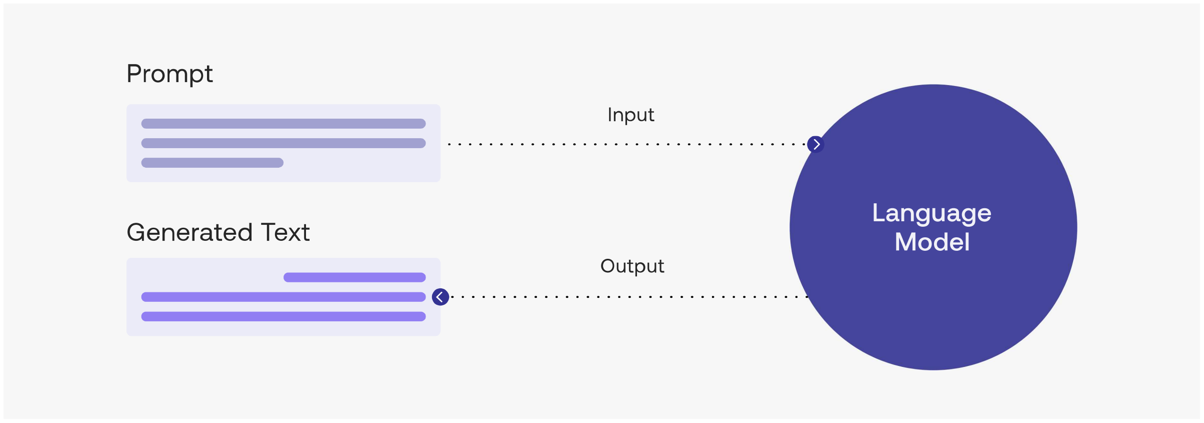
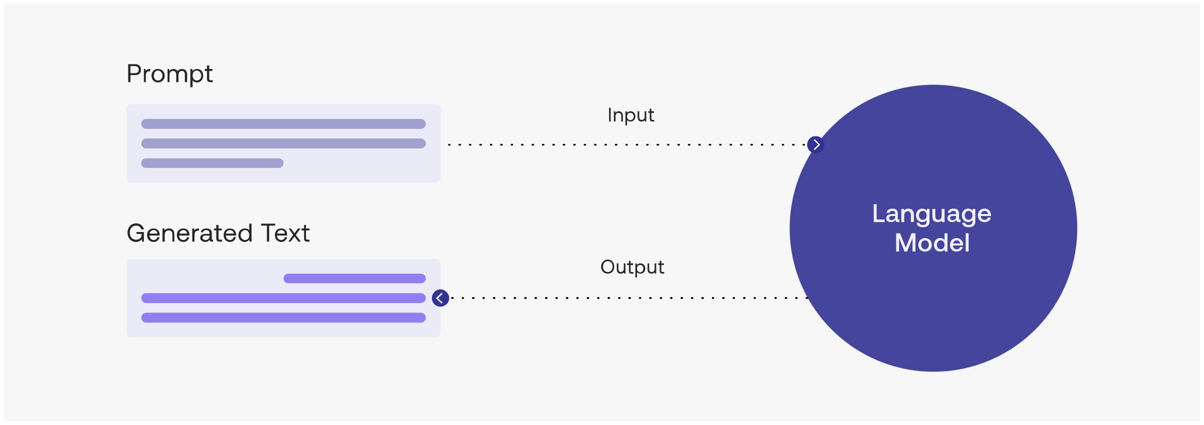
Prompt Engineering

Here, we discuss a few principles and techniques for writing prompts (inputs for our models) that will help you get the best generations for your task. Choosing the right temperature can also have a big influence on generation quality. We discuss temperature separately here.



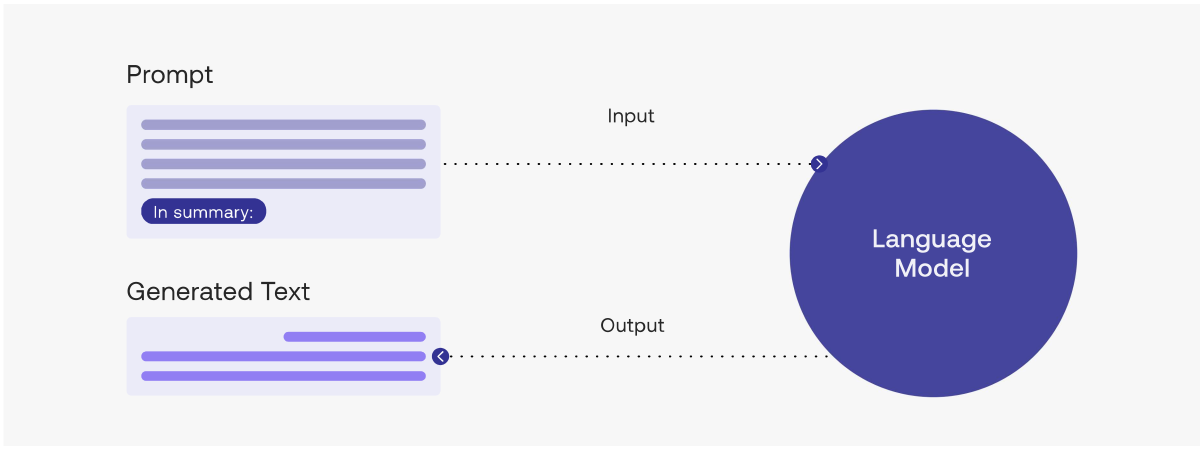


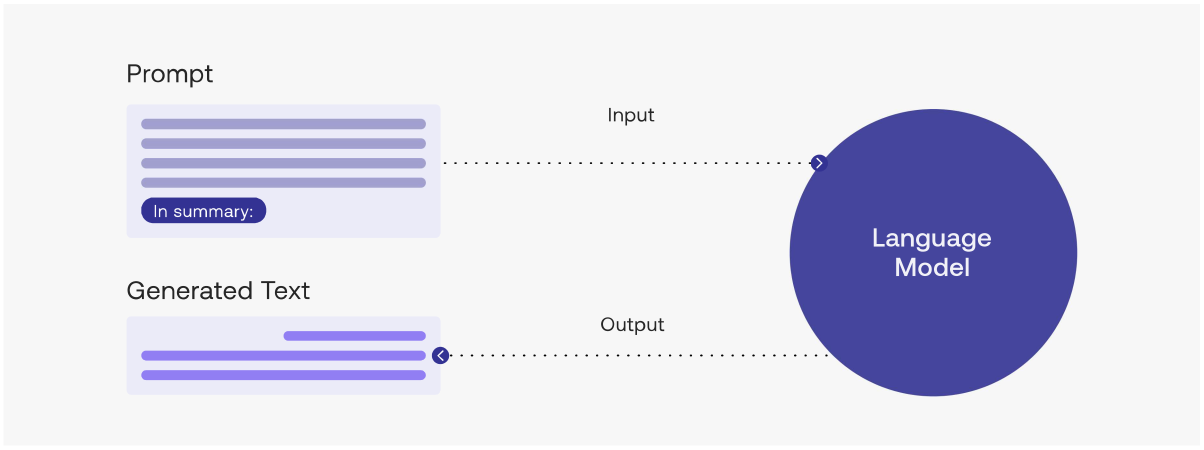
Main Principles

We find that there are two main ideas to keep in mind while designing prompts for our models.

1. A prompt guides the model to generate useful output

If you need a summary of an article, for example, a large language model trained on enough data can generate a summary if you guide it as such:





This prompt has two components: the text you want summarized, and the task description.

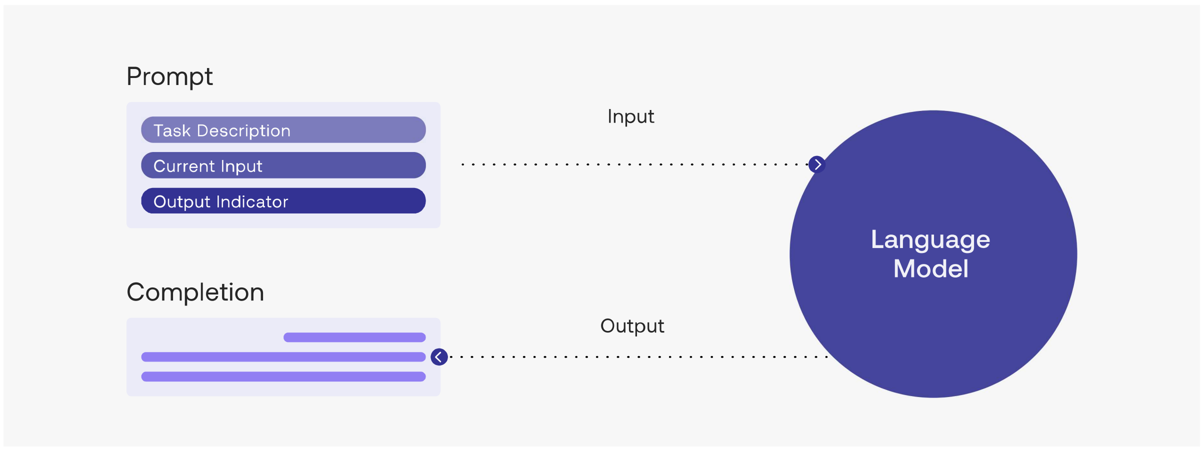
2. Try multiple formulations of your prompt to get the best generations

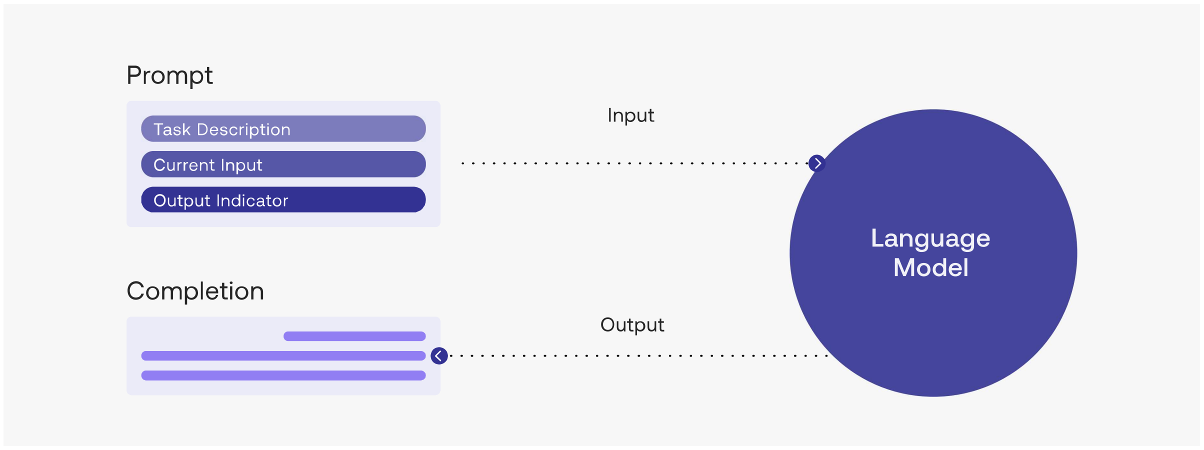
When using generate, it is useful to try a range of different prompts for the problem you are trying to solve. Different formulations of the same prompt which might sound similar to humans can lead to generations that are quite different from each other. This might happen, for instance, because our models have learned that the different formulations are actually used in very different contexts and for different purposes. Below we give a number of examples that we've found to work particularly well for different tasks.

In the summarization example, if “In summary,” doesn’t lead to a good generation, we may want to try “To summarize in plain language,“ or “The main point to take from this article is that”.

Additionally, you can also use the likelihood feature in the playground to see if there are particular words, phrases, or structures that the model has trouble understanding. However, keep in mind that the average likelihood of tokens will always be high at the beginning of the sequence. The model might assign low likelihood to the first time you introduce a novel concept or name, but once it has seen it once it can readily use it in the generation. You can also use the likelihood capability to see if there is any spelling or punctuation that is creating issues for tokenization.

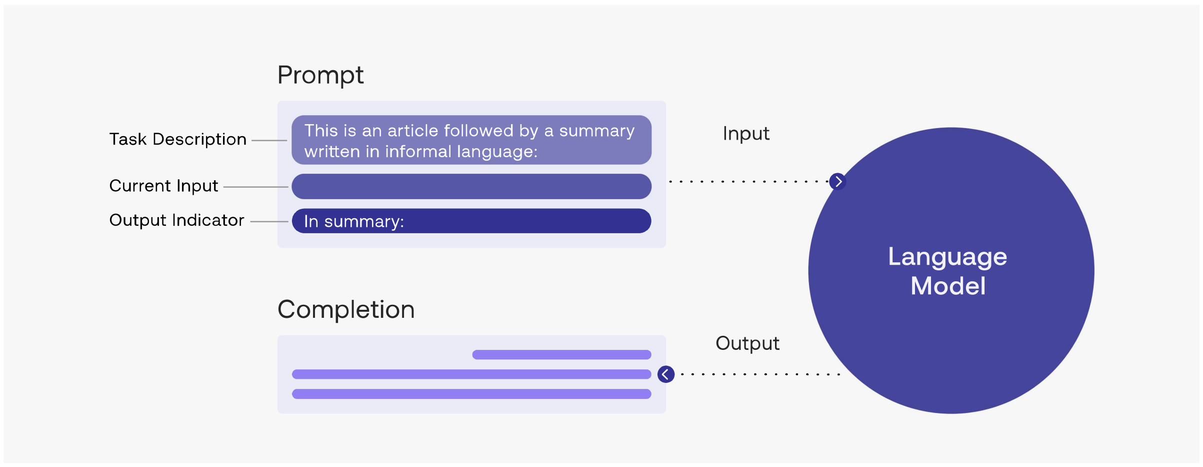
3. Describe the task and the general setting

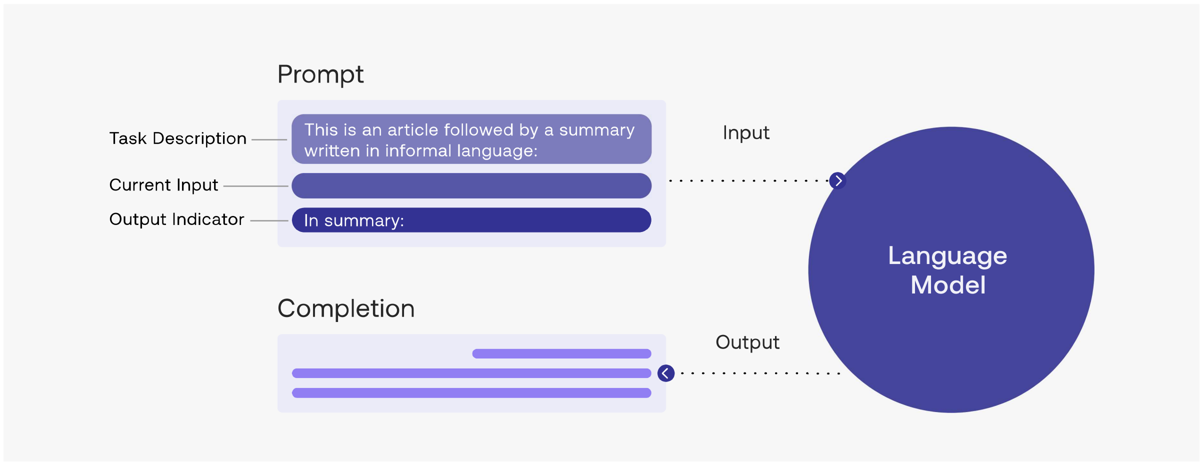




It's often useful to include additional components of the task description, naturally these tend to come after the input text we're trying to process.

Provide the model with enough context. For example, we can describe the summarization task in more detail before the article.





Example: shaping the task we need the model to do in natural language can use text both before and after the input text we aim to process.

Let's consider a few more aspects of this by looking at a different example. Suppose that you would like to use our models to assist your customer satisfaction department by automatically generating plausible responses to customer requests (note: the generations are not to be sent to the customers, this is only a simulation).

A customer contacts your company with the following question:
`Hi, I'd like a refund for the coffee maker I ordered. Would that be`

How do we design a prompt around this to get useful generations for the agent interacting with the customer? Let's begin with telling our model what the general setting is and what the remainder of the prompt is going to contain:

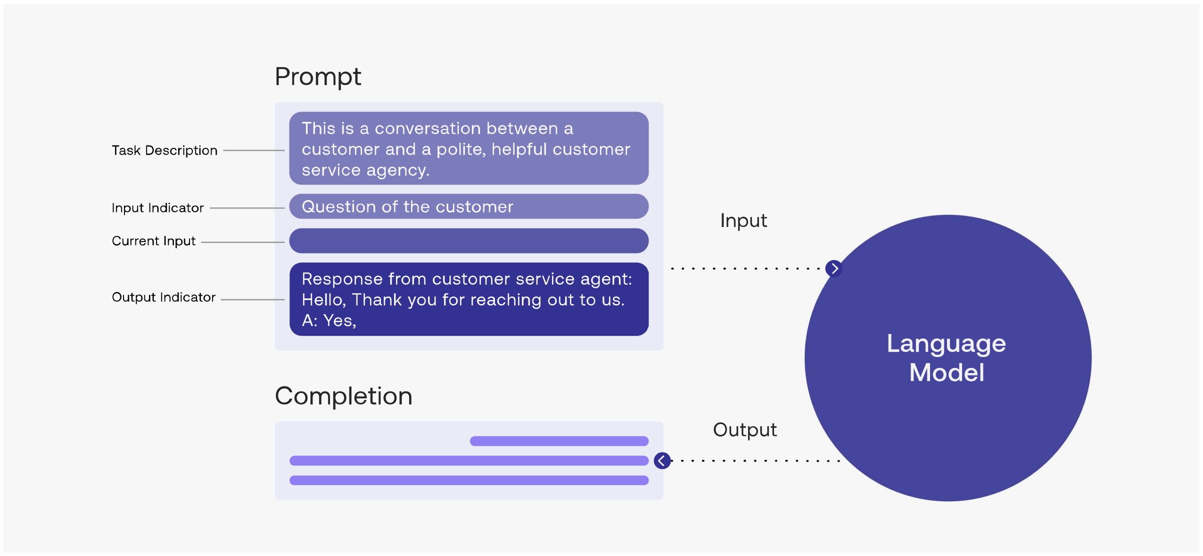
```
This is a conversation between a customer and a polite, helpful cust
Question of the customer: Hi, I'd like a refund for the coffee maker
```

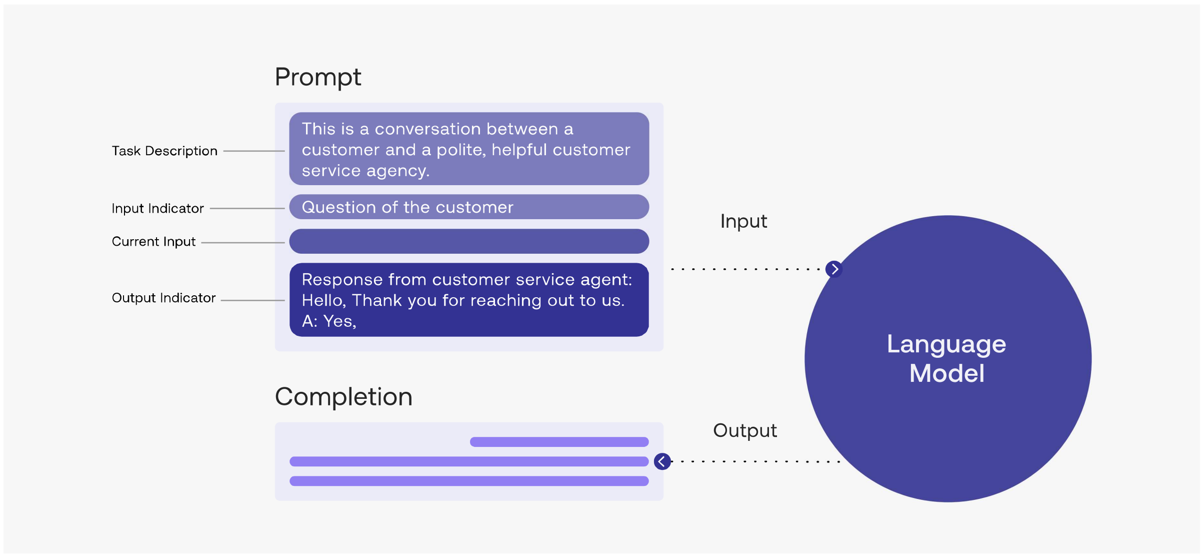
Great, we've told our model about what to expect and have made it clear that our query is a question of the customer. Next, let's show the model the beginning of the response we would like to give the customer.

```
Response by the customer service agent: Hello, thank you for reachin
```

Note how we've stated clearly that the next sentence is a response to the question, that it comes from a customer service agent, and that we want to give a positive answer. Putting this all together, we obtain the following prompt:

```
This is a conversation between a customer and a polite, helpful cust
Question of the customer: Hi, I'd like a refund for the coffee maker
Response by the customer service agent: Hello, thank you for reachin
```


Prompt
Task Description
This is a conversation between a customer and a polite, helpful customer service agency.
Input Indicator
Question of the customer
Current Input
Output Indicator
Response from customer service agent: Hello, Thank you for reaching out to us. A: Yes,
Input
Language Model
Completion
Output


Prompt
Task Description
This is a conversation between a customer and a polite, helpful customer service agency.
Input Indicator
Question of the customer
Current Input
Output Indicator
Response from customer service agent: Hello, Thank you for reaching out to us. A: Yes,
Input
Language Model
Completion
Output

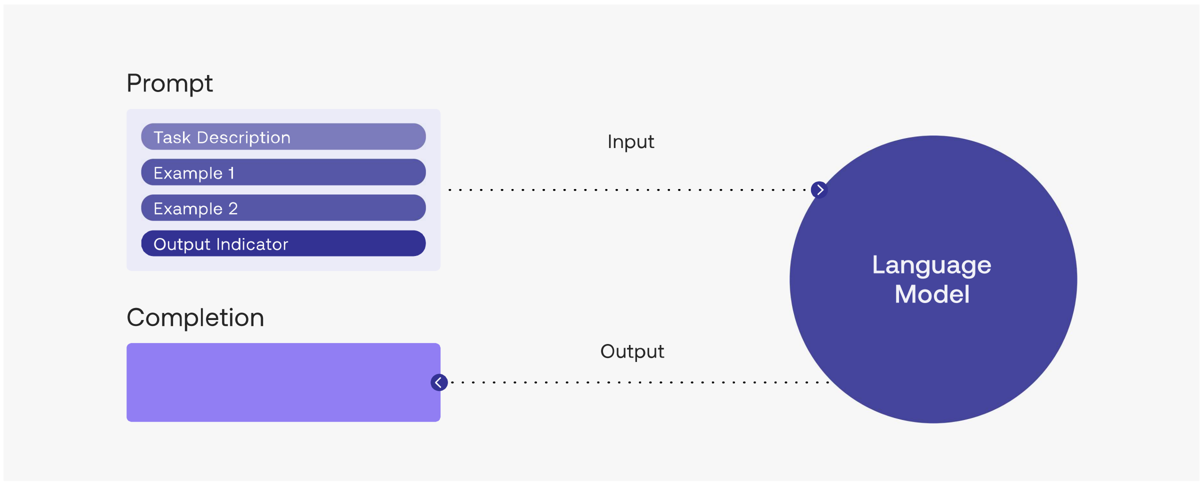
Certain components of prompts (like input and output indicators) are useful in describing a desired task to the model, especially when including multiple examples in the prompt (as the next figures will show).

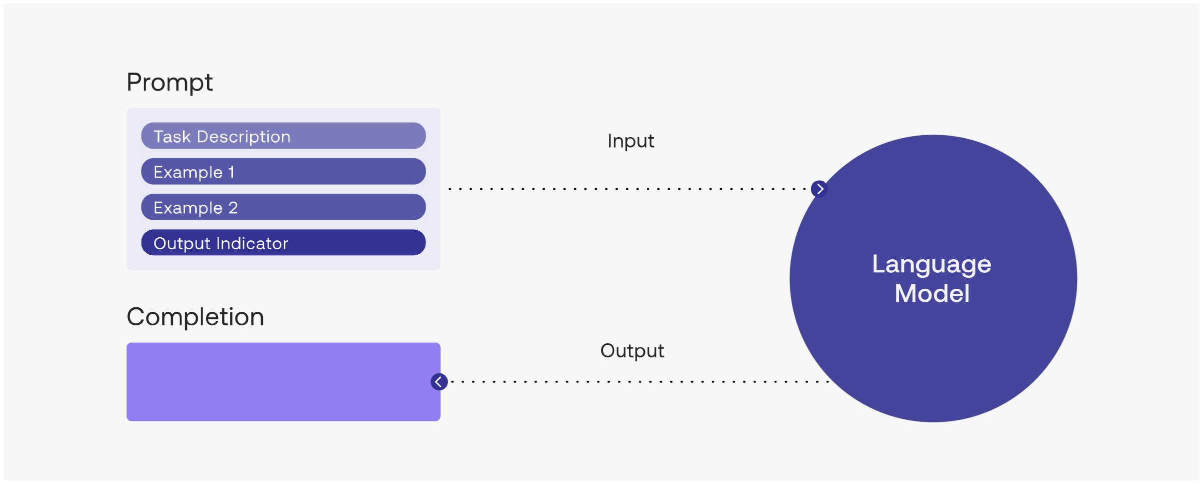
Feeding this into our Medium model multiple times we get the following completions:

- Yes, we are able to accept returns if the product is unused and unopened.
- Yes, we are happy to refund your purchase. However, we do require you to return the item to our store for a full refund.
- Yes, we can do that. Please send us a message with your name, phone number, and the reason for the refund. We will get back to you as soon as possible.

Note that even though this is a simplified example we get plausible completions from the baseline model using only a small number of customer service interactions! This could be further improved by finetuning it on examples of how you would like the model to handle specific questions and requests.

4. Show the model what you would like to see





Adding examples to a prompt is one of the key ways to achieving good generations. Examples demonstrate to the model the type of output we target.

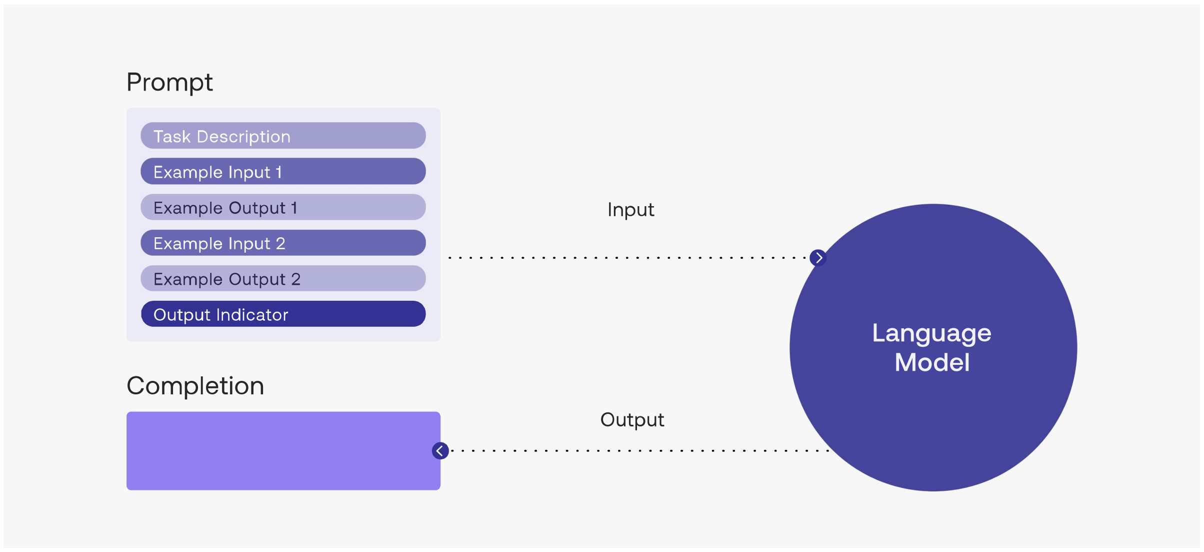
Give a few examples of the types of generations you want. This is called few-shot learning. Let's look at an example. Say, you'd like to use our models to classify whether a movie review is positive, negative or neutral. Imagine that you feed the following prompt into our model:

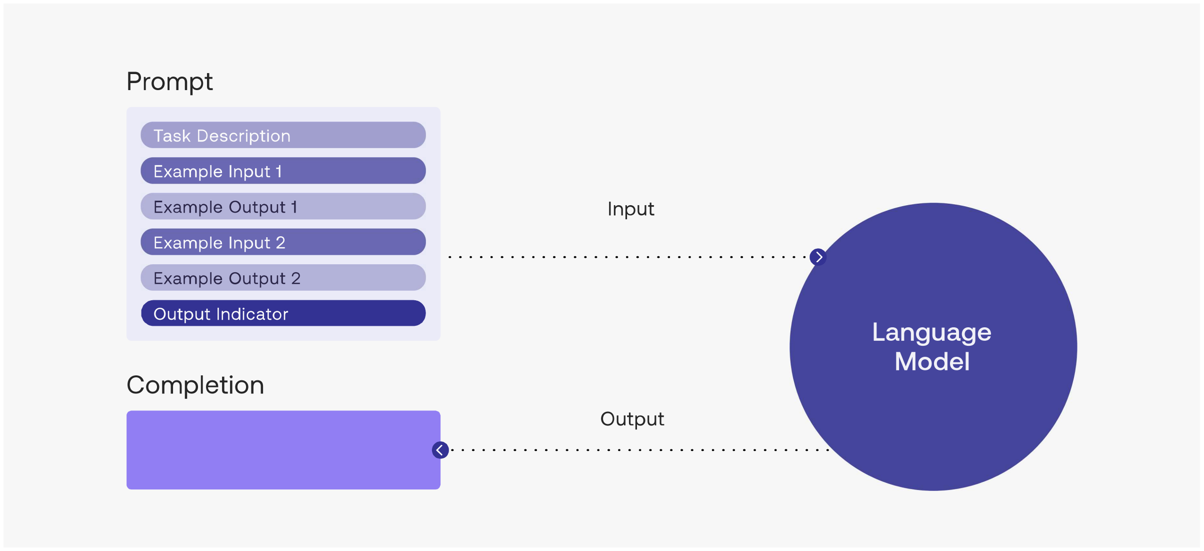
```
Review: "I really enjoyed this movie!"
This sentiment of this review is
```

An actual generation based on this prompt by our Medium model reads:

```
This sentiment of this review is apt, considering the movie's plot,
```

Clearly, there are generations that our model sees as likely that are not the type of generation we'd like to get.



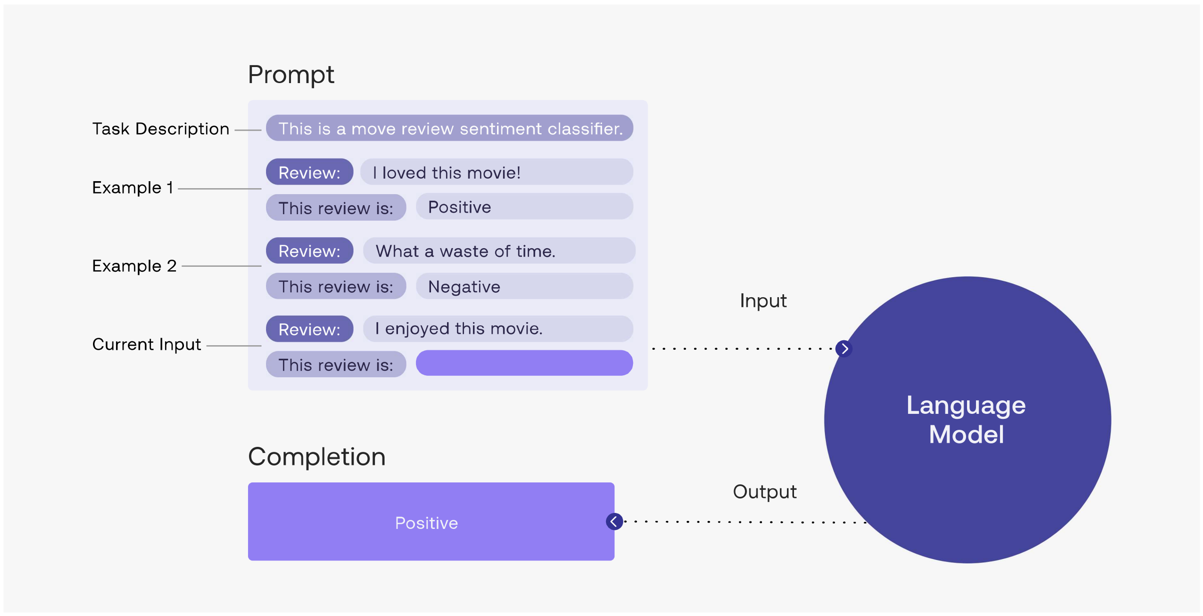


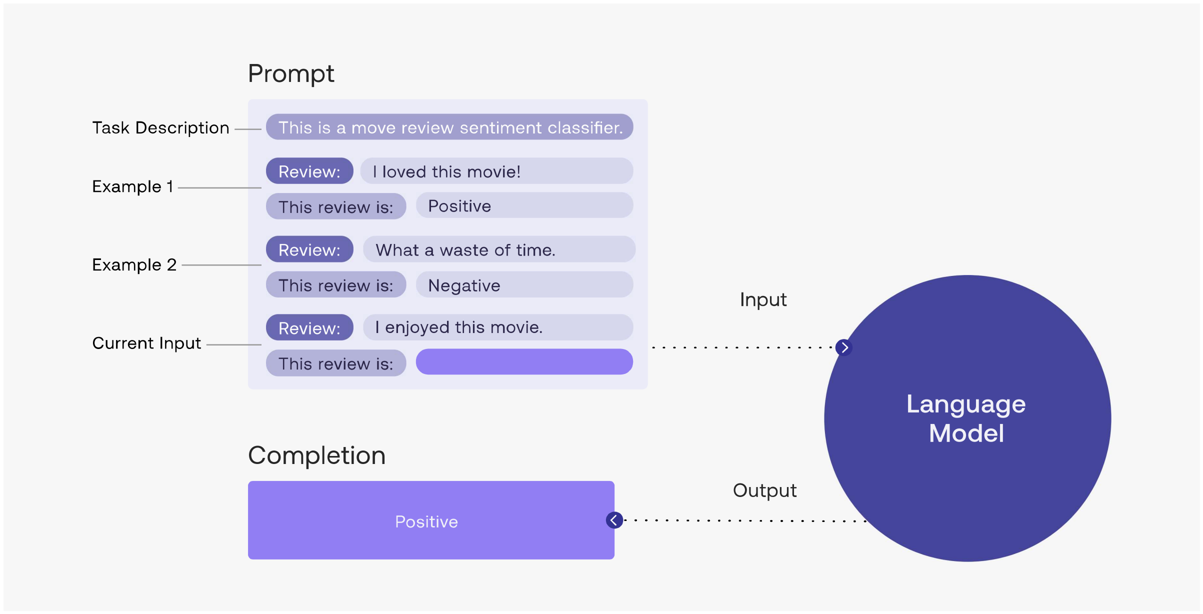
Examples in the prompt should include both an example input and the output we want the model emulate.

Putting this all together and feeding this new prompt into the Medium Generation model, we reliably get the generation `positive`.

```
This is a movie review sentiment classifier.
Review: "I loved this movie!"
This review is positive.
Review: "I don't know, it was ok I guess.."
This review is neutral.
Review: "What a waste of time, would not recommend this movie."
This review is negative.
Review: "I really enjoyed this movie!"
This review is
```

A simpler version of this prompt can be visualized like this:





An example bringing together the various components of a prompt. We can also repeat the task description with each example to emphasize the instruction to the model.

Few-shot generations will generally work better with our larger models. You can use the likelihood endpoint to see how uncertain the model is about the correct answers given in the examples.

If the command format does not work, try structuring as prose. An intuitive way to interact with the generate models is to give commands to the model about the type of generation that you want e.g. `Give a list of artistic professions:`. However, since much of the text that our models have seen is internet articles, sometimes this way of writing will be misunderstood. Try rephrasing the command into prose in a way such that the model will give the desired output:

```
The table lists the following professions as artistic careers:
1. Painter
2.
```

In general, you may want to experiment with different styles of writing until you get something that works. Examples include writing in the style of a news article, a blog post, or a dialogue.

Examples

Here we showcase how we can use to apply the principles above by looking at two specific tasks: generating keywords based on a given passage and generating additional examples given a few existing examples.

Keyword generation: Let's imagine that we have text passages that we'd like to automatically tag with the most relevant concepts appearing in the text.

By combining a number of the techniques discussed above, we can `generate` just that! First, we state what the setting for this prompt is at the beginning of the prompt. Then we show the model two examples of what we want it to do: label a passage from John von Neumann's Wikipedia page with the label "John von Neumann", and label a paragraph from the Wikipedia page on Feminism with the label "Feminism". Lastly, we give the model a passage from the Wikipedia page on Python.

```
This is a bot that automatically finds the most important keyword fo

Text: "John von Neumann (/vɒn ˈnɔɪmən/; Hungarian: Neumann János Laj

Most important key word: "John von Neumann"

Text: "Some scholars consider feminist campaigns to be a main force

Most important key word:  "Feminism"
```

```
Text: "Guido van Rossum began working on Python in the late 1980s, a

Most important key word:
```

This prompt reliably generates "Python" as an answer – while sometimes also returning "Guido van Rossum", another plausible option.

Example generation. A common task is to try to get the model to generate examples according to some description. Formulating the prompt as a list in the following style tends to work well.

```
This is a list of ideas for blog posts for tourists visiting Toronto

1. The best sights to see in Toronto
2. My favourite walks in Toronto
```

which then gives us generations like:

```
3. An overview of Toronto
4. Toronto events
5. Restaurants in Toronto
6. Shopping in Toronto
7. Travel tips for Toronto
8. Sightseeing in Toronto
9. What to do in Toronto
```