Chapter 6. Prompt Engineering

Introduction

- Models trained primarily for text generation, models that are often referred to as generative pre-trained transformers(GPT).
- These models have the remarkable ability to generate text in response to prompts from the user.
- Through prompt engineering, we can design these prompts in a way that enhances the quality of the generated text.
-

Choosing a GPT model

- Proprietary models or open source models.
- Foundation models or fine-tuned models.
- Smaller models or Larger models

let's continue using Phi-3-mini, which has 3.8 billion parameters. This makes it suitable for running with devices up to 8 GB of VRAM.

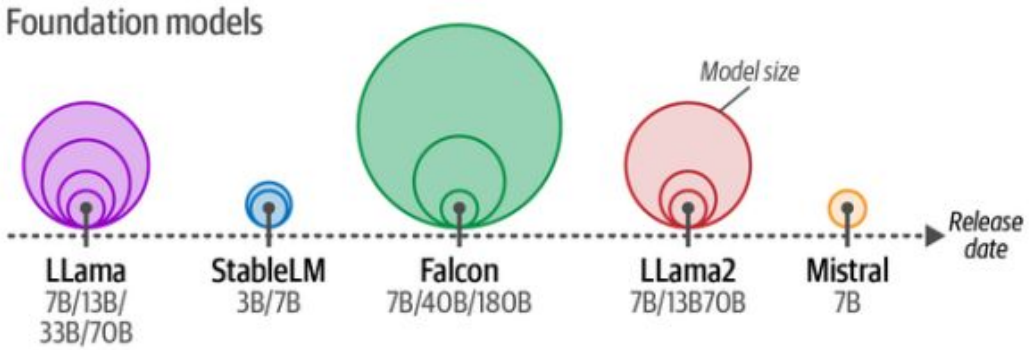

Figure 6-1. Foundation models are often released in several different sizes.

How we can control the output of a GPT model

Other than prompt engineering, we can control the kind of output we want by adjusting the model parameters such as temperature.

In GPT models, "temperature" is a parameter controls that the randomness and creativity of the generated text. A low temperature (e.g., 0 to 0.3) Makes the output more predictable and deterministic, using the most probable words. A high temperature (e.g., 0.7 to 1.0) results in more random, creative, and diverse text, sometimes but can reduce coherence.

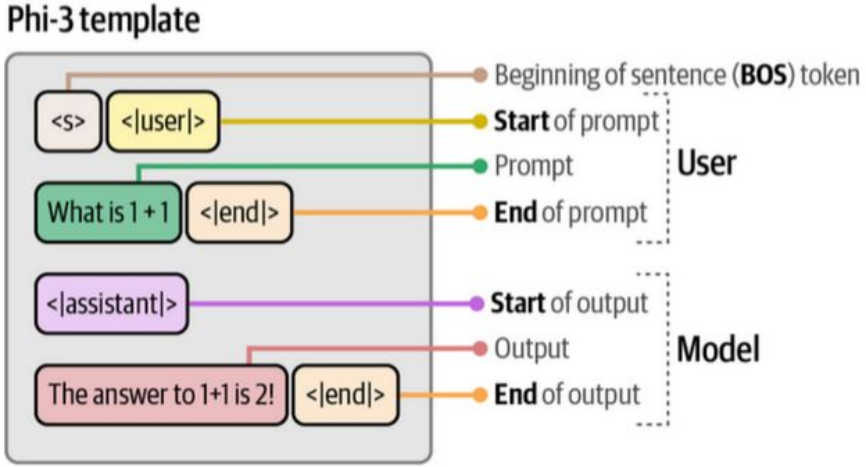

Figure 6-2. The template Phi-3 expects when interacting with the model.

How we can control the output of a GPT model

- A part of what makes LLMs exciting technology is that it can generate different responses for the exact same prompt.
- Each time an LLM needs to generate a token, it assigns a likelihood number to each possible token.

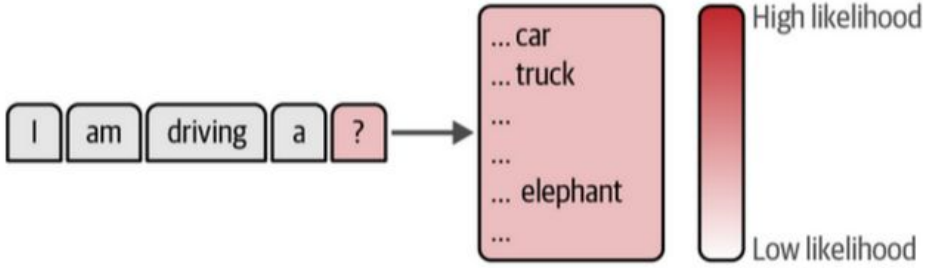

Figure 6-3. The model chooses the next token to generate based on their likelihood scores.

Therefore Parameters like temperature can control if the output should be car, truck or elephant

As illustrated in Figure 6-3, in the sentence “I am driving a...” the likelihood of that sentence being followed by tokens like “car” or “truck” is generally higher than a token like “elephant.” However, there is still a possibility of “elephant” being generated but it is much lower.

How temperature can control the output?

As illustrated in Figure 6-4, a higher value of temperature allows less probable words to be generated.

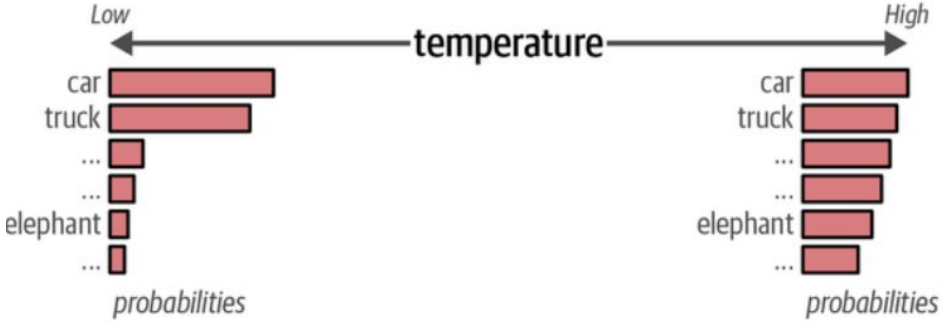

Figure 6-4. A higher `temperature` increases the likelihood that less probable tokens are generated and vice versa.

How to use the temperature parameter in code

You can use `temperature` in your pipeline as follows:

```
# Using a high temperature
output = pipe(messages, do_sample=True, temperature=1)
print(output[0]["generated_text"])
```

```
Why don't chickens like to go on a rollercoaster? Because
they're afraid they might suddenly become chicken-soup!
```

Note that every time you rerun this piece of code, the output will change! `temperature` introduces stochastic behavior since the model now randomly selects tokens.

Controlling the output of Gpt Model using top_p and top_k

top_p, also known as nucleus sampling, is a sampling technique that controls which subset of tokens (the nucleus) the LLM can consider. It will consider tokens until it reaches their cumulative probability. If we set top_p to 0.1, it will consider tokens until it reaches that value. If we set top_p to 1, it will consider all tokens.

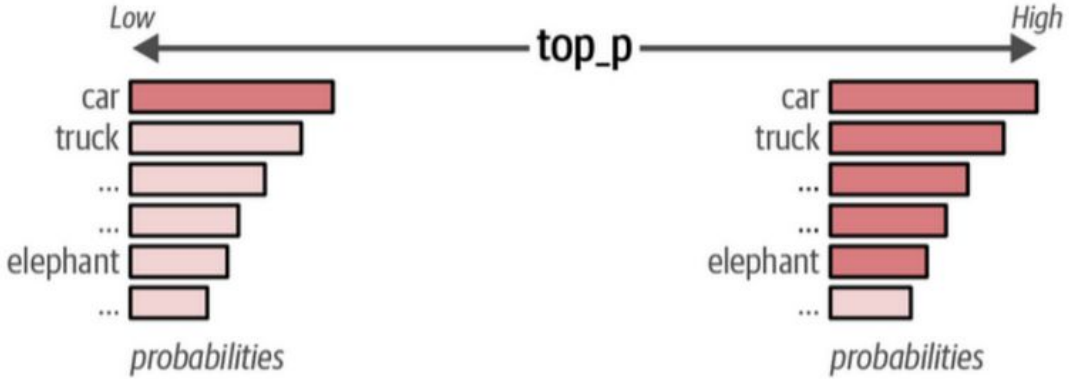

Figure 6-5. A higher `top_p` increases the number of tokens that can be selected to generate and vice versa.

Similarly, the top_k parameter controls exactly how many tokens the LLM can consider. If you change its value to 100, the LLM will only consider the top 100 most probable tokens.

Use-cases

Table 6-1. Use case examples when selecting values for temperature and top_p

Example Use Case	Temperature	top_p	Description
Brainstorming Session	High	High	High randomness with large pool of potential tokens. Results are highly diverse, often creative and unexpected.
Email Generation	Low	Low	Deterministic output with high probable predicted tokens. Produces predictable, focused, and conservative outputs.
Creative Writing	High	Low	High randomness with a small pool of potential tokens. Creative outputs that remain coherent.
Translation	Low	High	Deterministic output with high probable predicted tokens. Produces coherent output with wider vocabulary variety.

INTRODUCTION TO A PROMPT

- An essential part of working with text-generative LLMs is prompt engineering. By carefully designing our prompts we can guide the LLM to generate desired responses.
- It can be used as a tool to evaluate the output of a model as well as to design safeguards and safety mitigation methods.

The Basic Ingredients of a Prompt

An LLM is a prediction machine. Based on a certain input, the prompt, it tries to predict the words that might follow it.

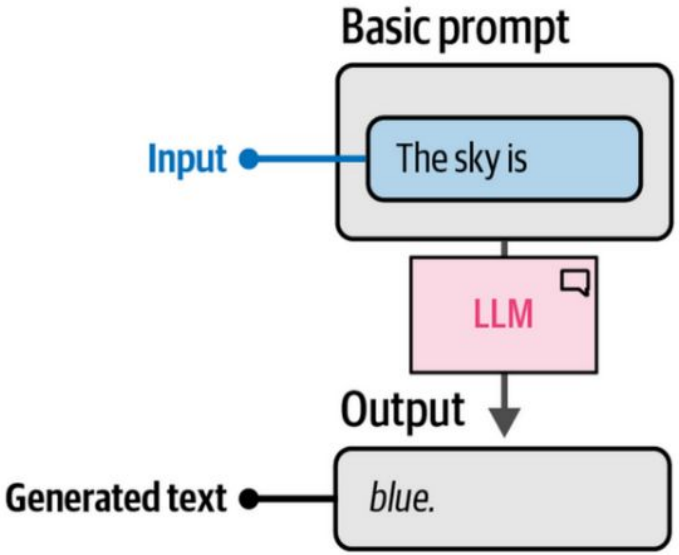

Figure 6-6. A basic example of a prompt. No instruction is given so the LLM will simply try to complete the sentence.

INTRODUCTION TO A PROMPT

Instruction Prompts

To elicit the desired response, we need a more structured prompt. For example, and as shown in Figure 6-7, we could ask the LLM to classify a sentence into either having positive or negative sentiment. This extends the most basic prompt to one consisting of two components—the instruction itself and the data that relates to the instruction.

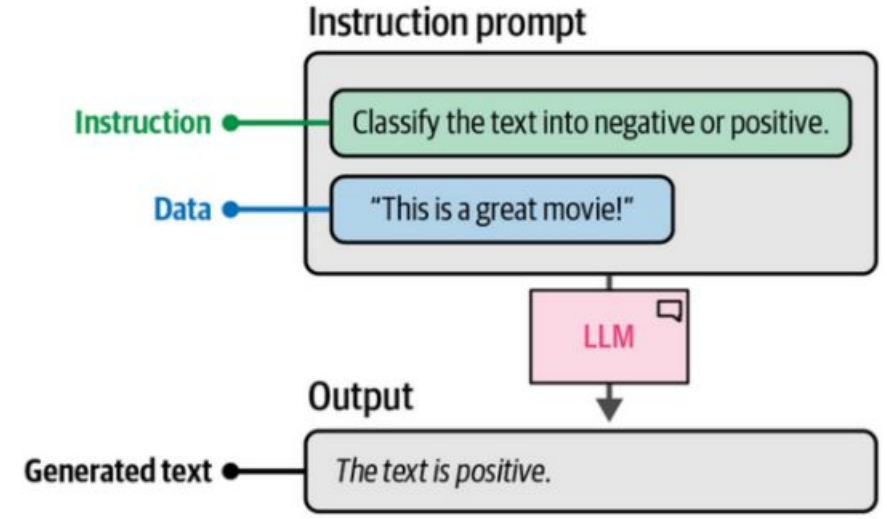

Figure 6-7. Two components of a basic instruction prompt: the instruction itself and the data it refers to

INTRODUCTION TO A PROMPT

Prompt with an output indicator

In Figure 6-8, we prefix the sentence with “Text:” and add “Sentiment:” to prevent the model from generating a complete sentence. Instead, this structure indicates that we expect either “negative” or “positive.” Although the model might not have been trained on these components directly, it was fed enough instructions to be able to generalize to this structure.

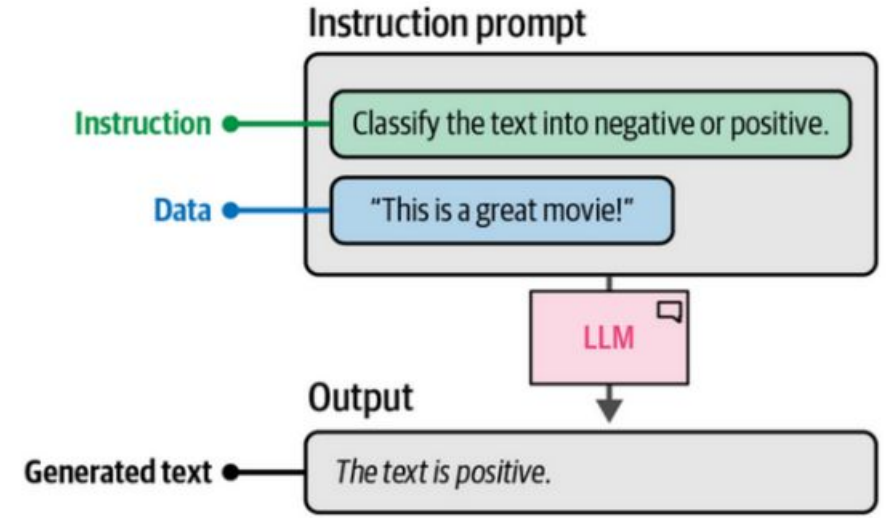

Figure 6-7. Two components of a basic instruction prompt: the instruction itself and the data it refers to

Instruction based Prompting

prompting is often used to have the LLM answer a specific question or resolve a certain task. This is referred to as **instruction-based prompting.**

Figure 6-9 illustrates a number of use cases in which instruction-based prompting plays an important role. We already did one of these in the previous example, namely supervised classification.

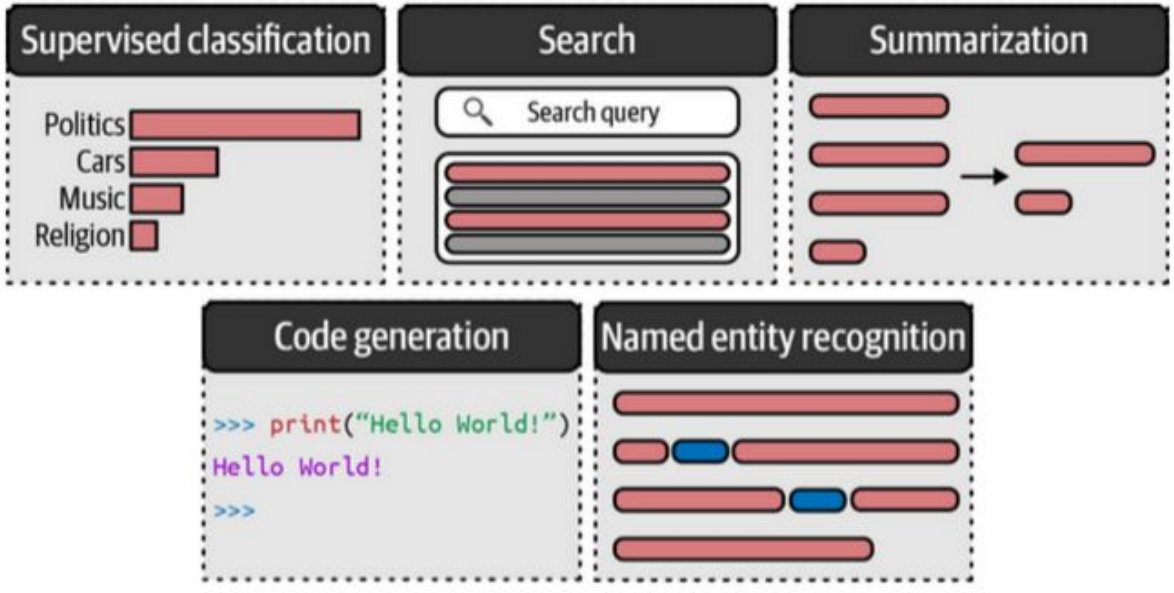

Figure 6-9. Use cases for instruction-based prompting.

Advanced Prompt Engineering - The Potential Complexity of a Prompt

No prompt is limited to just these three components and you can build it up to be as complex as you want.

Some common components are:

- Persona - Describe what role the LLM should take on. For example, use “You are an expert in astrophysics” if you want to ask a question about astrophysics.
- Instruction - The task itself. Make sure this is as specific as possible. We do not want to leave much room for interpretation.
- Context - Additional information describing the context of the problem or task. It answers questions like “What is the reason for the instruction?”
- Format - The format the LLM should use to output the generated text. Without it, the LLM will come up with a format itself, which is troublesome in
- Audience - The target of the generated text. This also describes the level of the generated output. For education purposes, it is often helpful to use ELI5 (“Explain it like I’m 5”).
- Tone - The tone of voice the LLM should use in the generated text. If you are writing a formal email to your boss, you might not want to use an informal tone of voice.
- Data - The main data related to the task itself.

Updating the basic classification prompt

As illustrated in Figure 6-12, we can slowly build up our prompt and explore the effect of each change.

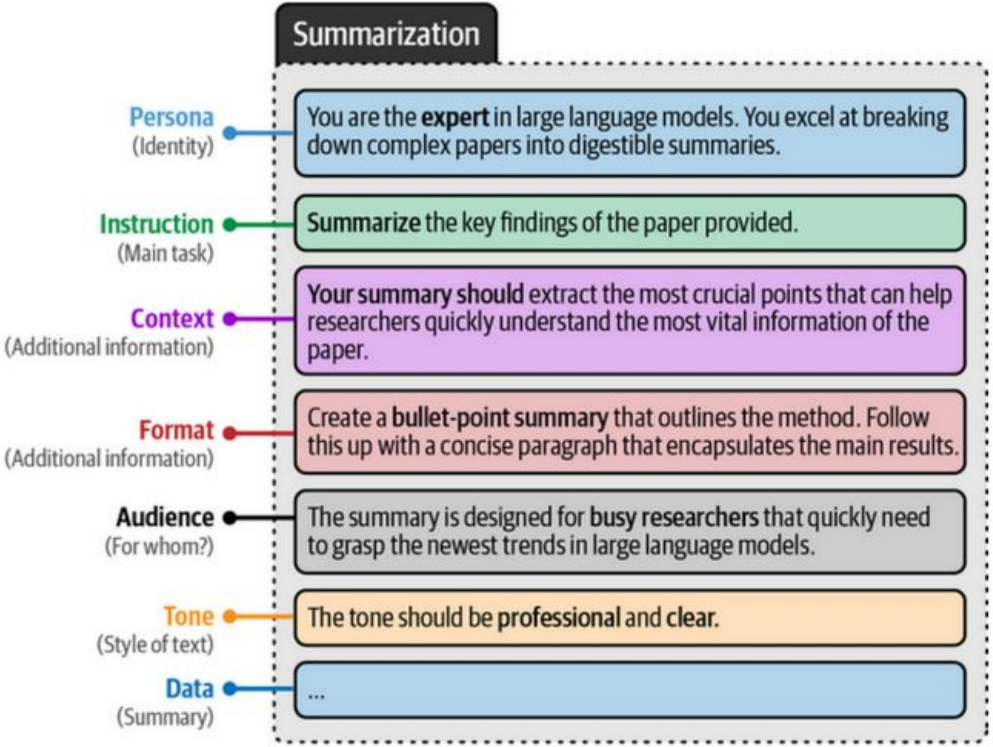

Figure 6-11. An example of a complex prompt with many components.

In-Context Learning: Providing Examples

Instead of describing the task, why do we not just show the task?

We can provide the LLM with examples of exactly the thing that we want to achieve. This is often referred to as in-context learning, where we provide the model with correct examples.3

As illustrated in Figure 6-13, this comes in a number of forms depending on how many examples you show the LLM. Zero-shot prompting does not leverage examples, one-shot prompts use a single example, and few-shot prompts use two or more examples.

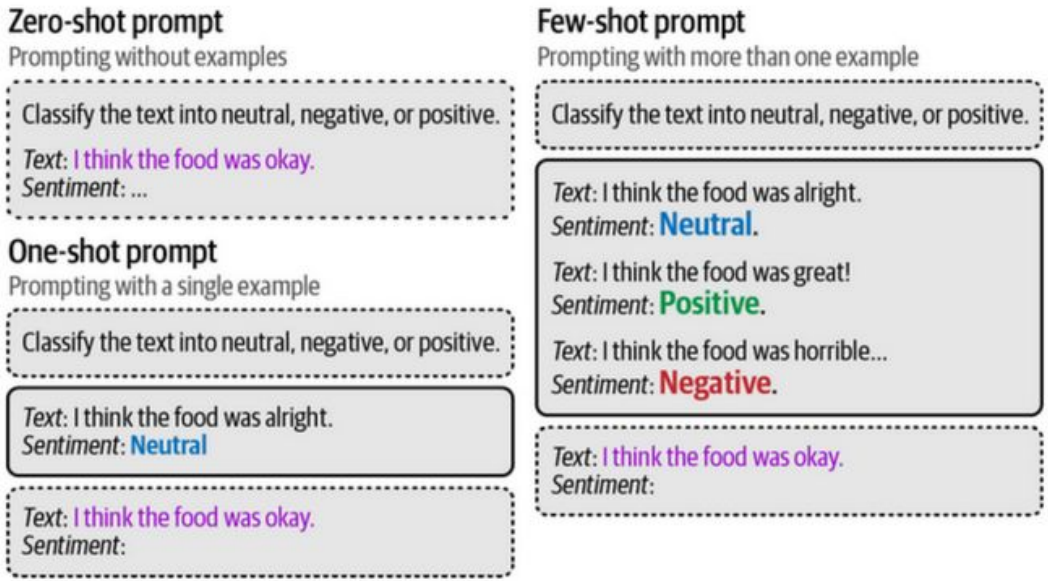

Figure 6-13. An example of a complex prompt with many components.

Chain Prompting: Breaking up the Problem

- In previous examples, we explored splitting up prompts into modular components to improve the performance of LLMs. Although this works well for many use cases, this might not be feasible for highly complex prompts or use cases.
- Essentially, we take the output of one prompt and use it as input for the next, thereby creating a continuous chain of interactions that solves our problem.

Example - Chain Prompting

let us say we want to use an LLM to create a product name, slogan, and sales pitch for us based on a number of product features.Although we can ask the LLM to do this in one go, we can instead break up the problem into pieces.

In Figure 6-14, we get a sequential pipeline that first creates the product name, uses that with the product features as input to create the slogan, and finally, uses the features, product name, and slogan to create the sales pitch.

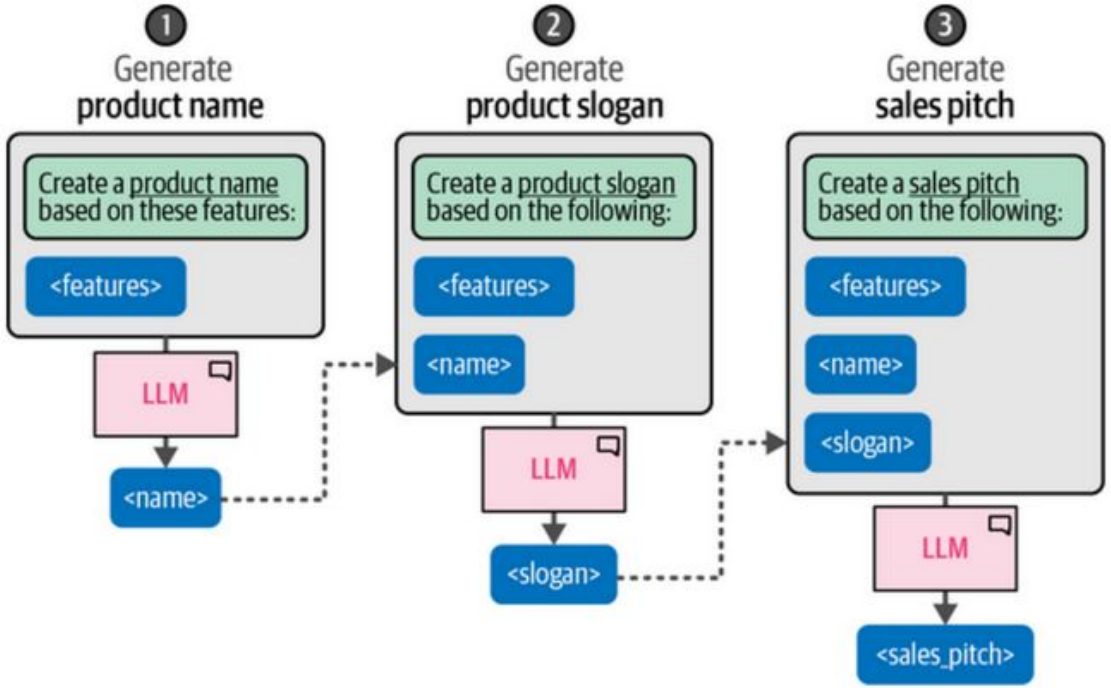

Figure 6-14. Using a description of a product's features, chain prompts to create a suitable name, slogan, and sales pitch.

Reasoning based Generative LLMs

To simplify, our methods of reasoning can be divided into system 1 and 2 thinking processes.

- System 1 - thinking represents an automatic, intuitive, and near-instantaneous process. It shares similarities with generative models that automatically generate tokens without any self-reflective behavior. In contrast,
- System 2 - thinking is a conscious, slow, and logical process, akin to brainstorming and Self-reflection.

If we could give a generative model the ability to mimic a form of self- reflection, we would essentially be emulating the system 2 way of thinking,

Chain-of-Thought: Think Before Answering

As illustrated in Figure 6-15, it provides examples in a prompt that demonstrate the reasoning the model should do before generating its response. These reasoning processes are referred to as “thoughts.” This helps tremendously for tasks that involve a higher degree of complexity,
like mathematical questions. Adding this reasoning step allows the model to distribute more compute over the reasoning process.

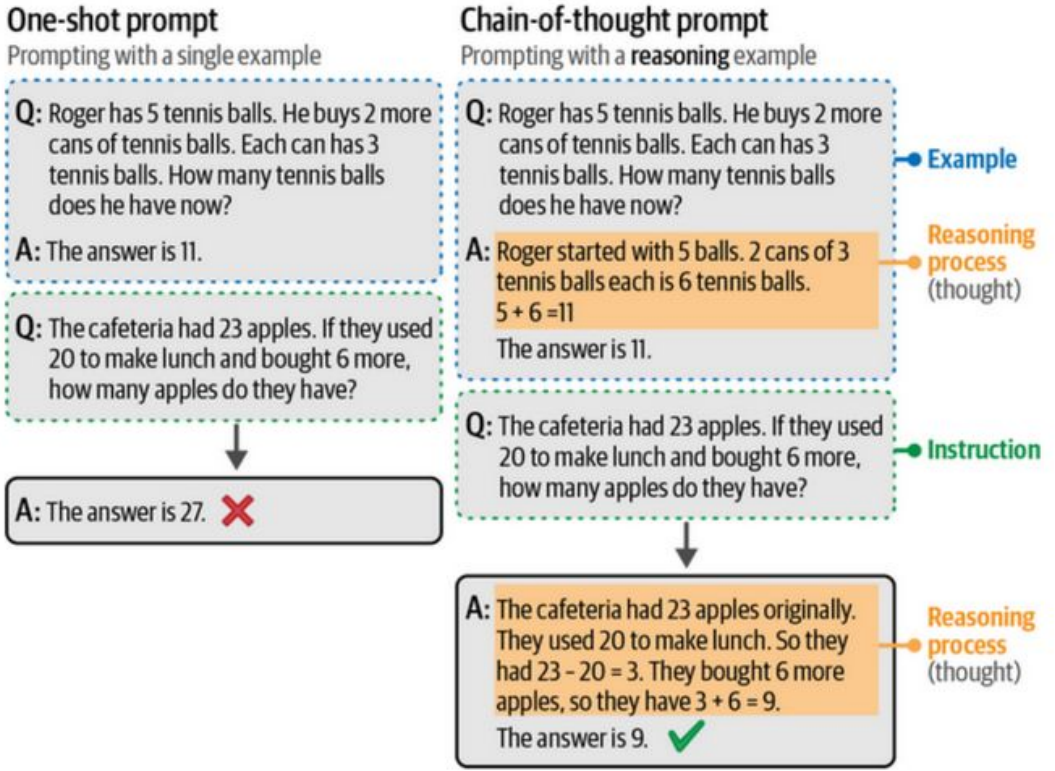

Figure 6-15. Chain-of-thought prompting uses reasoning examples to persuade the generative model to use reasoning in its answer.

Chain of Thoughts using Zero-shot Prompting

Instead of providing examples, we can simply ask the generative model to provide the reasoning (zero-shot chain-of-thought). There are many different forms that work but a common and effective method is to use the phrase “Let’s think step-by- step,” which is illustrated in Figure 6-16.

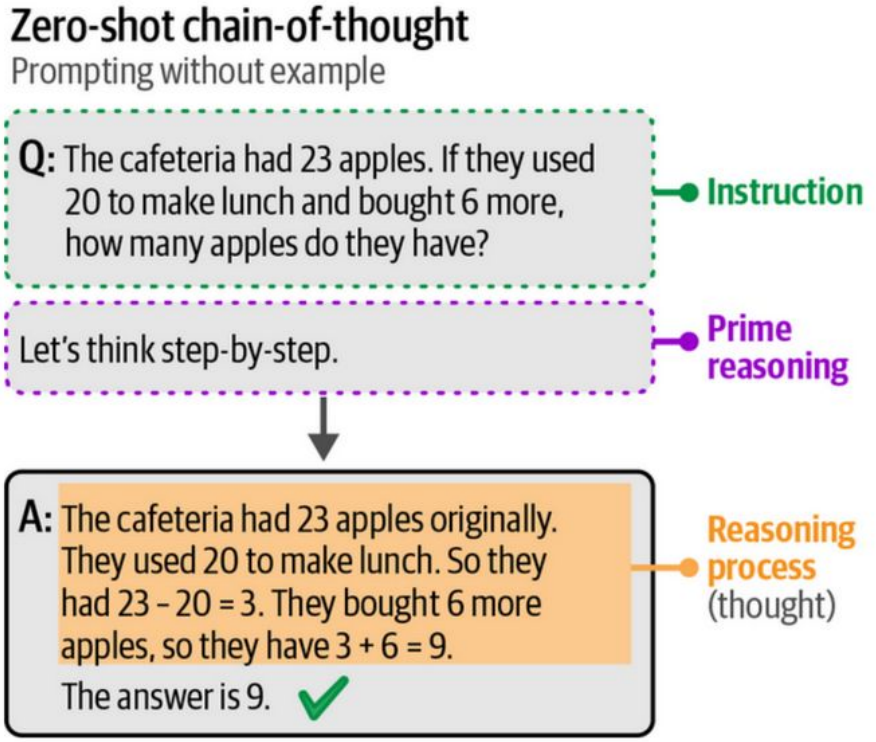

Figure 6-16. Chain-of-thought prompting without using examples. Instead, it uses the phrase “Let’s think step-by-step” to prime reasoning in its answer.