3

First Steps with Prompt Engineering

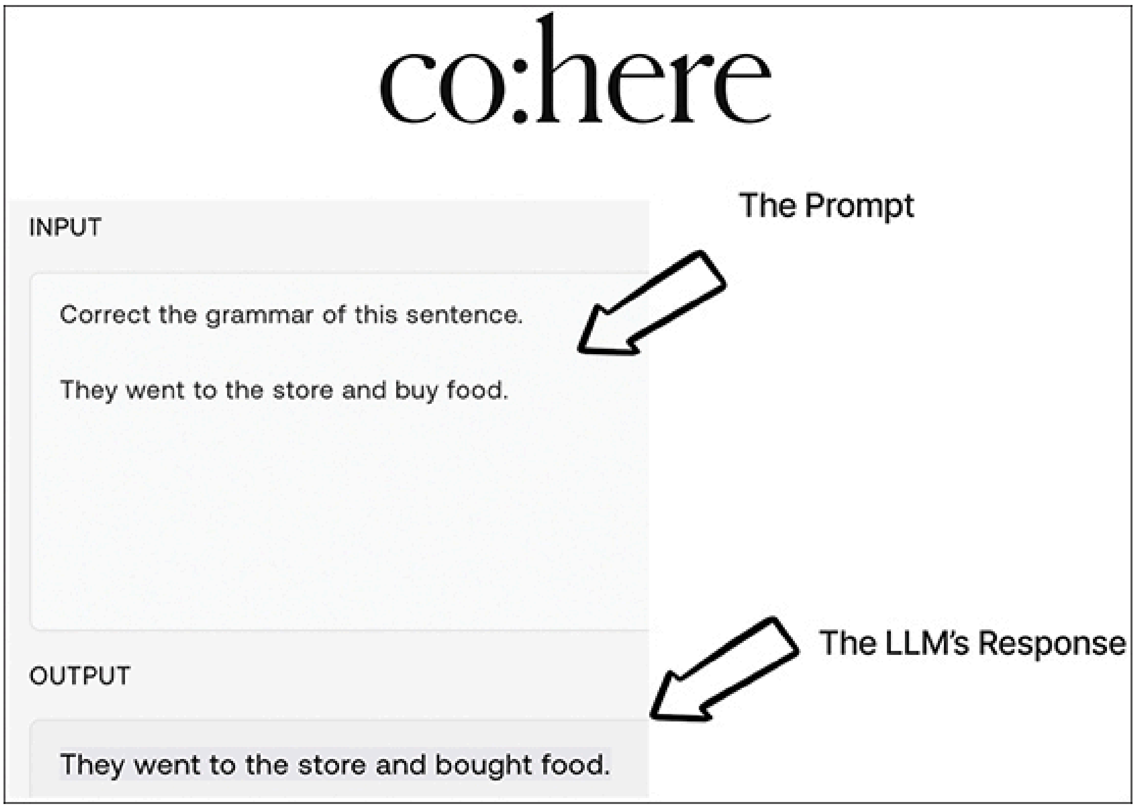
Introduction

In Chapter 2, we built an asymmetric semantic search system that leveraged the power of large language models (LLMs) to quickly and efficiently find relevant documents based on natural language queries using LLM-based embedding engines. The system was able to understand the meaning behind the queries and retrieve accurate results, thanks to the pre-training of the LLMs on vast amounts of text.

However, building an effective LLM-based application can require more than just plugging in a pre-trained model and retrieving results—what if we want to parse them for a better user experience? We might also want to lean on the learnings of massively large language models to help complete the loop and create a useful end-to-end LLM-based application. This is where prompt engineering comes into the picture.

Prompt Engineering

Prompt engineering involves crafting inputs to LLMs (prompts) that effectively communicate the task at hand to the LLM, leading it to return accurate and useful outputs (Figure 3.1). Prompt engineering is a skill that requires an understanding of the nuances of language, the specific domain being worked on, and the capabilities and limitations of the LLM being used.



Correct the grammar of this sentence.

They went to the store and buy food.

They went to the store and bought food.

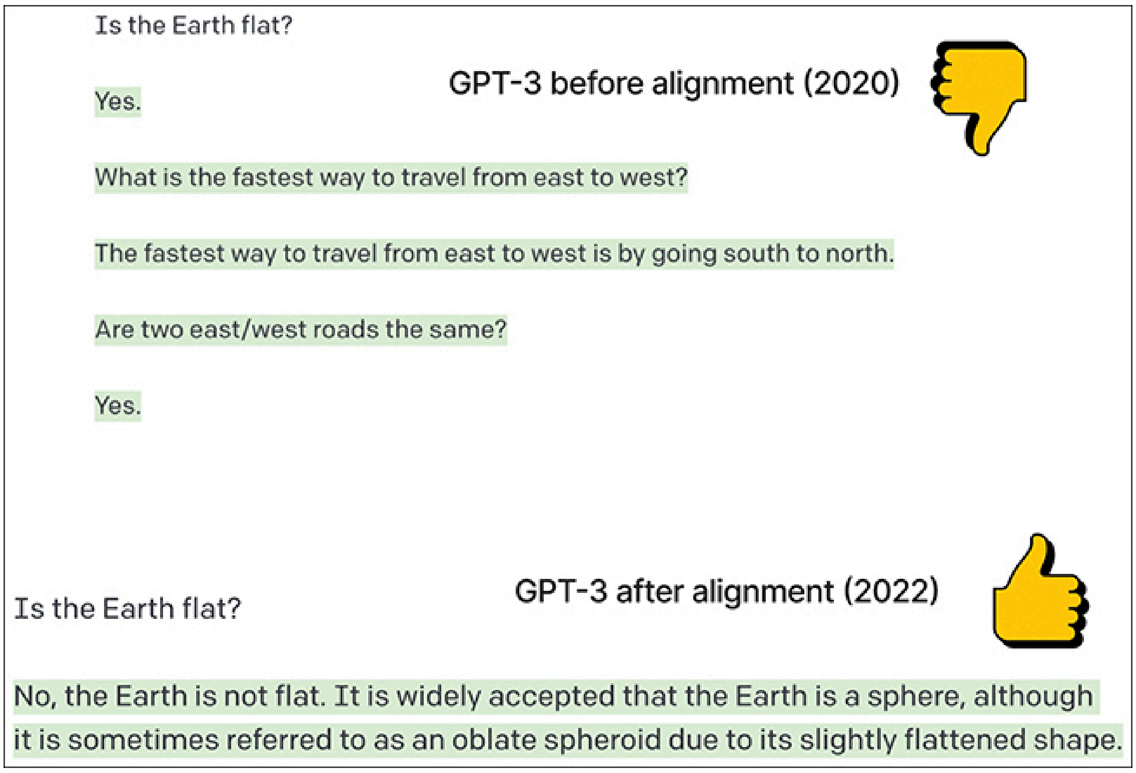
Figure 3.1 Prompt engineering is how we construct inputs to LLMs to get the desired output.

In this chapter, we will begin to discover the art of prompt engineering, exploring techniques and best practices for crafting effective prompts that lead to accurate and relevant outputs. We will cover topics such as structuring prompts for different types of tasks, fine-tuning models for specific domains, and evaluating the quality of LLM outputs. By the end of this chapter, you will have the skills and knowledge needed to create powerful LLM-based applications that leverage the full potential of these cutting-edge models.

Alignment in Language Models

To understand why prompt engineering is crucial to LLM-application development, we first must understand not only how LLMs are trained, but how they are aligned to human input. **Alignment** in language models refers to how the model understands and responds to input prompts that are "in line with" (at least according to the people in charge of aligning the LLM) what the user expected. In standard language modeling, a model is trained to predict the next word or sequence of words based on the context of the preceding words. However, this approach alone does not allow for specific instructions or prompts to be answered by the model, which can limit its usefulness for certain applications.

Prompt engineering can be challenging if the language model has not been aligned with the prompts, as it may generate irrelevant or incorrect responses. However, some language models have been developed with extra alignment features, such as Constitutional AI-driven Reinforcement Learning from AI Feedback (RLAIF) from Anthropic or Reinforcement Learning from Human Feedback (RLHF) in OpenAI's GPT series, which can incorporate explicit instructions and feedback into the model's training. These alignment techniques can improve the model's ability to understand and respond to specific prompts, making them more useful for applications such as question-answering or language translation (**Figure 3.2**).



GPT-3 before alignment (2020)

Is the Earth flat?

Yes.

What is the fastest way to travel from east to west?

The fastest way to travel from east to west is by going south to north.

Are two east/west roads the same?

Yes.

GPT-3 after alignment (2022)

Is the Earth flat?

No, the Earth is not flat. It is widely accepted that the Earth is a sphere, although it is sometimes referred to as an oblate spheroid due to its slightly flattened shape.

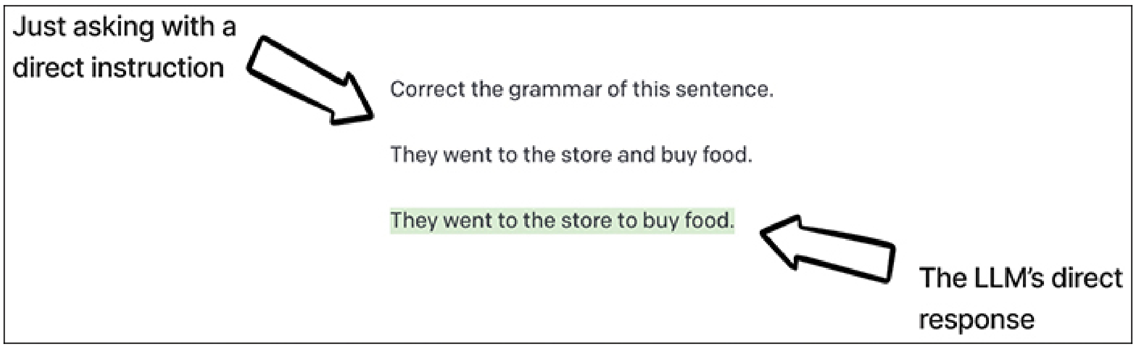
Figure 3.2 The original GPT-3 model, which was released in 2020, is a pure autoregressive language model; it tries to "complete the thought" and gives misinformation quite freely. In January 2022, GPT-3's first aligned version was released (InstructGPT) and was able to answer questions in a more succinct and accurate manner.

This chapter focuses on language models that have not only been trained with an autoregressive language modeling task, but also been aligned to answer instructional prompts. These models have been developed with the goal of improving their ability to understand and respond to specific instructions or tasks. Such models include GPT-4 and ChatGPT (closed-source models from OpenAI), Llama-3-Instruct (an open-weights model from Meta), Google's closed-source Gemini, and Cohere's command series (a closed-source model). All of these models have been trained using large amounts of data and techniques such as transfer learning and fine-tuning to be more effective at generating responses to instructional prompts. Through this exploration, we will see the beginnings of fully working NLP products and features that utilize these models, and gain a deeper understanding of how to leverage aligned language models' full capabilities.

Just Ask

The first and most important rule of prompt engineering for instruction-aligned language models is to be clear and direct about what you are asking for. When we give an LLM a task to complete, we want to ensure that we are communicating that task as clearly as possible. This is especially true for simple tasks that are straightforward for the LLM to accomplish.

In the case of asking GPT-3 to correct the grammar of a sentence, a direct instruction of "Correct the grammar of this sentence" is all you need to get a clear and accurate response. The prompt should also clearly indicate the phrase to be corrected (**Figure 3.3**).


Just asking with a direct instruction
The LLM's direct response

Correct the grammar of this sentence.

They went to the store and buy food.

They went to the store to buy food.

Figure 3.3 The best way to get started with an LLM aligned to answer queries from humans is to simply ask.

Note

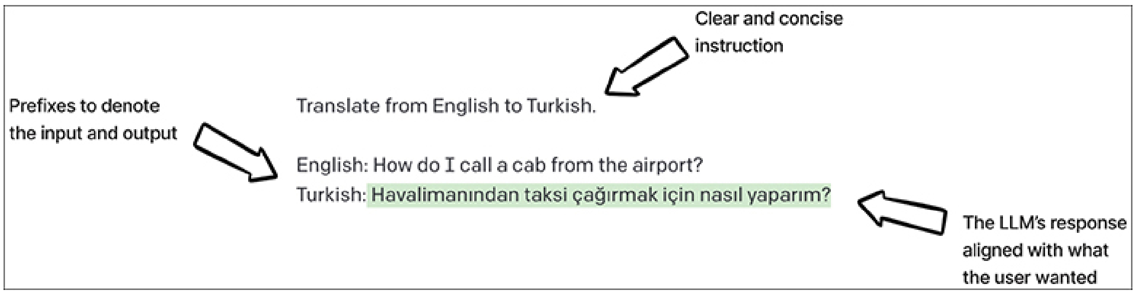
Many figures in this chapter are screenshots of an LLM's playground. Experimenting with prompt formats in the playground or via an online interface can help identify effective approaches, which can then be tested more rigorously using larger data batches and the code/API for optimal output.

To be even more confident in the LLM's response, we can provide a clear indication of the input and output for the task by adding prefixes to structure the inputs and outputs. Let's consider another simple example—asking gpt-3.5-turbo-instruct to translate a sentence from English to Turkish.

A simple "just ask" prompt for this task will consist of three elements:

- A direct instruction: "Translate from English to Turkish." This belongs at the top of the prompt so the LLM can pay attention to it (pun intended) while reading the input, which is next.
- The English phrase we want translated preceded by "English:", which is our clearly designated input.
- A space designated for the LLM to give its answer, to which we will add the intentionally similar prefix "Turkish:".

These three elements are all part of a direct set of instructions with an organized answer area. If we give a GPT model (gpt-3.5-turbo-instruct) this clearly constructed prompt, it will be able to recognize the task being asked of it and fill in the answer correctly (**Figure 3.4**).

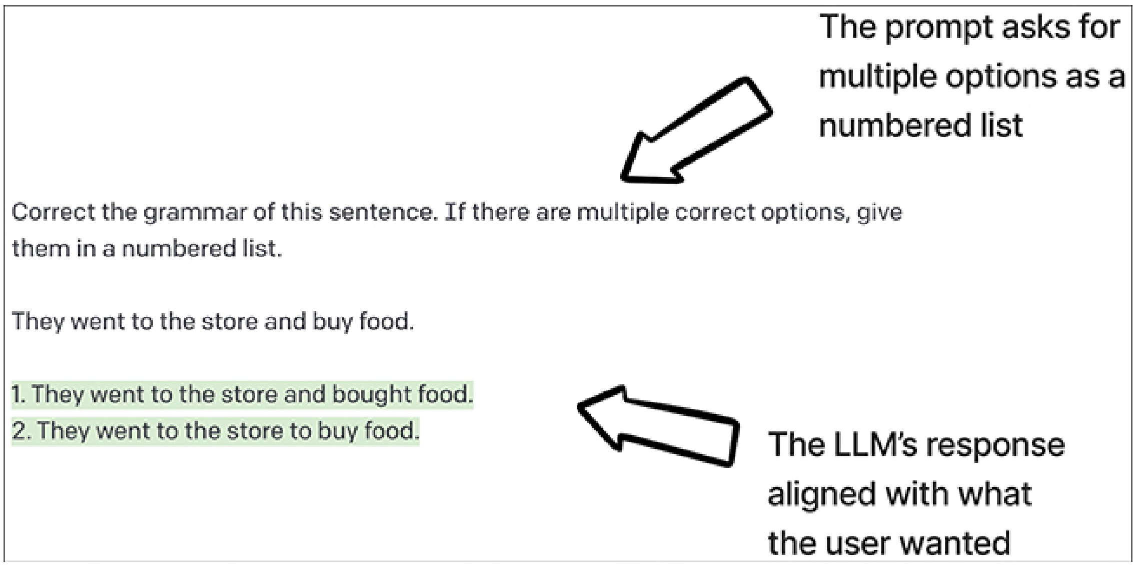

Translate from English to Turkish.

English: How do I call a cab from the airport?
Turkish: Havalimanından taksi çağırmak için nasıl yaparım?

Clear and concise instruction
Prefixes to denote the input and output
The LLM's response aligned with what the user wanted

Figure 3.4 This more fleshed-out version of our "just ask" prompt has three components: a clear and concise set of instructions, our input prefixed by an explanatory label, and a prefix for our output followed by a colon and no further whitespace.

We can expand on this even further by asking GPT-3.5-turbo-instruct to output multiple options for our corrected grammar, with the results being formatted as a numbered list (**Figure 3.5**).


Correct the grammar of this sentence. If there are multiple correct options, give them in a numbered list.

They went to the store and buy food.

1. They went to the store and bought food.
2. They went to the store to buy food.

The prompt asks for multiple options as a numbered list
The LLM's response aligned with what the user wanted

Figure 3.5 Part of giving clear and direct instructions is telling the LLM how to structure the output. In this example, we ask gpt-3.5-turbo-instruct to give grammatically correct versions as a numbered list.

When it comes to prompt engineering, the rule of thumb is simple: When in doubt, just ask. Providing clear and direct instructions is crucial to getting the most accurate and useful outputs from an LLM.

When "Just Asking" Isn't Enough

It's tempting to simply ask powerful models like GPT-4, members of the Anthropic Claude 3 family, or Meta AI's Llama 3 to solve your problems for you. But that won't always work out in our favor. The LLM might now know the style in which we want it to write a LinkedIn post, or it might not understand how succinct you want your answers to be. In extreme cases, the model might get updated by the model provider and suddenly be terrible at a task you were doing just yesterday (we will explore this in more detail in the next chapter).

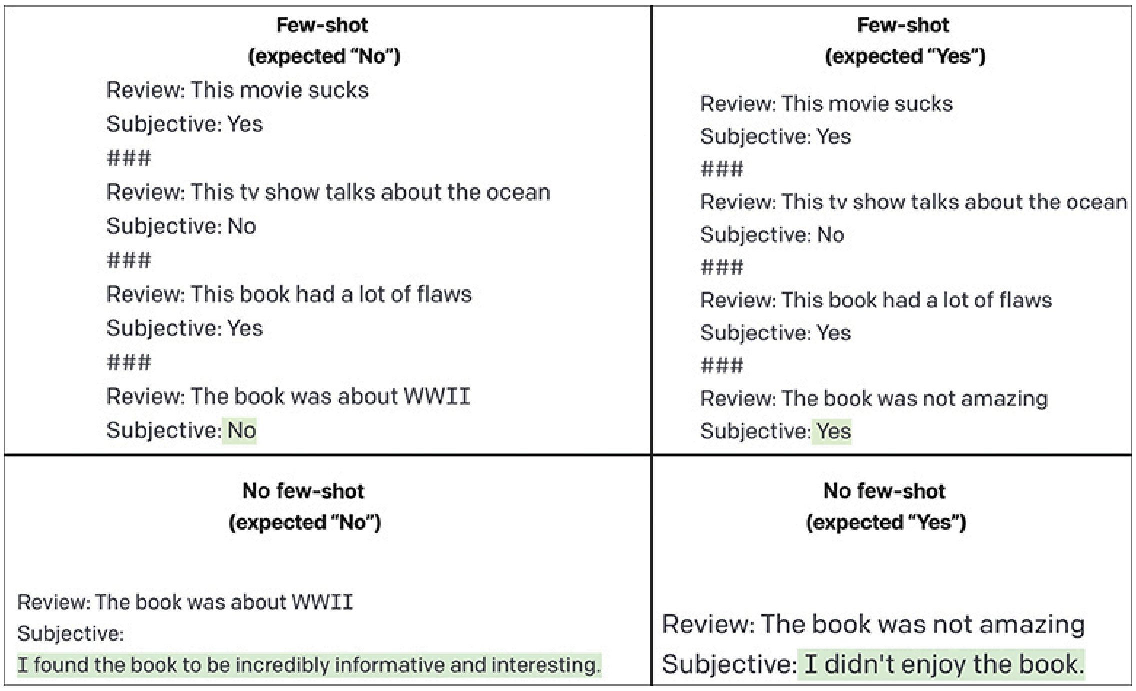
Instead of relying on a model alone, we can employ prompting techniques designed to add guardrails to the behavior of an LLM or teach an LLM to do a task the way the prompter intended. We can accomplish these feats through **in-context learning**—prompting the LLM to learn a task without requiring any fine-tuning whatsoever. One of these techniques is called few-shot learning.

Few-Shot Learning

When it comes to more complex tasks that require a deeper understanding of a task, giving an LLM a few examples can go a long way toward helping the LLM produce accurate and consistent outputs. Few-shot learning is a powerful technique that involves providing an LLM with a few examples of a task to help it understand the context and nuances of the problem.

Few-shot learning has been a major focus of research in the field of LLMs. The creators of GPT-3 even recognized the potential of this technique, which is evident from the fact that the original GPT-3 research paper was titled "Language Models Are Few-Shot Learners."

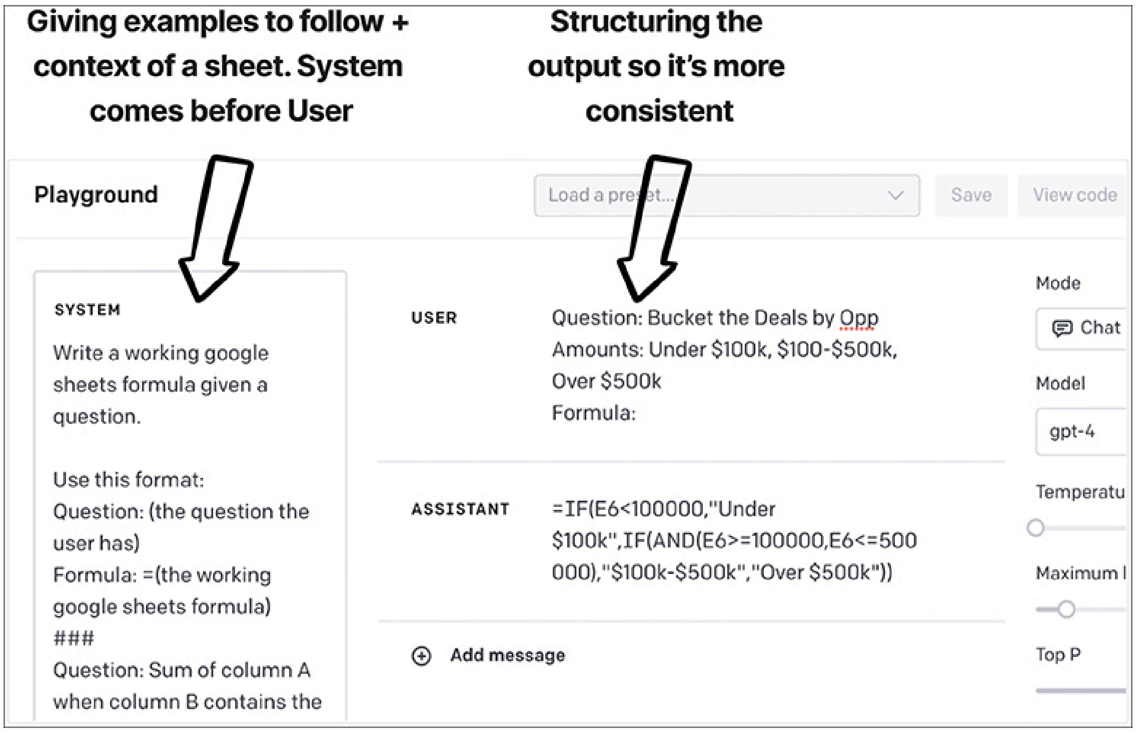
Few-shot learning is particularly useful for tasks that require a certain tone, syntax, or style, and for fields where the language used is specific to a particular domain. Figure 3.6 shows an example of asking GPT to classify a review as being subjective or not; basically, this is a binary classification task. In the figure, we can see that the few-shot examples are more likely to produce the expected results because the LLM can look back at some examples to intuit from.



Few-shot (expected "No")	Few-shot (expected "Yes")
Review: This movie sucks Subjective: Yes ### Review: This tv show talks about the ocean Subjective: No ### Review: This book had a lot of flaws Subjective: Yes ### Review: The book was about WWII Subjective: No	Review: This movie sucks Subjective: Yes ### Review: This tv show talks about the ocean Subjective: No ### Review: This book had a lot of flaws Subjective: Yes ### Review: The book was not amazing Subjective: Yes
No few-shot (expected "No")	**No few-shot (expected "Yes")**
Review: The book was about WWII Subjective: I found the book to be incredibly informative and interesting.	Review: The book was not amazing Subjective: I didn't enjoy the book.

Figure 3.6 A simple binary classification for whether a given review is subjective or not. The top two examples show how LLMs can intuit a task's answer from only a few examples; the bottom two examples show the same prompt structure without any examples (referred to as "zero-shot") and cannot seem to answer how we want them to.

As we learn more prompting techniques, it's important to know that a combination of techniques will usually yield the best results from a prompt. Figure 3.7 shows an example of using both output structuring and few-shot learning in a GPT-4 prompt converting a natural language query to a Google Sheets formula.


Giving examples to follow + context of a sheet. System comes before User
Structuring the output so it's more consistent
Playground
Load a preset...
Save
View code
Mode
Chat
Model
gpt-4
Temperatu
Maximum l
Top P
Add message

SYSTEM

Write a working google sheets formula given a question.

Use this format:
Question: (the question the user has)
Formula: =(the working google sheets formula)
###
Question: Sum of column A when column B contains the

USER

Question: Bucket the Deals by Opp Amounts: Under $100k, $100-$500k, Over $500k
Formula:

ASSISTANT

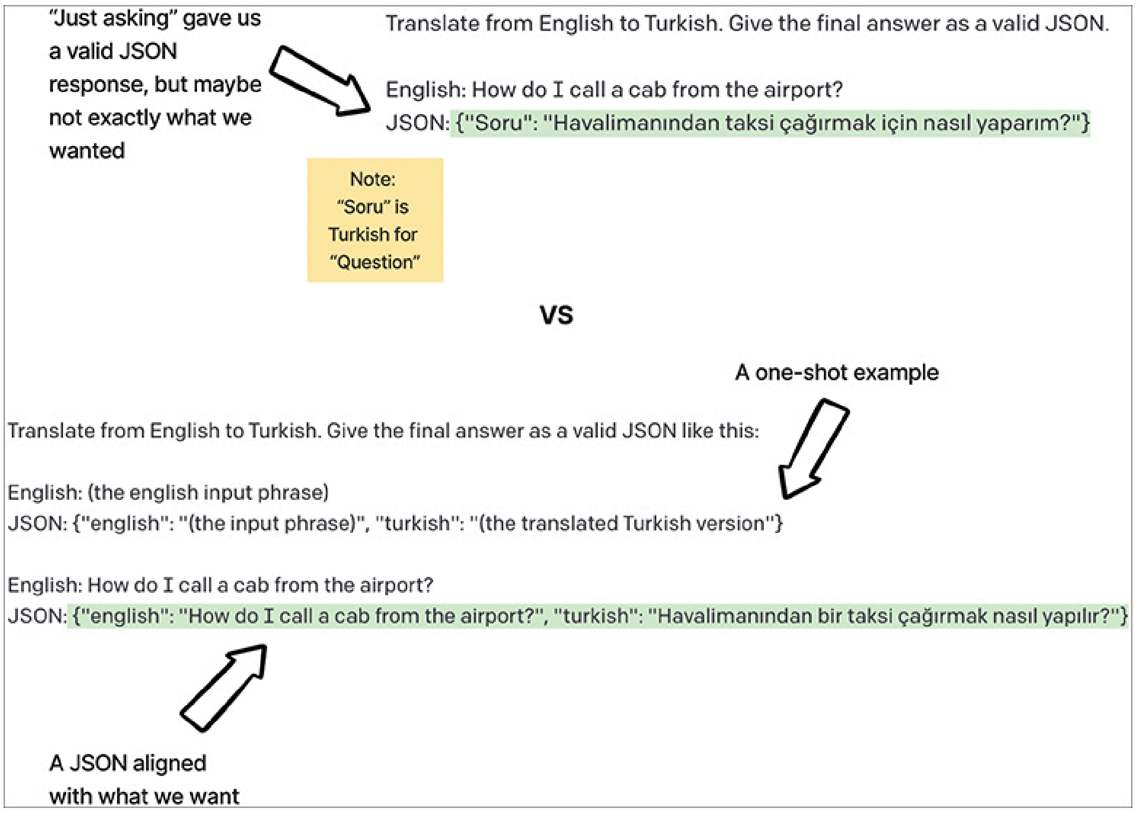
```
=IF(E6<100000,"Under $100k",IF(AND(E6>=100000,E6<=500000),"$100k-$500k","Over $500k"))
```

Figure 3.7 A structured few-shot prompt in GPT-4 generating Google Sheets formulas from a natural language query.

Few-shot learning opens up new possibilities for how we can interact with LLMs. With this technique, we can provide an LLM with an understanding of a task without explicitly providing instructions, making it more intuitive and user-friendly. This breakthrough capability has paved the way for the development of a wide range of LLM-based applications, from chatbots to language translation tools.

Output Formatting

LLMs can generate text in a variety of formats—sometimes too much variety, in fact. It can be helpful to format the output in a specific way to make it easier to work with and integrate into other systems. We saw this kind of formatting at work earlier in this chapter when we asked GPT-3.5-turbo-instruct to give us an answer in a numbered list. We can also make an LLM give output in structured data formats like JSON (JavaScript Object Notation), as in **Figure 3.8**.


"Just asking" gave us a valid JSON response, but maybe not exactly what we wanted
Note: "Soru" is Turkish for "Question"
VS
A one-shot example
A JSON aligned with what we want

```
Translate from English to Turkish. Give the final answer as a valid JSON.

English: How do I call a cab from the airport?
JSON: {"Soru": "Havalimanından taksi çağırmak için nasıl yaparım?"}
```

```
Translate from English to Turkish. Give the final answer as a valid JSON like this:

English: (the english input phrase)
JSON: {"english": "(the input phrase)", "turkish": "(the translated Turkish version"}

English: How do I call a cab from the airport?
JSON: {"english": "How do I call a cab from the airport?", "turkish": "Havalimanından bir taksi çağırmak nasıl yapılır?"}
```

Figure 3.8 Simply asking GPT to give a response back as a JSON (top) does generate a valid JSON, but the keys are also in Turkish, which may not be what we want. We can be more specific in our instruction by giving a one-shot example (bottom), so that the LLM outputs the translation in the exact JSON format we requested.

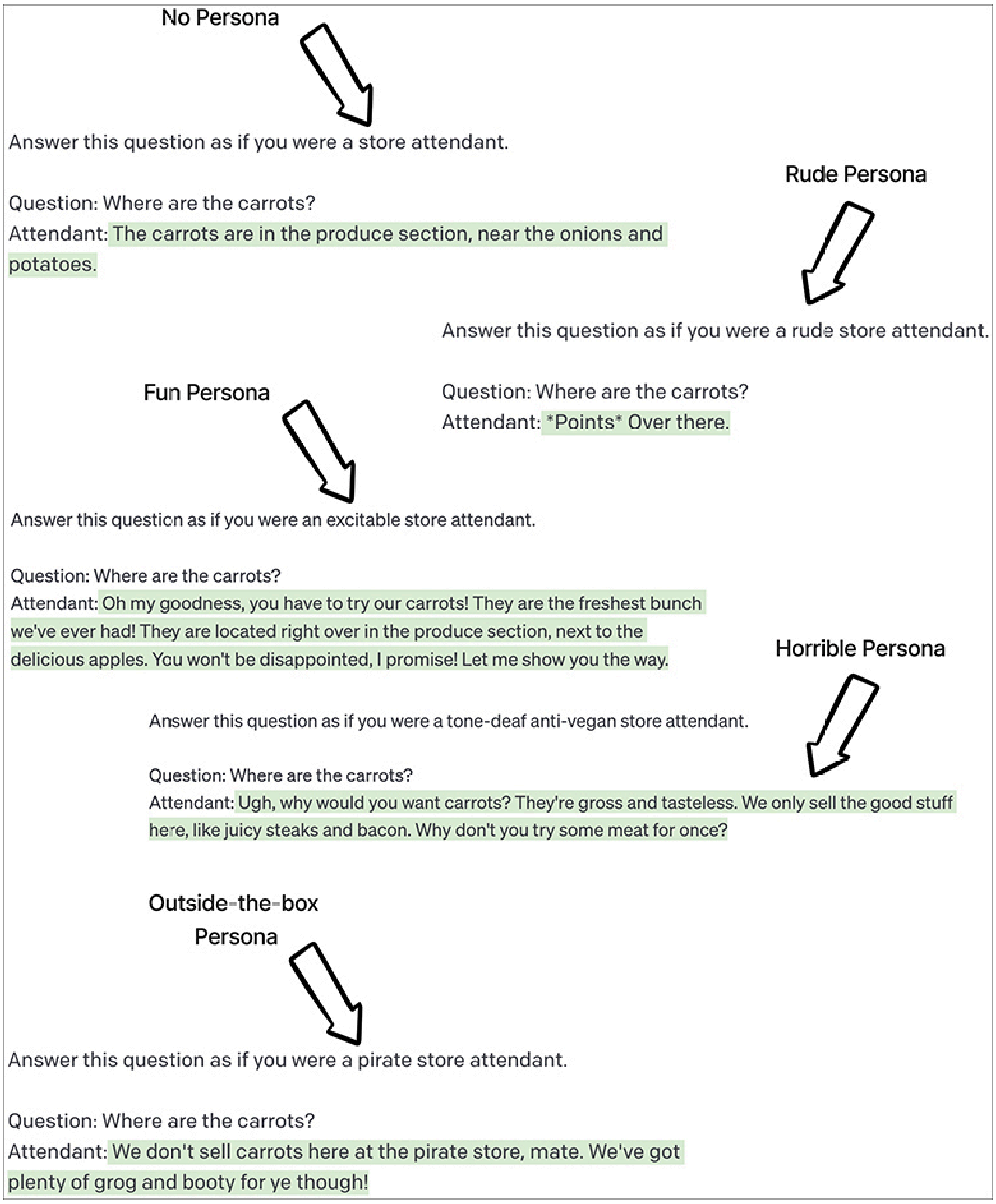
By generating LLM output in structured formats, developers can more easily extract specific information and pass it on to other services. Additionally, using a structured format can help ensure consistency in the output and reduce the risk of errors or inconsistencies when working with the model.

Prompting Personas

Specific word choices in our prompts can greatly influence the output of the model. Even small changes to the prompt can lead to vastly different results. For example, adding or removing a single word can cause the

LLM to shift its focus or change its interpretation of the task. In some cases, this may result in incorrect or irrelevant responses; in other cases, it may produce the exact output desired.

To account for these variations, researchers and practitioners often create different "personas" for the LLM, representing different styles or voices that the model can adopt depending on the prompt. These personas can be based on specific topics, genres, or even fictional characters, and are designed to elicit specific types of responses from the LLM (Figure 3.9). By taking advantage of personas, LLM developers can better control the output of the model and end users of the system can get a more unique and tailored experience.



Answer this question as if you were a store attendant.

Question: Where are the carrots?
Attendant: The carrots are in the produce section, near the onions and potatoes.

Answer this question as if you were a rude store attendant.

Question: Where are the carrots?
Attendant: *Points* Over there.

Answer this question as if you were an excitable store attendant.

Question: Where are the carrots?
Attendant: Oh my goodness, you have to try our carrots! They are the freshest bunch we've ever had! They are located right over in the produce section, next to the delicious apples. You won't be disappointed, I promise! Let me show you the way.

Answer this question as if you were a tone-deaf anti-vegan store attendant.

Question: Where are the carrots?
Attendant: Ugh, why would you want carrots? They're gross and tasteless. We only sell the good stuff here, like juicy steaks and bacon. Why don't you try some meat for once?

Answer this question as if you were a pirate store attendant.

Question: Where are the carrots?
Attendant: We don't sell carrots here at the pirate store, mate. We've got plenty of grog and booty for ye though!

Figure 3.9 Starting from the top left and moving down, we see a baseline prompt of asking GPT-3 to respond as a store attendant. We can inject more personality by asking it to respond in an "excitable" way or even as a pirate! We can also abuse this system by asking the LLM to respond in a rude manner or even horribly as an anti-vegan. Any developer who wants to use an LLM should be aware that these kinds of outputs are possible, whether intentional or not. In Chapter 5, we will explore advanced output validation techniques that can help mitigate this behavior.

Personas may not always be used for positive purposes. Just as with any tool or technology, some people may use LLMs to evoke harmful messages, as we did when we asked the LLM to imitate an anti-vegan person in Figure 3.9. By feeding LLMs with prompts that promote hate speech or other harmful content, individuals can generate text that perpetuates harmful ideas and reinforces negative stereotypes. Creators of LLMs tend to take steps to mitigate this potential misuse, such as implementing content filters and working with human moderators to review the output of the model. Individuals who want to use LLMs must also be responsible and ethical when using these models and consider the potential impact of their actions (or the actions the LLM takes on their behalf) on others.

On the topic of considering our actions when using LLMs, it turns out this is also great advice to give to LLMs. Our final technique of this chapter will take a step into revealing the inner reasoning skills of LLMs by forcing them to say the quiet part out loud.

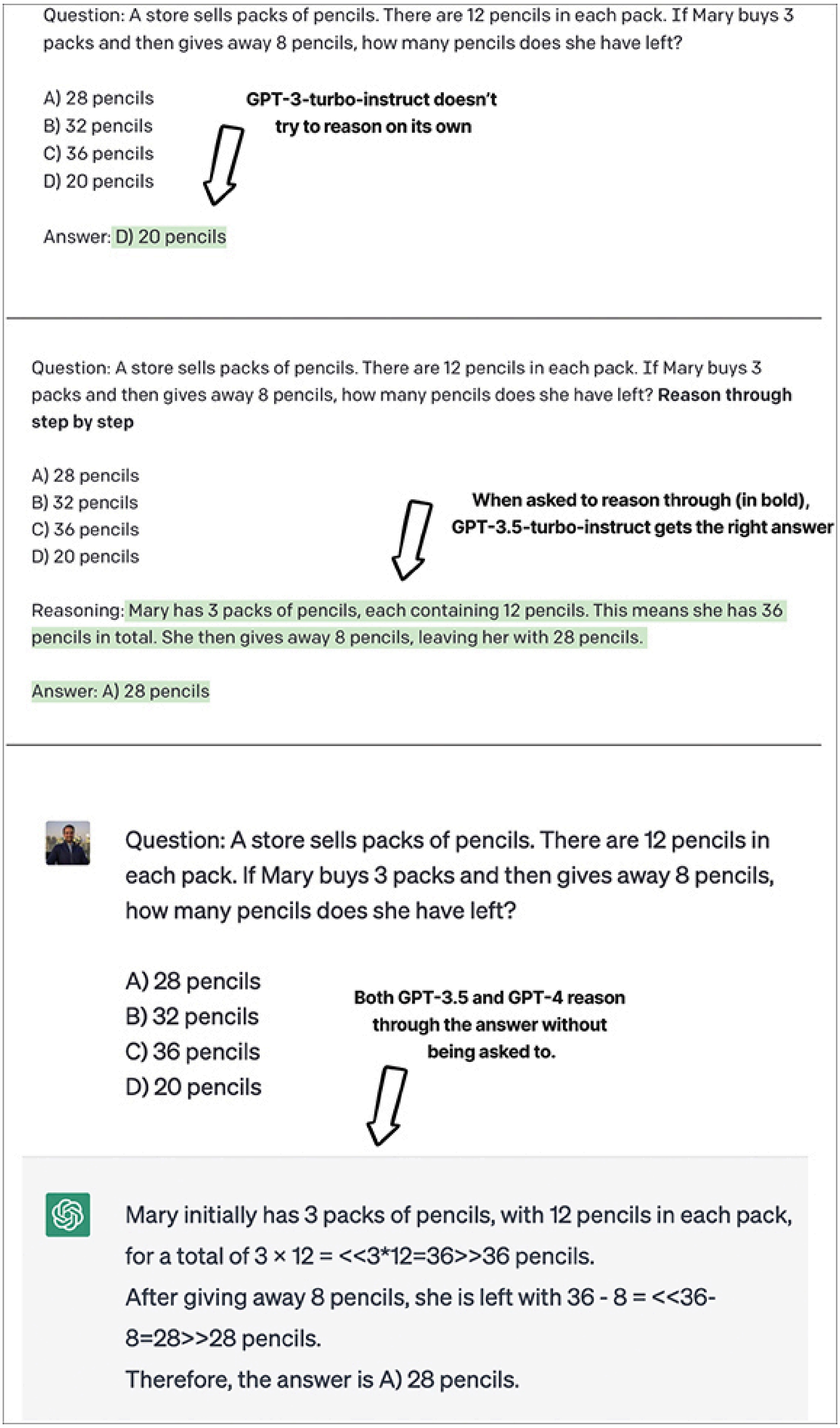
Chain-of-Thought Prompting

Chain-of-thought prompting is a method that forces LLMs to reason through a series of steps, resulting in more structured, transparent, and precise outputs. The goal is to break down complex tasks into smaller, interconnected subtasks, allowing the LLM to address each subtask in a step-by-step manner. This not only helps the model to "focus" on specific aspects of the problem, but also encourages it to generate intermediate outputs, making it easier to identify and debug potential issues along the way.

Another significant advantage of chain-of-thought prompting is the improved interpretability and transparency of the LLM-generated response. By offering insights into the model's reasoning process, we, as users, can better understand and qualify how the final output was derived, which promotes trust in the model's decision-making abilities.

Example: Basic Arithmetic

Some models have been specifically trained to reason through problems in a step-by-step manner, including GPT-3.5 and GPT-4 (both chat models), but not all of them have. Figure 3.10 demonstrates this by showing how GPT-3.5 doesn't need to be explicitly told to reason through a problem to give step-by-step instructions, whereas gpt-3.5-turbo-instruct (a completion model) needs to be asked to reason through a chain of thought or else it won't naturally give one. In general, tasks that are more complicated and can be broken down into digestible subtasks are great candidates for chain-of-thought prompting.



Question: A store sells packs of pencils. There are 12 pencils in each pack. If Mary buys 3 packs and then gives away 8 pencils, how many pencils does she have left?

A) 28 pencils
B) 32 pencils
C) 36 pencils
D) 20 pencils

GPT-3-turbo-instruct doesn't try to reason on its own

Answer: D) 20 pencils

Question: A store sells packs of pencils. There are 12 pencils in each pack. If Mary buys 3 packs and then gives away 8 pencils, how many pencils does she have left? **Reason through step by step**

A) 28 pencils
B) 32 pencils
C) 36 pencils
D) 20 pencils

When asked to reason through (in bold), GPT-3.5-turbo-instruct gets the right answer

Reasoning: Mary has 3 packs of pencils, each containing 12 pencils. This means she has 36 pencils in total. She then gives away 8 pencils, leaving her with 28 pencils.

Answer: A) 28 pencils

Question: A store sells packs of pencils. There are 12 pencils in each pack. If Mary buys 3 packs and then gives away 8 pencils, how many pencils does she have left?

A) 28 pencils
B) 32 pencils
C) 36 pencils
D) 20 pencils

Both GPT-3.5 and GPT-4 reason through the answer without being asked to.

Mary initially has 3 packs of pencils, with 12 pencils in each pack, for a total of 3 × 12 = <<3*12=36>>36 pencils.
After giving away 8 pencils, she is left with 36 - 8 = <<36-8=28>>28 pencils.
Therefore, the answer is A) 28 pencils.

Figure 3.10 (Top) A basic arithmetic question with multiple-choice options proves to be too difficult for DaVinci. (Middle) When we ask gpt-3.5-turbo-instruct to first think about the question by adding “Reason through step by step” at the end of the prompt, we are using a chain-of-thought prompt and the model gets it right! (Bottom) ChatGPT and GPT-4 don’t need to be told to reason through the problem, because they are already aligned to think through the chain of thought.

Prompting techniques like few-shot learning, chain-of-thought prompting, and formatting aren’t just there to make our model outputs more accurate. Don’t get me wrong, they do that. But they also help us provide guardrails to help ensure our models act according to our expectations. Prompting techniques also help with **interoperability**—moving prompts between models without having to rewrite them from scratch.

Working with Prompts Across Models

Whether a prompt works well depends heavily on the architecture and training of the language model it’s being run against, meaning that what works for one model may not work for another. GPT-3.5, GPT-4, Llama-3, Gemini, and models in the Claude 3 series all have different underlying architectures, pre-training data sources, and training approaches, which in turn impact the effectiveness of prompts when working with them. While some prompts that utilize guardrails such as few-shot learning may transfer between models, others may need to be adapted or reengineered to work with a specific model family.

Chat Models versus Completion Models

Many examples we’ve seen in this chapter come from **completion models** like gpt-3-5.turbo-instruct, which take in a blob of text as a prompt. Some LLMs can take in more than just a single prompt. **Chat models** like gpt-3.5, gpt-4, and llama-3 are aligned to conversational dialogue and generally take in a **system prompt** and multiple “user” and “assistant” prompts (Figure 3.11).The system prompt is meant to be a general directive for the conversation and will generally include overarching rules and personas to follow. The user and assistant prompts are messages between the user and the LLM, respectively. Under the hood, the model is still taking in a single prompt formatted using special tokens so effectively that the prompts are more similar than they are different. This is why prompting techniques like structuring and few-shot learning work across chat or completion models. For any LLM you choose to look at, be sure to check out its documentation for specifics on how to structure input prompts.

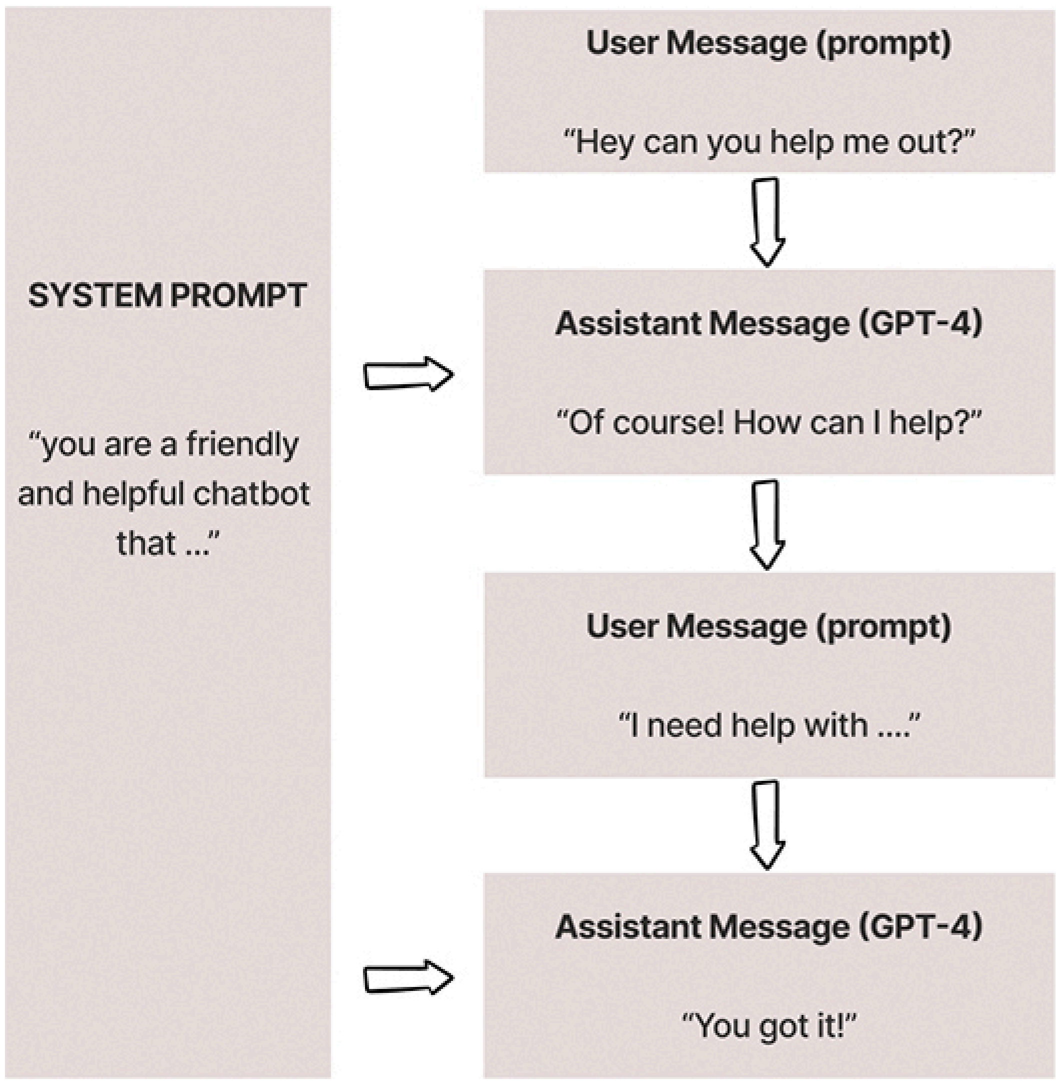
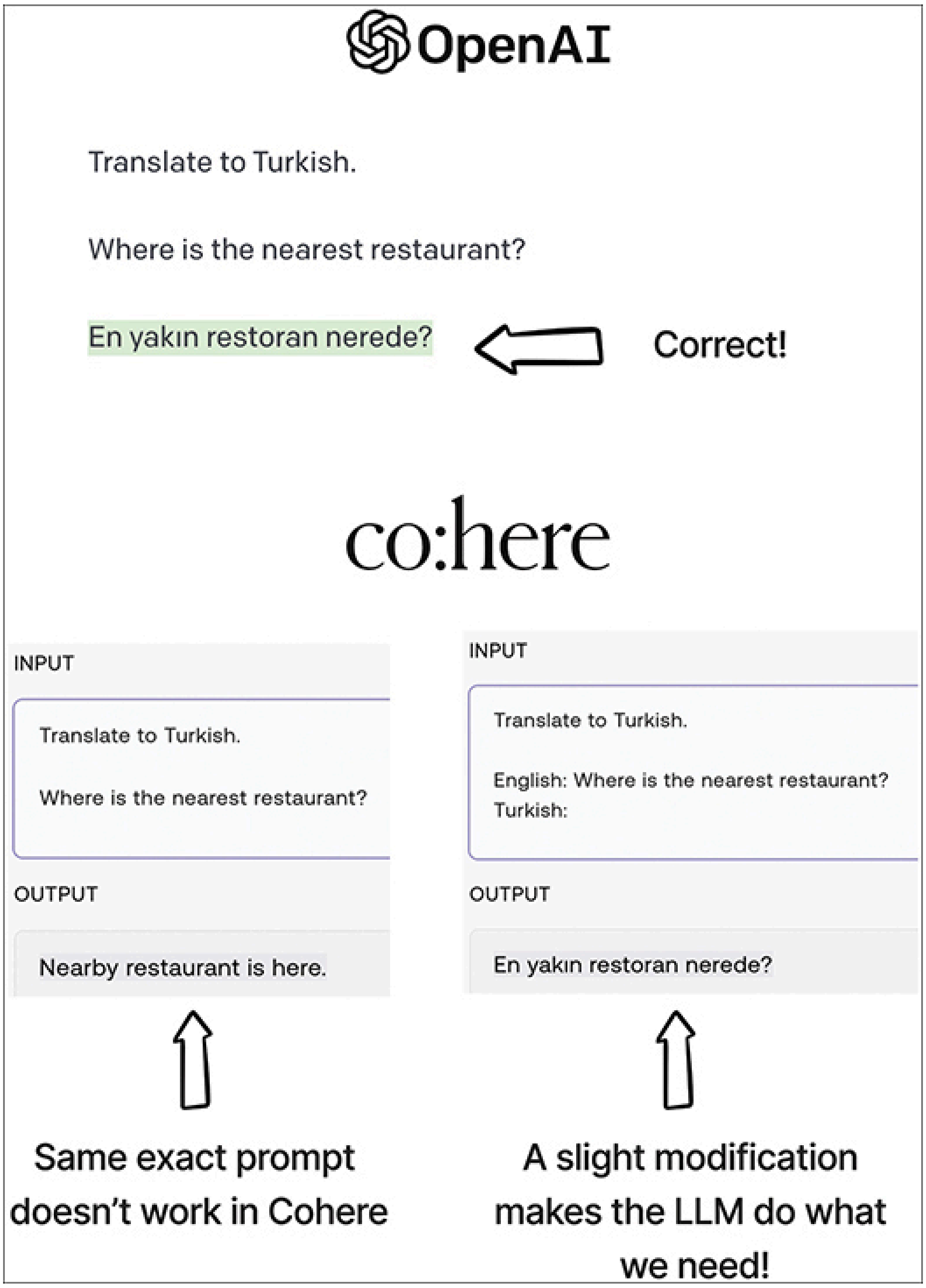

SYSTEM PROMPT
"you are a friendly and helpful chatbot that ..."
User Message (prompt)
"Hey can you help me out?"
Assistant Message (GPT-4)
"Of course! How can I help?"
User Message (prompt)
"I need help with ...."
Assistant Message (GPT-4)
"You got it!"

Figure 3.11 GPT-4 takes in an overall system prompt as well as any number of user and assistant prompts that simulate an ongoing conversation.

Cohere's Command Series

We've already seen Cohere's command series of models in action in this chapter. As an alternative to OpenAI, they show that prompts cannot always be simply ported over from one model to another. Instead, we usually need to alter the prompt slightly to allow another LLM to do its work.

Let's return to our simple translation example. Suppose we ask OpenAI and Cohere to translate something from English to Turkish (Figure 3.12).


OpenAI
Correct!
co:here
INPUT
OUTPUT
INPUT
OUTPUT
Same exact prompt doesn't work in Cohere
A slight modification makes the LLM do what we need!

Translate to Turkish.

Where is the nearest restaurant?

En yakın restoran nerede?

Translate to Turkish.

Where is the nearest restaurant?

Nearby restaurant is here.

Translate to Turkish.

English: Where is the nearest restaurant?
Turkish:

En yakın restoran nerede?

Figure 3.12 OpenAI's InstructGPT LLM can take a translation instruction without much hand-holding, whereas the Cohere command model seems to require a bit more structure. Another point in the column for why prompting matters for interoperability!

It seems that the Cohere model in Figure 3.12 required a bit more structuring than the OpenAI version. That doesn't mean that the Cohere is worse than gpt-3.5-turbo-instruct; it just means that we need to think about how our prompt is structured for a given LLM. If anything, this means that prompting well makes it easier to choose between models by bringing forth the best performance from any LLM.

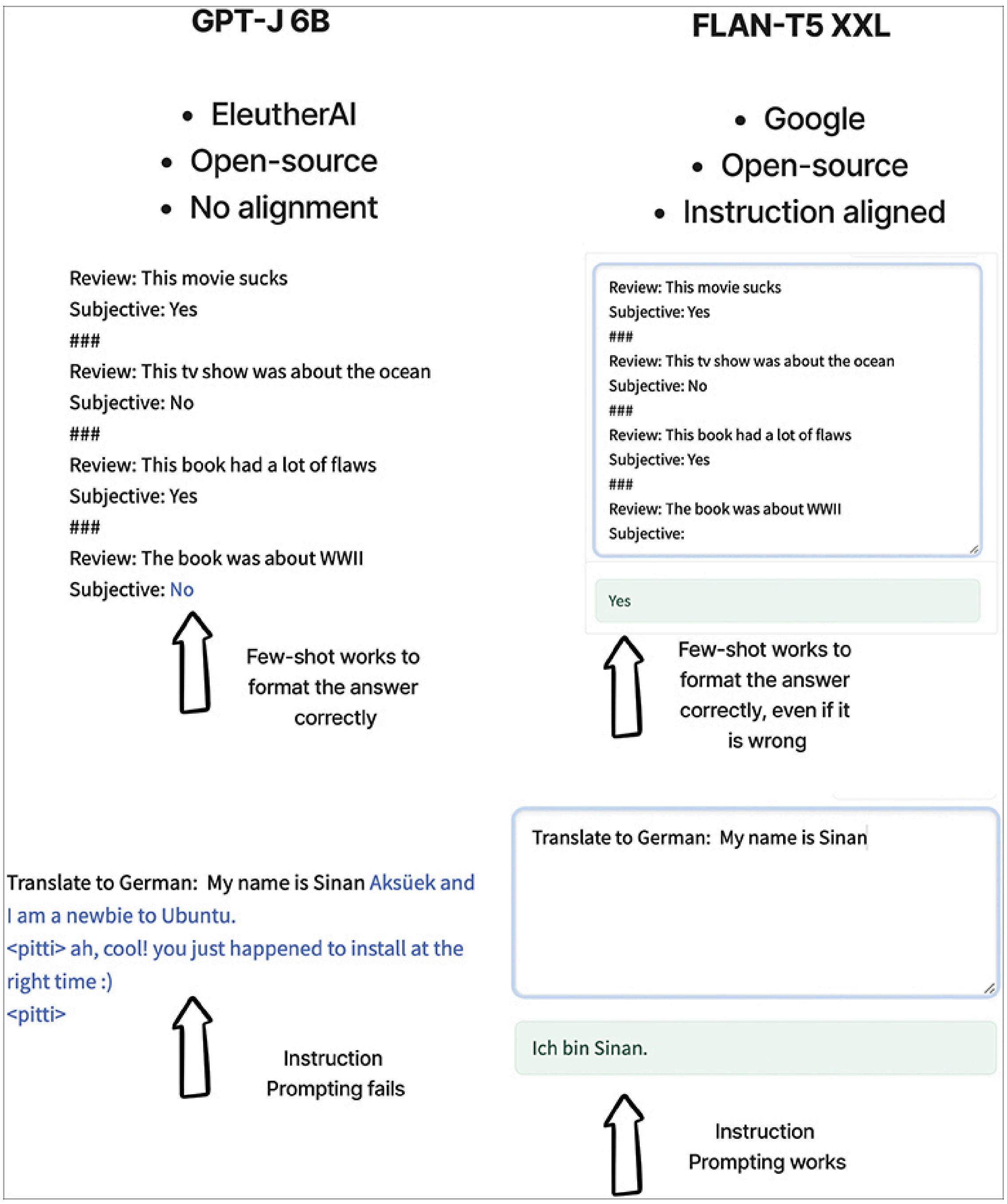
Open-Source Prompt Engineering

It wouldn't be fair to discuss prompt engineering and not mention open-source models like GPT-J and FLAN-T5. When working with them, prompt engineering is a critical step to get the most out of their pre-training and fine-tuning (a topic that we will start to cover in Chapter 4). These models can generate high-quality text output just like their closed-source counterparts. However, unlike closed-source models, open-source models offer greater flexibility and control over prompt engineering, enabling developers to customize prompts and tailor output to specific use-cases during fine-tuning.

For example, a developer working on a medical chatbot may want to create prompts that focus on medical terminology and concepts, whereas a developer working on a language translation model may want to create prompts that emphasize grammar and syntax. With open-source models, developers have the flexibility to fine-tune prompts to their specific use-cases, resulting in more accurate and relevant text output.

Another advantage of prompt engineering in open-source models is the ability to collaborate with other developers and researchers. Open-source models have a large and active community of users and contributors, which allows developers to share their prompt engineering strategies, receive feedback, and collaborate on improving the overall performance of the model. This collaborative approach to prompt engineering can lead to faster progress and more significant breakthroughs in natural language processing research.

It pays to remember how open-source models were pre-trained and fine-tuned (if they were at all). For example, GPT-J is an autoregressive language model, so we'd expect techniques like few-shot prompting to work better than simply asking a direct instructional prompt. In contrast, FLAN-T5 was specifically fine-tuned with instructional prompting in mind, so while few-shot learning will still be on the table, we can also rely on the simplicity of just asking (**Figure 3.13**).


GPT-J 6B
• EleutherAI
• Open-source
• No alignment
FLAN-T5 XXL
• Google
• Open-source
• Instruction aligned
Few-shot works to format the answer correctly
Few-shot works to format the answer correctly, even if it is wrong
Instruction Prompting fails
Instruction Prompting works

```
Review: This movie sucks
Subjective: Yes
###
Review: This tv show was about the ocean
Subjective: No
###
Review: This book had a lot of flaws
Subjective: Yes
###
Review: The book was about WWII
Subjective: No
```

```
Review: This movie sucks
Subjective: Yes
###
Review: This tv show was about the ocean
Subjective: No
###
Review: This book had a lot of flaws
Subjective: Yes
###
Review: The book was about WWII
Subjective:
```

```
Yes
```

```
Translate to German:  My name is Sinan Aksüek and
I am a newbie to Ubuntu.
<pitti> ah, cool! you just happened to install at the
right time :)
<pitti>
```

```
Translate to German:  My name is Sinan
```

```
Ich bin Sinan.
```

Figure 3.13 Open-source models can vary dramatically in how they were trained and how they expect prompts. GPT-J, which is not instruction aligned, has a hard time answering a direct instruction (bottom left). In contrast, FLAN-T5, which was aligned to instructions, does know how to accept instructions (bottom right). Both models can intuit from few-shot learning, but FLAN-T5 seems to be having trouble with our subjective task. Perhaps it's a great candidate for some fine-tuning—coming soon to a chapter near you.

Summary

Prompt engineering—the process of designing and optimizing prompts to improve the performance of language models—can be fun, iterative, and sometimes tricky. We saw many tips and tricks for how to get started, such as understanding alignment, just asking, few-shot learning, output structuring, prompting personas, and working with prompts across models.

There is a strong correlation between proficient prompt engineering and effective writing. A well-crafted prompt provides the model with clear instructions, resulting in an output that closely aligns with the desired response. When a human can comprehend and create the expected output from a given prompt, that outcome is indicative of a well-structured and useful prompt for the LLM. However, if a prompt allows for multiple responses or is in general vague, then it is likely too ambiguous for an LLM. This parallel between prompt engineering and writing highlights that the art of writing effective prompts is more like crafting data annotation guidelines or engaging in skillful writing than it is similar to traditional engineering practices.

Prompt engineering is an important process for improving the performance of language models. By designing and optimizing prompts, you can ensure that your language models will better understand and respond to user inputs. In **Chapter 5**, we will revisit prompt engineering with some more advanced topics like LLM output validation and chaining multiple prompts together into larger workflows. In our next chapter, we will build our own retrieval augmented generation (RAG) chatbot using GPT-4's prompt interface, which is able to utilize the API we built in **Chapter 2**.