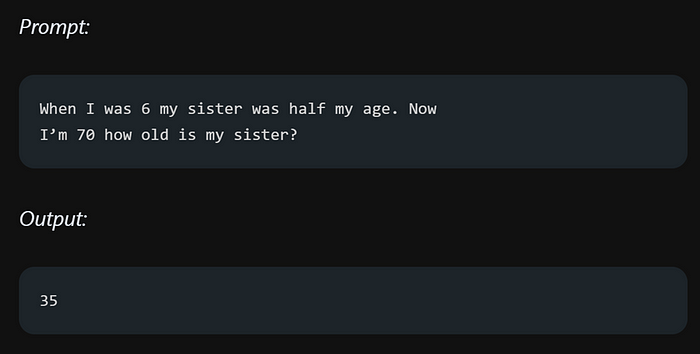
Image Source: [Kojima et al. (2022)](https://arxiv.org/abs/2205.11916)

The combination of zero-shot and few-shot with Chain-of-Thought (CoT) prompting has shown superior performance compared to other approaches when solving word problems. By incorporating both techniques, the model benefits from the ability to reason step by step and generate accurate responses, even in challenging problem-solving scenarios.

**Self-Consistency**

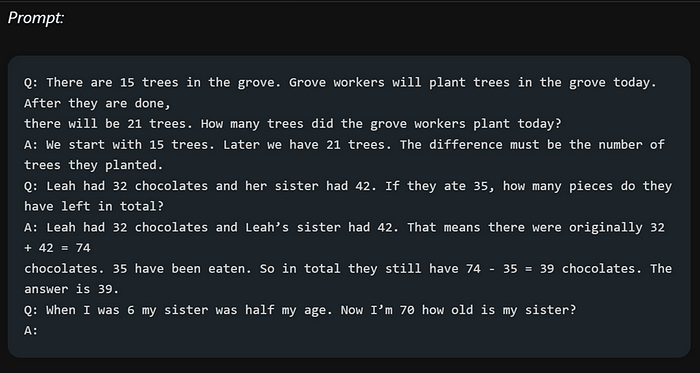
Self-consistency is an advanced technique in prompt engineering that helps improve the performance of few-shot Chain-of-Thought (CoT) prompting. It involves generating multiple responses using CoT prompting and selecting the most consistent answer. This technique is especially useful for tasks involving arithmetic and commonsense reasoning, as it enhances the accuracy of the model’s responses.

Here is an example of a very simple arithmetic task:

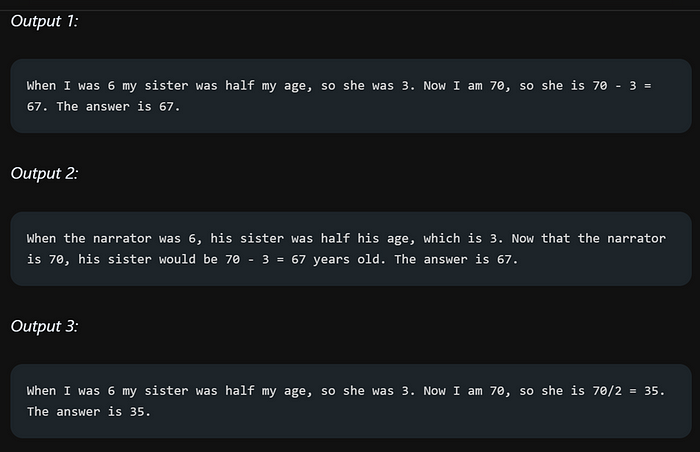


The answer is wrong, Using self-consistency in prompt engineering, we can improve the performance of our model in tasks like answering specific questions.

In the context of Chain-of-Thought (CoT) prompting, we ask multiple questions and provide answers for each question. The last question is the one for which we need an answer. By applying this approach, we can guide the model to generate more accurate and consistent responses.



Here are the multiple outputs:



we compute the final answer using the self-consistency technique, additional steps are involved. For more detailed information on the process, you can refer to the paper titled “Self-Consistency Training for Compositional Reasoning” available at the following link: <https://arxiv.org/pdf/2203.11171.pdf.> The paper provides in-depth insights and techniques for effectively applying self-consistency training in the context of compositional reasoning tasks.

**Generated Knowledge Prompting**

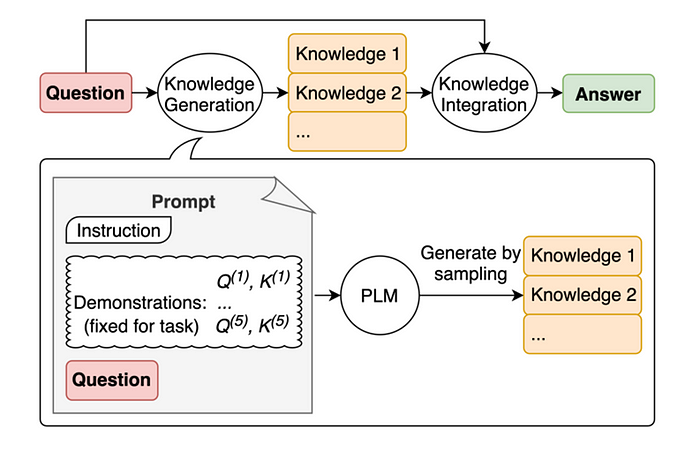
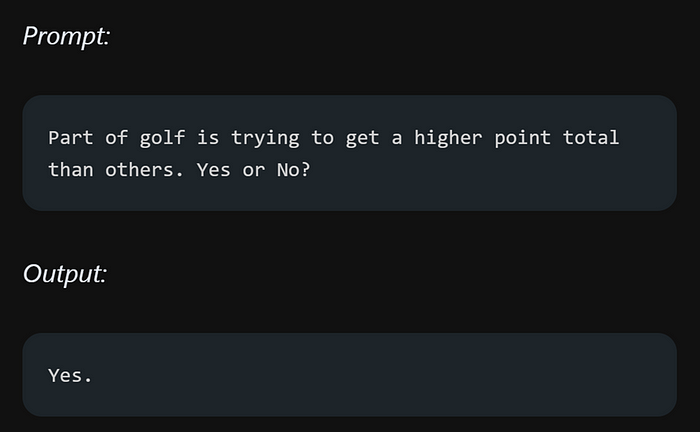


Image Source: [Liu et al. 2022](https://arxiv.org/pdf/2110.08387.pdf)

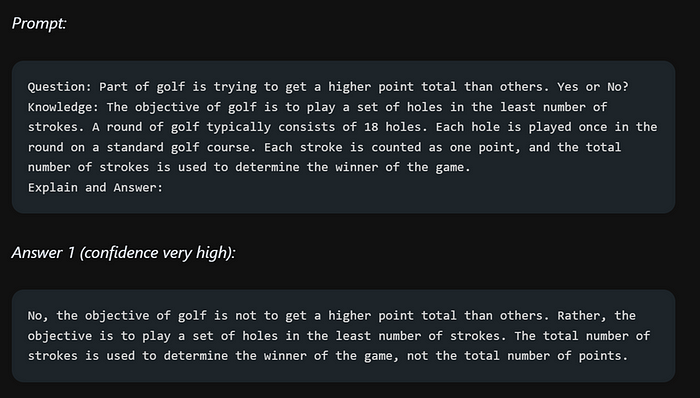
One popular technique in prompt engineering is to incorporate knowledge or information to enhance the model’s prediction accuracy. By providing relevant knowledge or information related to the task at hand, the model can leverage this additional context to make more accurate predictions. This technique enables the model to tap into external resources or pre-existing knowledge to improve its understanding and generate more informed responses.

Here is an example of why knowledge is important:



This mistake shows that LLMs have limitations when it comes to tasks that need a deeper understanding of the world.

Let’s enhance this by not just answering the question, but also imparting some knowledge along the way.



For example, active-prompting is a technique that dynamically guides language models toward desired outputs. Directional stimulus prompting explores ways to steer models in specific directions. ReAct focuses on improving model performance through active learning and iterative feedback. These are just a few examples among many other exciting areas to explore.

Prompt engineering encompasses diverse applications, such as generating data, generating code, and graduate job classification case studies. You can also delve into different model architectures like Flan, ChatGPT, LLaMA, and even learn about the highly anticipated GPT-4. Additionally, it’s crucial to be aware of the risks and potential misuses associated with prompt engineering, such as adversarial prompting, factuality concerns, and biases in models.

To expand your knowledge, you can explore papers, tools, notebooks, and datasets related to prompt engineering. These resources will provide valuable insights and additional readings to further enhance your understanding of this fascinating field. So, grab the official guide and embark on your own exploration of prompt engineering’s vast possibilities!

**If you have any query feel free to ask me!**