

Examples of Prompts

The previous section introduced a basic example of how to prompt LLMs.

This section will provide more examples of how to use prompts to achieve different tasks and introduce key concepts along the way. Often, the best way to learn concepts is by going through examples. The few examples below illustrate how you can use well-crafted prompts to perform different types of tasks.

Topics:

- [Text Summarization](#)
 - [Information Extraction](#)
 - [Question Answering](#)
 - [Text Classification](#)
 - [Conversation](#)
 - [Code Generation](#)
 - [Reasoning](#)
-

Basic Prompt Examples for LLMs





Learn more about prompting techniques and examples in our new AI courses.

[Join now!](#)

Use code BLACKFRIDAY to get an extra 35% off. This offer ends on 29th November 2024.

Text Summarization

One of the standard tasks in natural language generation is text summarization. Text summarization can include many different flavors and domains. In fact, one of the most promising applications of language models is the ability to summarize articles and concepts into quick and easy-to-read summaries. Let's try a basic summarization task using prompts.

Let's say you are interested to learn about antibiotics, you could try a prompt like this:

Prompt:

Explain antibiotics

A:

Output:

Antibiotics are a type of medication used to treat bacterial infections. They work by either killing the bacteria or preventing them from reproducing, allowing the body's immune system to fight off the infection. Antibiotics are usually taken orally in the form of pills, capsules, or liquid solutions, or sometimes administered intravenously. They are not effective against viral infections, and using them inappropriately can lead to antibiotic resistance.

The "A:" is an explicit prompt format that you use in question answering. You used it here to tell the model that there is an answer expected further. In this example, it's not clear how this is useful vs not using it but we will leave it that for later examples. Let's just assume that this is too much information and you want to summarize it further. In fact, you can instruct the model to summarize into one sentence like so:

Prompt:

Antibiotics are a type of medication used to treat bacterial infections. They work by either killing the bacteria or preventing them from reproducing, allowing the body's immune system to fight off the infection. Antibiotics are usually taken orally in the form of pills, capsules, or liquid solutions, or sometimes administered intravenously. They are not effective against viral infections, and using them inappropriately can lead to antibiotic resistance.

Explain the above in one sentence:

Output:

Antibiotics are medications used to treat bacterial infections by either killing the bacteria or stopping them from reproducing, but they are not effective against viruses and overuse can lead to antibiotic resistance.

Without paying too much attention to the accuracy of the output above, which is something we will touch on in a later guide, the model tried to summarize the paragraph in one sentence. You can get clever with the instructions but we will leave that for a later chapter. Feel free to pause here and experiment to see if you get better results.

Information Extraction

While language models are trained to perform natural language generation and related tasks, it's also very capable of performing classification and a range of other natural language processing (NLP) tasks.

Here is an example of a prompt that extracts information from a given paragraph.

Prompt:

Author-contribution statements and acknowledgements in research papers should state clearly and specifically whether, and to what extent, the authors used AI technologies such as ChatGPT in the preparation of their manuscript and analysis. They should also indicate which LLMs were used. This will alert editors and reviewers to scrutinize manuscripts more carefully for potential biases, inaccuracies and improper source crediting. Likewise, scientific journals should be transparent about their use of LLMs, for example when selecting submitted manuscripts.

Mention the large language model based product mentioned in the paragraph above:

Output:

The large language model based product mentioned in the paragraph above is ChatGPT.

There are many ways you can improve the results above, but this is already very useful.

By now it should be obvious that you can ask the model to perform different tasks by simply instructing it what to do. That's a powerful capability that AI product developers are already using to build powerful products and experiences.

Paragraph source: [ChatGPT: five priorities for research](#)

Question Answering

One of the best ways to get the model to respond with specific answers is to improve the format of the prompt. As covered before, a prompt could combine instructions, context, input, and output indicators to get improved results. While these components are not required, it becomes a good practice as the more specific you are with instruction, the better results you will get. Below is an example of how this would look following a more structured prompt.

Prompt:

Answer the question based on the context below. Keep the answer short and concise. Respond "Unsure about answer" if not sure about the answer.

Context: Teplizumab traces its roots to a New Jersey drug company called Ortho Pharmaceutical. There, scientists generated an early version of the antibody, dubbed OKT3. Originally sourced from mice, the molecule was able to bind to the surface of T cells and limit their cell-killing potential. In 1986, it was approved to help prevent organ rejection after kidney transplants, making it the first therapeutic antibody allowed for human use.

Question: What was OKT3 originally sourced from?

Answer:

Output:

Mice.

Context obtained from [Nature](#).

Text Classification

So far, you have used simple instructions to perform a task. As a prompt engineer, you need to get better at providing better instructions. But that's not all! You will also find that for harder use cases, just providing instructions won't be enough. This is where you need to think more about the context and the different elements you can use in a prompt. Other elements you can provide are `input data` or `examples`.

Let's try to demonstrate this by providing an example of text classification.

Prompt:

```
Classify the text into neutral, negative or positive.  
Text: I think the food was okay.  
Sentiment:
```

Output:

```
Neutral
```

You gave the instruction to classify the text and the model responded with `'Neutral'`, which is correct. Nothing is wrong with this but let's say that what you really need is for the model to give the label in the exact format you want. So instead of `Neutral`, you want it to return `neutral`. How do you achieve this? There are different ways to do this. You care about specificity here, so the more information you can provide the prompt, the better results. You can try providing examples to specify the correct behavior. Let's try again:

Prompt:

```
Classify the text into neutral, negative or positive.  
Text: I think the vacation is okay.  
Sentiment: neutral  
Text: I think the food was okay.  
Sentiment:
```

Output:

neutral

Perfect! This time the model returned `neutral` which is the specific label you were looking for. It seems that the example provided in the prompt helped the model to be specific in its output.

To highlight why sometimes being specific is important, check out the example below and spot the problem:

Prompt:

```
Classify the text into nutral, negative or positive.  
Text: I think the vacation is okay.  
Sentiment:
```

Output:

Neutral

What is the problem here? As a hint, the made-up `nutral` label is completely ignored by the model. Instead, the model outputs `Neutral` as it has some bias towards that label. But let's assume that what you really want is `neutral`. How would you fix this? Maybe you can try adding descriptions to the labels or add more examples to the prompt? If you are not sure, we will discuss a few ideas in the upcoming sections.

Conversation

Perhaps one of the more interesting things you can achieve with prompt engineering is instructing the LLM system on how to behave, its intent, and its identity. This is particularly useful when you are building conversational systems like customer service chatbots.

For instance, let's create a conversational system that's able to generate more technical and scientific responses to questions. Note how you are explicitly telling it how to behave through the instruction. This is sometimes referred to as *role prompting*.

Prompt:

The following is a conversation with an AI research assistant. The assistant tone is technical and scientific.

Human: Hello, who are you?

AI: Greeting! I am an AI research assistant. How can I help you today?

Human: Can you tell me about the creation of blackholes?

AI:

Output:

Sure! Black holes are regions of spacetime where the gravitational force is so strong that nothing, not even light, can escape from it. They are created when a very massive star dies and its core collapses in on itself, forming a singularity of infinite density. The intense gravity of the singularity pulls in all the matter and radiation around it, creating the black hole.

Our AI research assistant sounds a bit too technical, right? Okay, let's change this behavior and instruct the system to give more accessible answers.

Prompt:

The following is a conversation with an AI research assistant. The assistant answers should be easy to understand even by primary school students.

Human: Hello, who are you?

AI: Greeting! I am an AI research assistant. How can I help you today?

Human: Can you tell me about the creation of black holes?

AI:

Output:

Sure! Black holes are created when a very massive star runs out of fuel and collapses in on itself. The gravity of the star is so strong that nothing, not even light, can escape its pull. This creates a point in space-time with an intense gravitational force called a black hole.

You made some progress. You can continue improving it. If you add more examples, you might get even better results.

Code Generation

One application where LLMs are quite effective is code generation. Copilot is a great example of this. There are a vast number of code-generation tasks you can perform with clever prompts. Let's look at a few examples below.

First, let's try a simple program that greets the user.

Prompt:

```
/*  
Ask the user for their name and say "Hello"  
*/
```

Output:

```
let name = prompt("What is your name?");  
console.log(`Hello, ${name}!`);
```

Notice that you didn't even need to specify the language to use.

Let's switch levels a bit. The example below shows how powerful LLMs can be with a little more effort in designing the prompts.

Prompt:

```
""  
Table departments, columns = [DepartmentId, DepartmentName]  
Table students, columns = [DepartmentId, StudentId, StudentName]  
Create a MySQL query for all students in the Computer Science Department  
""
```

Output:

```
SELECT StudentId, StudentName  
FROM students  
WHERE DepartmentId IN (SELECT DepartmentId FROM departments WHERE DepartmentName = 'Computer  
Science');
```

This is very impressive. In this case, you provided data about the database schema and asked it to generate a valid MySQL query.

Reasoning

Perhaps one of the most difficult tasks for an LLM today is one that requires some form of reasoning. Reasoning is one of most interesting areas due to the types of complex applications that can emerge from LLMs.

There have been some improvements in tasks involving mathematical capabilities. That said, it's important to note that current LLMs struggle to perform reasoning tasks so this requires even more advanced prompt engineering techniques. We will cover these advanced techniques in the next guide. For now, we will cover a few basic examples to show arithmetic capabilities.

Prompt:

What is $9,000 * 9,000$?

Output:

81,000,000

Let's try something more difficult.

Prompt:

The odd numbers in this group add up to an even number: 15, 32, 5, 13, 82, 7, 1.
A:

Output

No, the odd numbers in this group add up to an odd number: 119.

That's incorrect! Let's try to improve this by improving the prompt.

Prompt:

The odd numbers in this group add up to an even number: 15, 32, 5, 13, 82, 7, 1.
Solve by breaking the problem into steps. First, identify the odd numbers, add them, and indicate whether the result is odd or even.

Output:

Odd numbers: 15, 5, 13, 7, 1
Sum: 41
41 is an odd number.

Much better, right? By the way, we tried this task a couple of times and the model sometimes fails. If you provide better instructions combined with examples, it might help get more accurate results.

In the upcoming section, we will cover even more advanced prompt engineering concepts and techniques for improving performance on all these and more difficult tasks.

Notebook

If you want to practice with the prompts above using Python, we have prepared a notebook to test some of the prompts using the OpenAI models.

 **Getting Started with Prompt Engineering**

Last updated on November 18, 2024