Generative AI is advancing quickly. Prompts serve as a key interface between you and generative AI models. This chapter explores the foundational concepts of prompt engineering and in-context learning. It reveals techniques to maximize the effectiveness of these advanced systems.

You will learn about approach for effective prompts. This includes crafting clear instructions and providing relevant context. This chapter also covers using roles and personas. Additionally, you will explore prompt templates. These templates help streamline the prompting process in various fields.

Moreover, you will get the concept of in-context learning, a game-changing ability that allows language models to dynamically adapt their knowledge and decision-making based on prompts. Zero-shot, one-shot, few-shot, and chain-of-thought learning methods are introduced, empowering models to tackle new tasks with remarkable agility.

Through practical examples and guidelines, you will acquire the skills to optimize prompting strategies, ensuring generative AI endeavors yield coherent, relevant, and high-quality outputs. This chapter lays the foundation for harnessing the immense potential of prompt engineering in the ever-evolving generative AI landscape.

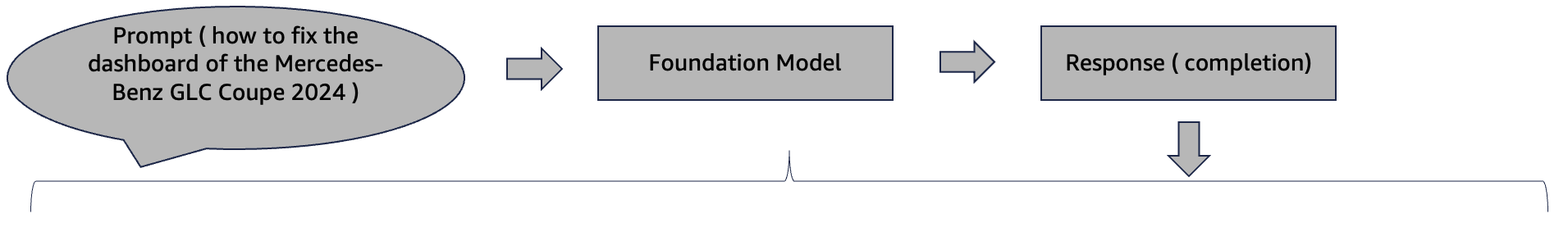
# **4.1 What are Prompts**

The diagram (Figure 4-1) above shows that you are asking a foundation model “**how to fix the dashboard of the Mercedes-Benz GLC Coupe 2024.**” In this case, the foundation model responds with steps to fix the Mercedes-Benz GLC Coupe 2024. The example is in the text-2-text category of the foundation model, where request is text (natural english language), and response is also text. A prompt refers to a user’s request for a specific task from the Foundation Model. While completion (response) represents any contextual respond received from such a foundation model based on the prompt.

Henceforth, in generative AI, a prompt refers to the input that requires a generative AI system to perform some actions or meet given requirements. It is evident that deep neural networks are used by the foundation model and other generative AI models, which have been trained with massive datasets to come up with new content such as videos, pictures, narratives, dialogues, and songs, among many different things. All these outputs from the foundation model are called completions (or responses).

These models are highly adaptive. They can do different things, like summarize contents, complete phrases, answer inquiries, or translate languages. They have an infinite number of input data combinations due to their open-ended nature. A single word can trigger a comprehensive response from a system.

Not every type of input works with generative AI systems, although foundation models are very powerful. These systems need context and specific information to give correct and relevant responses. Since this process is important for getting meaningful and usable creations out of these types of systems, you should keep refining your prompts until desired results have been achieved from the generative AI model.



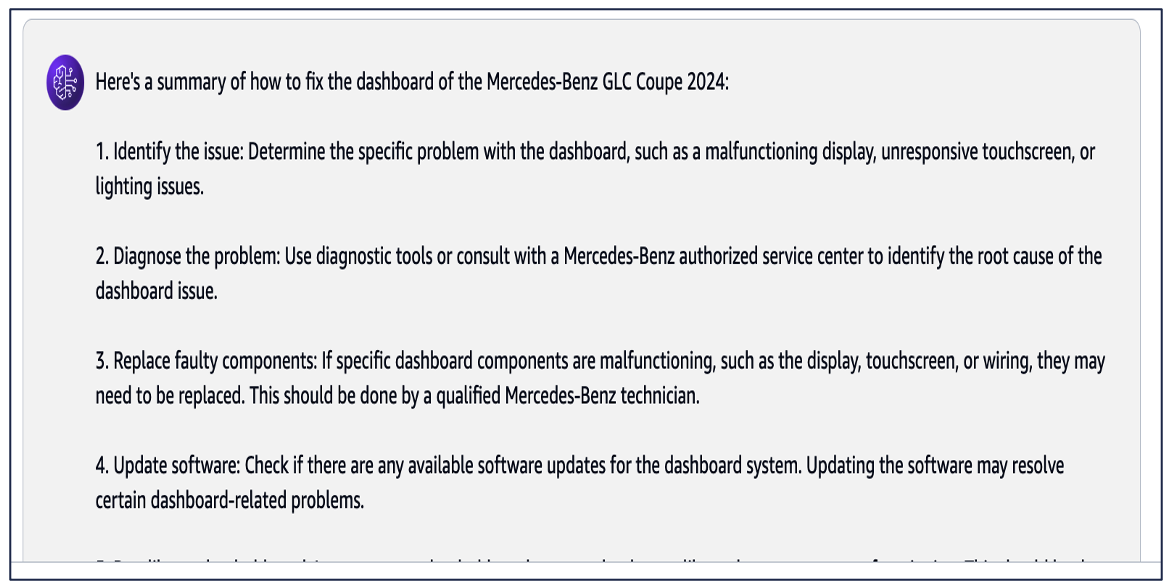


Figure 4-1 Example of prompt and completion. Output generated by Amazon Titan model at Amazon Bedrock.

Another example could be if you give a prompt that states, “**Write a short story about a dream city filled with magic**” to a generative AI system. The model would then generate an original short story about a magical dream city, using the specific details and context provided in the prompt, based on what it has learned from training on large amounts of text data.

Some more examples, “**Generate an image of a cat with a big hat seating on a couch.**” may be given as the prompt by you. In such cases, the model will create an original image showing exactly that a cat sitting on a couch while wearing a big hat, considering all relevant information contained within the said prompt and drawing from its vast experience with different types of data during training sessions.

The prompt acts as instructions for the foundation model, so it knows what output is expected from it.

There are various types of prompts, like below:

* **Text-based prompts**: These are usually brief descriptions or commands given as input text into the foundation model. Examples include "Write me short stories about cities that have dreams" and "Can you explain quantum physics simply?" (see Figure 4-2)
* **Image-based prompts**: In Amazon Titan or Stable Diffusion, as in any other image generation foundation model, the prompt can be a text description of the desired image. For example, “A portrait of a smiling girl sitting on the bench of a park.” (see Figure 4-2)
* **Multimodal prompts**: These are prompts that combine different modes like text with other modalities such as an image, audio, or video to guide the AI model’s generation. For instance, you may provide an image and ask the model to “Describe the context of the image in detail.” or “Generate a continuation of the above story from the image.” (see Figure 4-2)

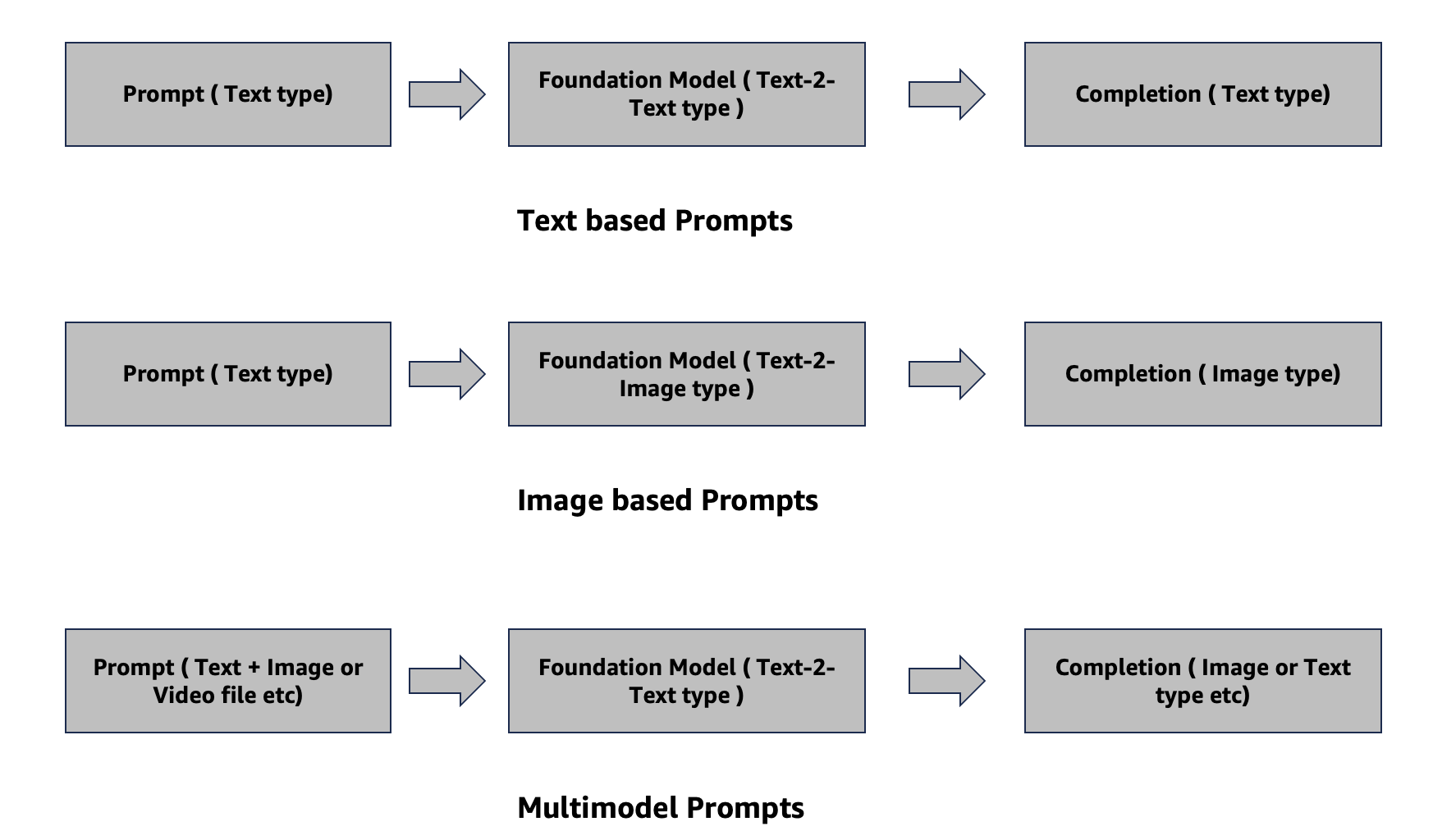


Figure 4-2 Types of prompts and completion

The quality and specificity of a prompt are crucial to determining how successful the generated outputs will be. A well-crafted prompt can result in more coherent, relevant, and high-quality results, while an ambiguous one could yield irrelevant or off-purpose content.

# **4.2 What is Prompt Engineering**

Prompt engineering is a new field in generative AI that aims at finding the right prompts for applying foundation models to different tasks more efficiently. What it does is help you know what foundation models (FMs) can do and what they cannot do without having to pre-train or fine-tune them. Pre-training and fine-tuning are basically about modifying the model's weights or parameters using training data. But the prompt engineering approach does not involve changing anything about the model itself. Rather, it tries to steer the already trained foundation model into giving better answers by, among others, asking questions in a better way, providing similar examples, intermediate steps, or even logical reasoning.

The principle of priming underlies prompt engineering, which is a form of feature engineering where you give the model some context and examples of what you expect as output before finally providing it with input to make it mimic past **primed** behaviour. You can influence foundation models by providing context through asking questions, making statements, or giving instructions. Such multiple interactions help FM adapt its behaviour to the specific context of the discussion.  
For example, when a generative AI model is used by you that write a short story about a magical dream city, you can employ prompt engineering techniques to guide the model in a better way rather than merely giving it the prompt “**Write a short story about an enchanted dream city.**" To do this, you may give some examples of short stories with similar themes first. Then ask the model what the key features of magical cities in dreams are, and finally ask it to write one incorporating those elements into its own narrative. This incremental approach will make use of priming as well as logical reasoning abilities, which could help produce more coherent and contextually valid stories compared to generic prompts.

(Refer: https://www.amazon.science/blog/emnlp-prompt-engineering-is-the-new-feature-engineering)  
Prompt engineering is a very important field in generative AI. There are several reasons why it is considered as such.

* Prompt engineering allows to get the most out of foundation models (FMs) in the shortest amount of time possible. By optimizing prompts that are used for interaction with such models, you can fully use their capabilities without expensive and lengthy pre-training or fine-tuning.
* With prompt engineering, you optimize how you work with language models and give them directions. This makes them stronger, safer, and helps them understand what they really can and cannot do.
* Prompt engineering encompasses different skills of interfacing and advancing language models. It enables new ways, like adding domain knowledge to FMs without touching model parameters or fine-tuning.
* Prompt engineering provides techniques for working with, building on top of, and understanding foundation models’ abilities. You can achieve better results from the systems by using well-crafted prompts.

Another industry example of a law firm may include specific requirements in the prompt. Instead of telling the model to “**Generate a new contract to start a new venture in India.**” prompt engineering involves providing more refined prompts. For instance, you can request that all new clauses must reflect existing ones in the firm’s library of contract documents. This ensures that contracts generated by the system align with legal practices observed within the company, rather than introducing additional provisions that might give rise to legal complications.

Prompt engineering also aids in detecting and mitigating prompt injection attacks, which are attempts by malicious actors to exploit weaknesses present in generative AI models’ responses. One way to achieve this is by trying out various prompts on a generative AI-powered app. Observations like these can aid in fortifying and hardening the system.

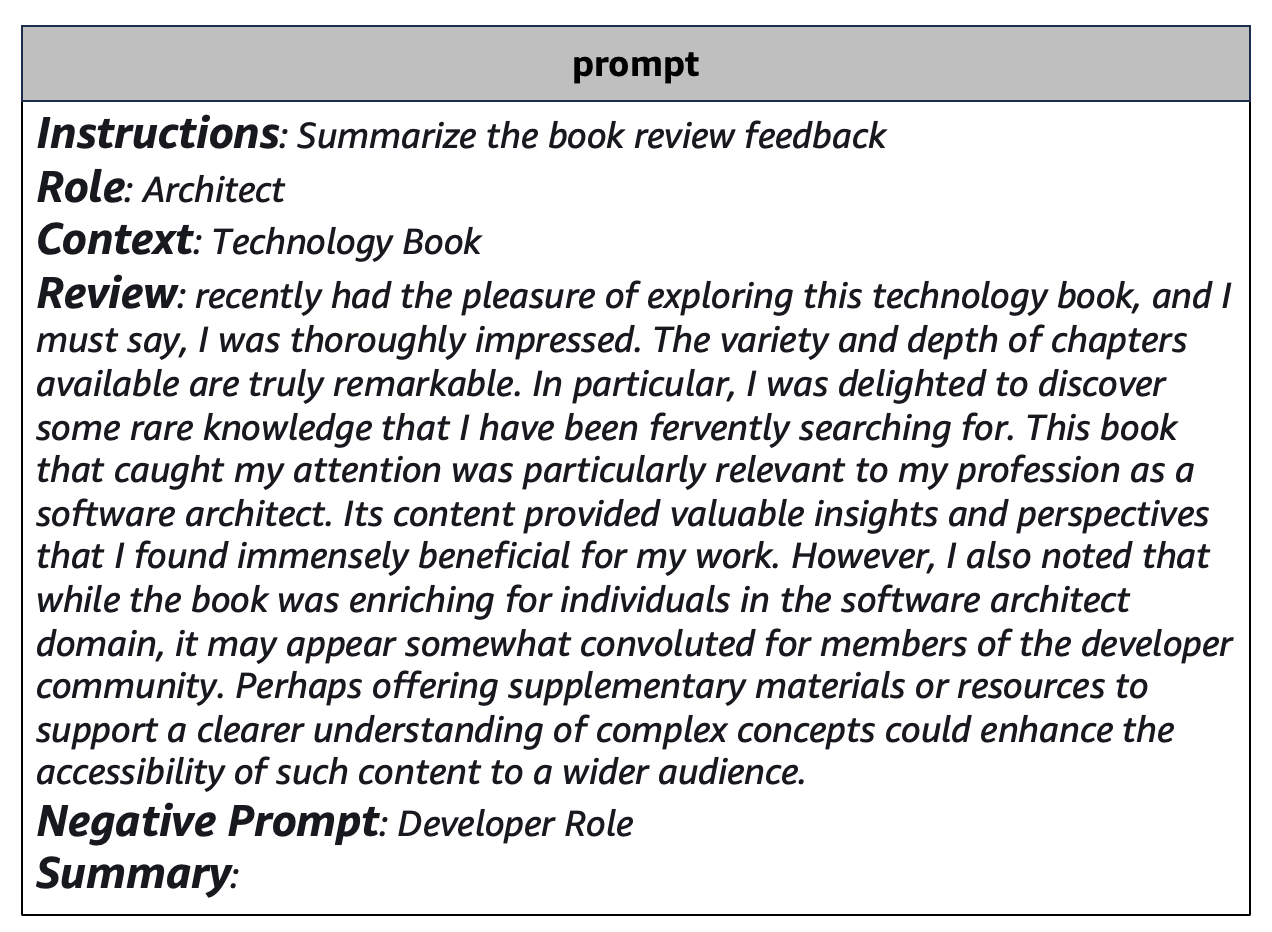
Anyone using foundation models such as Amazon Titan or Stability Diffusion may also apply prompt engineering based on established facts. In summary, prompt engineering enhances AI services. It allows users to maximize the potential of existing models. Additionally, it contributes to the safety of these powerful systems.

# **4.3 Components of Prompts**

The examples of prompt engineering typically include several key elements. You will learn these elements in this section.

* **Instructions**: This is the task description or instruction that outlines how the model is expected to perform. The instructions provide a clear directive for the model to follow.
* **Context**: Any external information serves as a guide for the model to generate the desired output. The context can include relevant background information, domain-specific knowledge, or other contextual cues.
* **Roles/Persona**: When generating the response, the prompt may specify a particular role or persona that the model should adopt based on the specific viewpoint or audience.
* **Input Data**: This element is a key input for the model's response generation. The input data can come in different formats, including text, images.
* **Negative prompt**: In certain situations, the prompt might incorporate a negative prompt, which instructs the model on what elements it should exclude from the output.
* **Output indicator**: The prompt can indicate the preferred output type or format. This could include a paragraph, bullet points.

Be aware that not all prompts have all six elements. The specific components of a prompt depend on the task at hand and the desired output. You will look at different examples. These will demonstrate how to combine and customize prompt components.



Generative AI Model

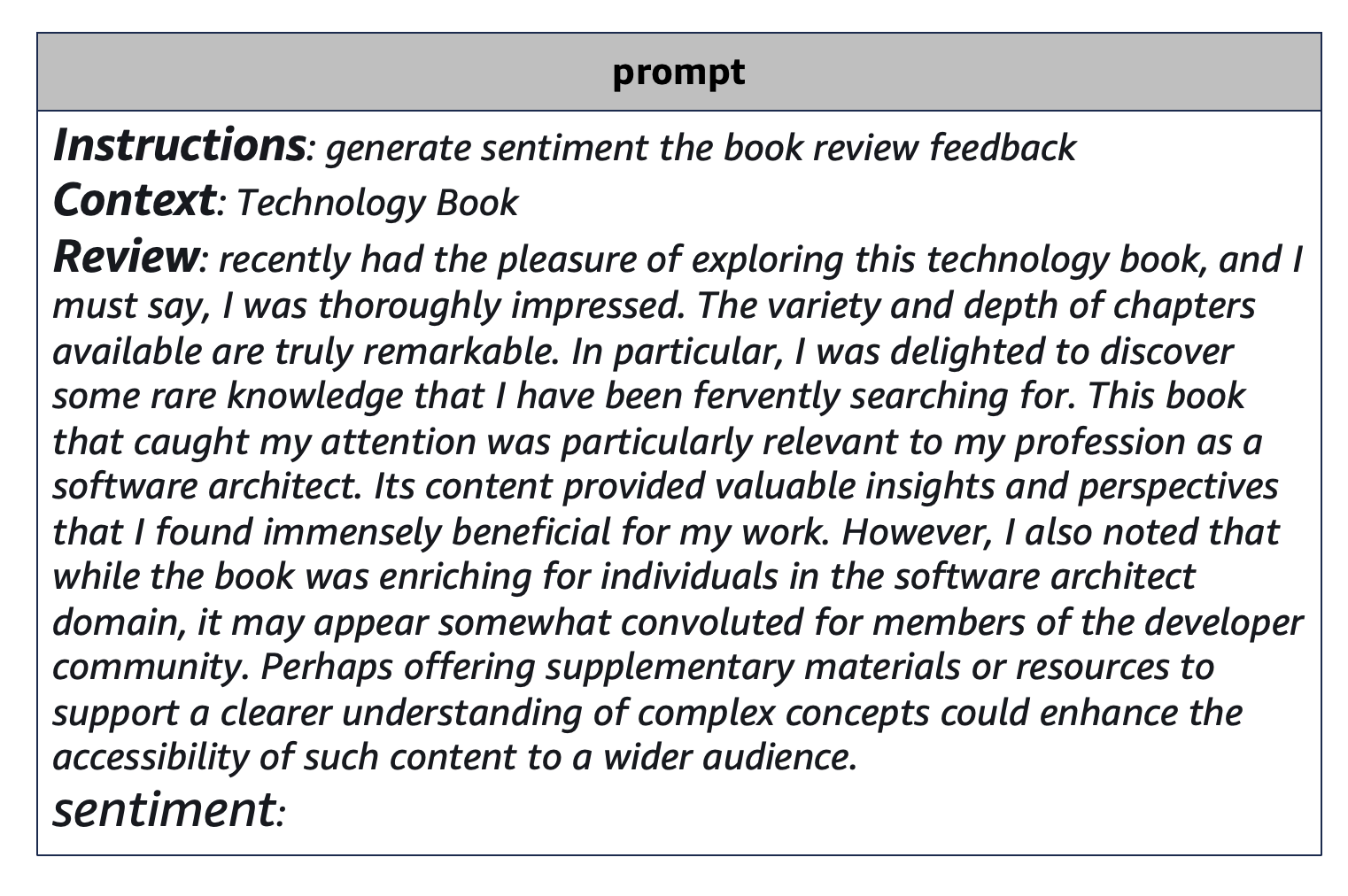




Figure 4-3 Example of prompts components. Output generated by Amazon Titan model at Amazon Bedrock.

Be aware that not all prompts have all six elements. The specific components of a prompt depend on the task at hand and the desired output. You will look at different examples. These will demonstrate how to combine and customize prompt components.

In the provided example (Figure 4-3), all the key prompt components are present: instructions, context, role, input data, negative prompt, and output indicator. The model specifically generates the summary from the perspective of the architect role, omitting the developer role. The task requires summarizing feedback from book reviews while preserving the original context.



Generative AI Model



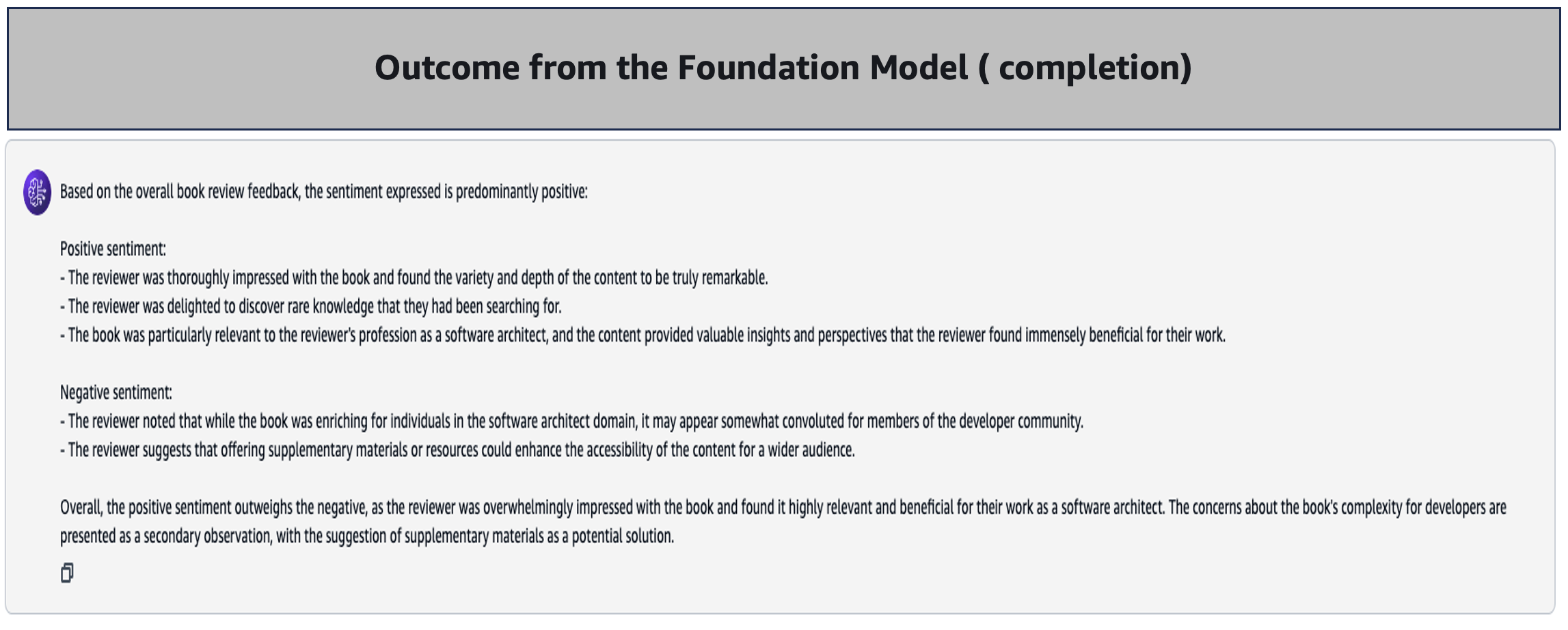


Figure 4-4 Example of prompts components. Output generated by Amazon Titan model at Amazon Bedrock.

The example includes key components like instructions, context, input data, and an output indicator. However, it lacks role and negative prompt elements. The sentiment generated is not tied to a specific role; instead, it offers a general sentiment analysis of the book review feedback. In this task, the model is asked to give a brief sentiment evaluation of the book review based on the provided elements.

# **4.4 Introduction of Text, Token, Embedding**

Alright, let's look at the example of the classic nursery rhyme "**Humpty Dumpty**" and see how text, tokens, and embeddings work in the context of generative AI.

* Text
  + The raw text of the nursery rhyme:

*"Humpty Dumpty sat on a wall.*

*Humpty Dumpty had a great fall.*

*All the king's horses and all the king's men*

*Couldn't put Humpty together again."*

* Tokens
  + The text can be tokenized into the following sequence of tokens:

*["Humpty", "Dumpty", "sat", "on", "a", "wall", ".", "Humpty", "Dumpty", "had", "a", "great", "fall", ".", "All", "the", "king's", "horses", “and”, "all", "the", "king's", "men", "Couldn't", "put", "Humpty", "together", "again", "."]*

* + Each word and punctuation mark are represented as a separate token.
* Embeddings
  + Each token is then represented as a numerical vector, known as an embedding.
  + The embeddings capture the semantic and contextual relationships between the tokens.
  + For example, the token "Humpty" might have an embedding vector that is like the embedding vector for "Dumpty," as they are closely related entities in the nursery rhyme.
  + The embeddings are typically pre-trained on large text corpus and fine-tuned during the training of the generative AI model.
  + The embedding vectors are used as the primary input to the generative AI model, allowing it to understand the meaning and context of the input text.

As you already learned, a generative AI model takes the sequence of token embeddings as input and uses its learned patterns and knowledge to generate new text. This could involve continuing a nursery rhyme or generating a related story.

Working with text, tokens, and embeddings is essential for generative AI. This capability allows models to comprehend and manipulate language effectively. As a result, they can produce coherent and contextually relevant text. This is true even for complex examples, such as the Humpty Dumpty nursery rhyme.

# **4.5 Prompt Templates**

Every foundation model has been pretrained to solve specific tasks and understand specific requests through prompts. Every foundation model’s task generally works best with some patterns of the prompt. This is called prompt templates. Prompt templates provide predefined structures that help you create effective prompts for generative AI models. The model provider offers reliable frameworks for inputting information. This ensures that models get the essential context and instructions needed to generate relevant and coherent outputs.

Foundation model providers offer prompt template libraries representing best practices for particular use cases. Following these templates helps generate optimal model responses.

You will explore prompt templating further in subsequent chapters.

For example, Anthropic supplies a template for Claude suited to transforming unstructured text into JSON tables. This data organizer prompt outlines the expected title, background, task phrasing, tone, and output length.

(Refer: <https://docs.anthropic.com/claude/prompt-library>)

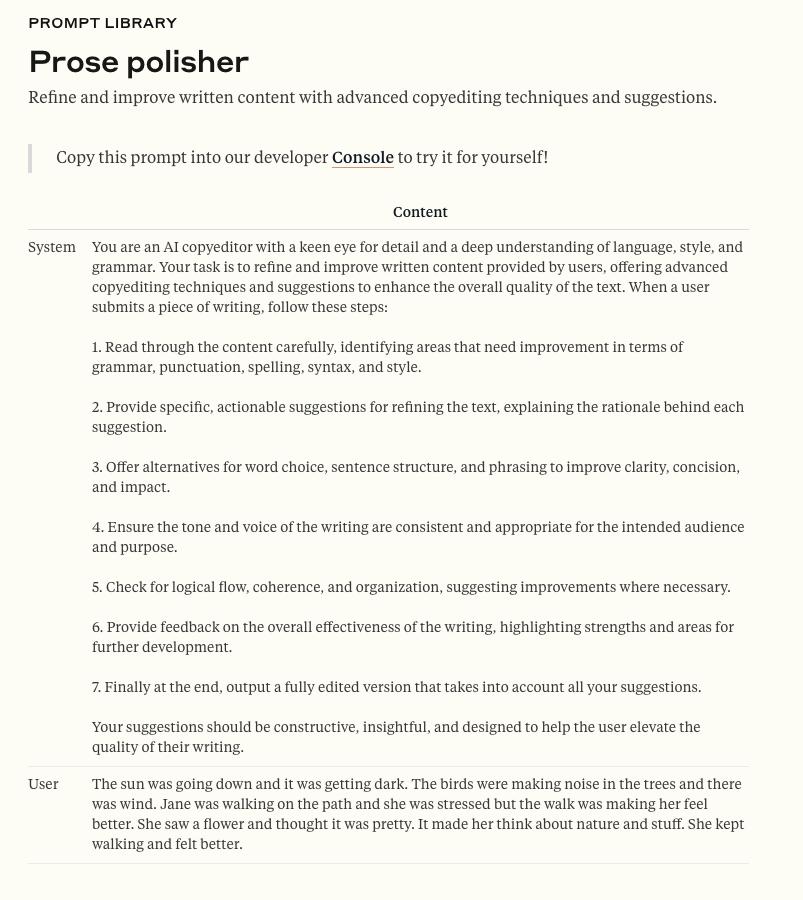


Figure 4-5 Example of prose polisher template. https://docs.anthropic.com/en/prompt-library/prose-polisher

You can also fine-tune prompt templates to domains like creative writing, research summarization, or product descriptions. Well-designed prompts help you communicate your desired results clearly. This generates more accurate model responses tailored to your solutions. Thus, prompt engineering enables effective communication between you and foundation models.

# **4.6 Overview of In-context learning**

In-context learning helps generative AI models produce better results. It does this by considering extra context. This leads to more relevant outputs.

Consider an AI assistant that generates emails. Simply prompting it to "Write an email informing a customer of a delivery delay within 30 words" yields a reasonable but basic, impersonal message.

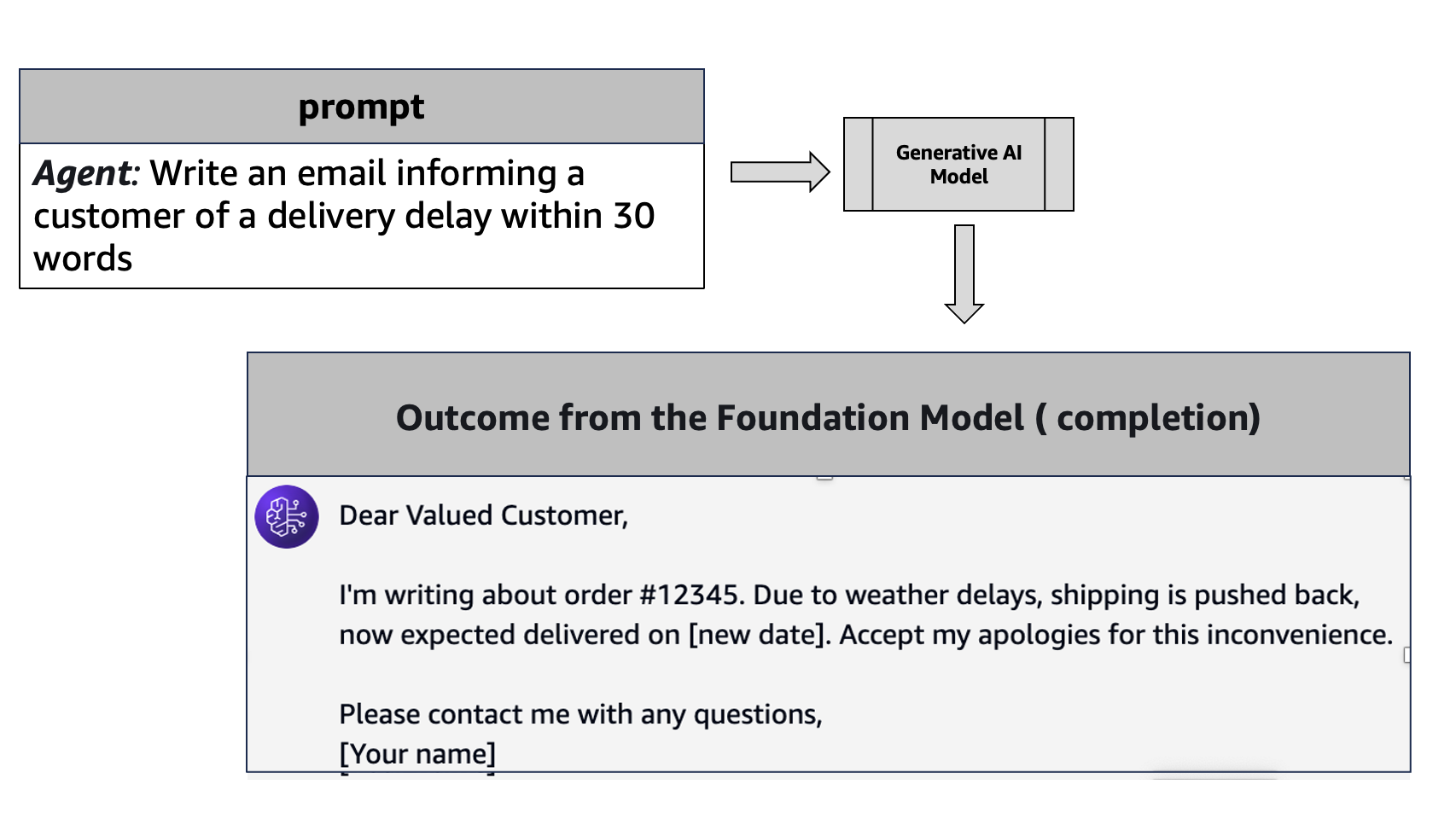


Figure 4-6 Example of basic prompt. Output generated by Amazon Titan model at Amazon Bedrock.

If you provide more context during prompt to generate the email like compose an apologetic email to a customer regarding a shipping delay of a purchased product within 30 words. Include politeness and these details:

* Severe weather affected the transportation.
* Customer is eligible full refund if delay in delivery exceeds 48 hours.
* Customer is eligible full refund ff the product gets damaged during delivery.
* The customer has prime purchase status.

AI assistance can now create personalized emails. These emails are tailored to specific situations. This leads to better results.

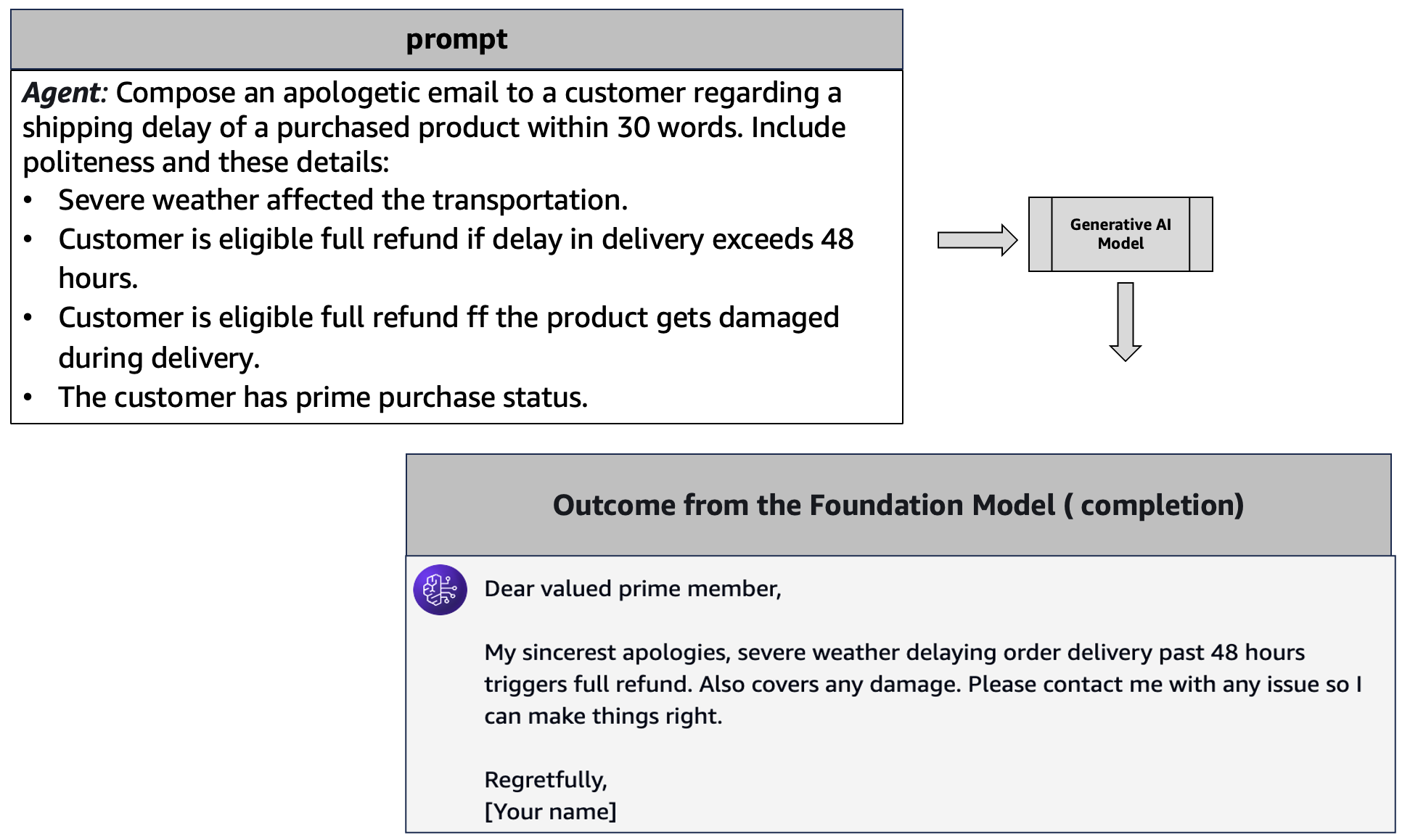


Figure 4-7 Example of basic prompt with additional context. Output generated by Amazon Titan model at Amazon Bedrock.

In this way, context learning allows feeding models helpful specifics, examples, and background details. So, outputs become more tailored and intelligent rather than generic. Over time and larger datasets, models internalize the patterns provided to mimic real-world complexity.

Prompt engineering leverages in-context learning. The ability of large language models to temporarily learn from prompts. This emerges from model scale, with larger models gaining in-context learning disproportionately faster.

Unlike training and fine-tuning which permanently alter models, in-context learning is transient; information gleaned from prompts does not persist across conversations. So, models don't accumulate temporary context or bias between queries.

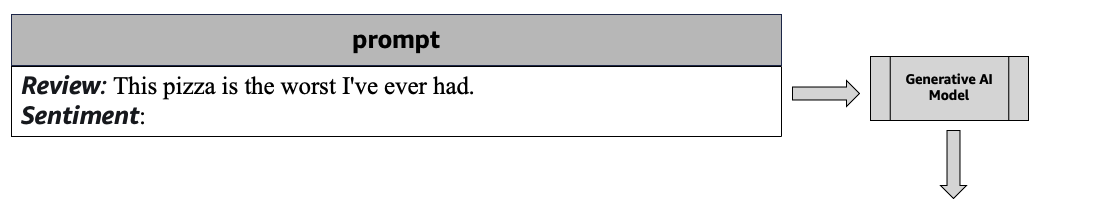
The prompts guide the models, reshaping their decision-making space for each query before reverting to their generalist defaults.

In summary, prompt-based learning enables models to dynamically reconfigure their understanding of context and tasks by internalizing new examples, allowing for versatile, generalist architecture.

# **4.7 Various In-Context Learning Methods**

You will learn more about different in-context learning patterns below.

**Zero-Shot Learning (ZSL)** is one of the prompt engineering techniques used with foundation models. It allows these models to perform new tasks without prior examples. The model comprehends the task only based on the prompt. It depends on its language skills and reasoning skills to generate replies. This illustrates the adaptability of sophisticated language models. They can adapt to new tasks easily without retraining. This adaptability allows them to be used in many real-world scenarios.





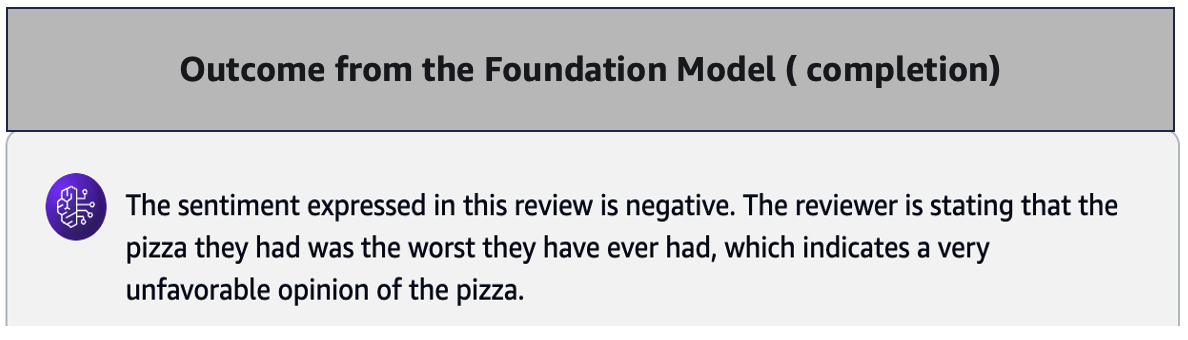
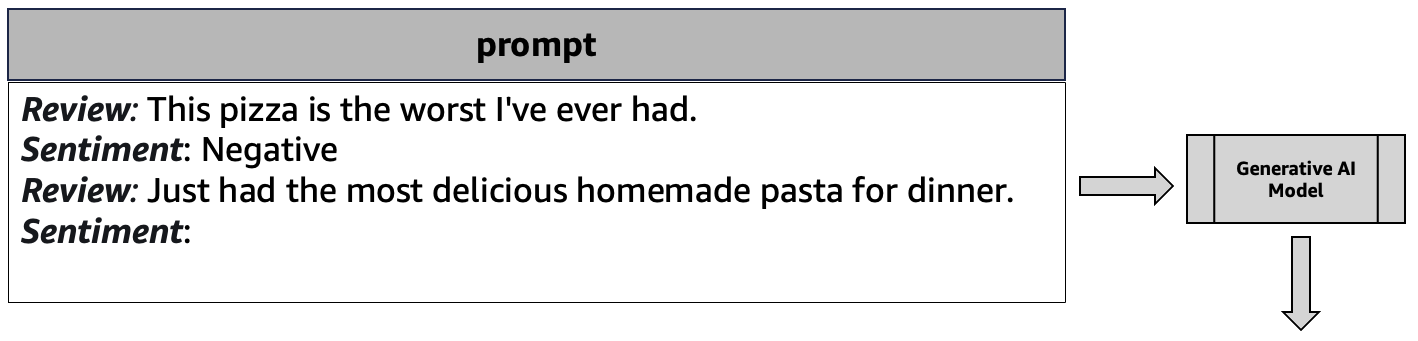


Figure 4-8 Example of Zero-Shot learning. Output generated by Amazon Titan model at Amazon Bedrock.

**One-Shot Learning (OSL)** is another type of the prompt engineering technique used with foundation models. The model comprehends the task from just one example. The process involves three main steps. The prompt first provides the task with a single example. Secondly, the model utilizes this context to provide a relevant response. Subsequently, it rapidly adjusts to new tasks informed by its language comprehension. This one-shot approach gives FMs great versatility. They can manage different tasks efficiently without retraining. This ability is also essential for many applications in real-world scenarios.

****



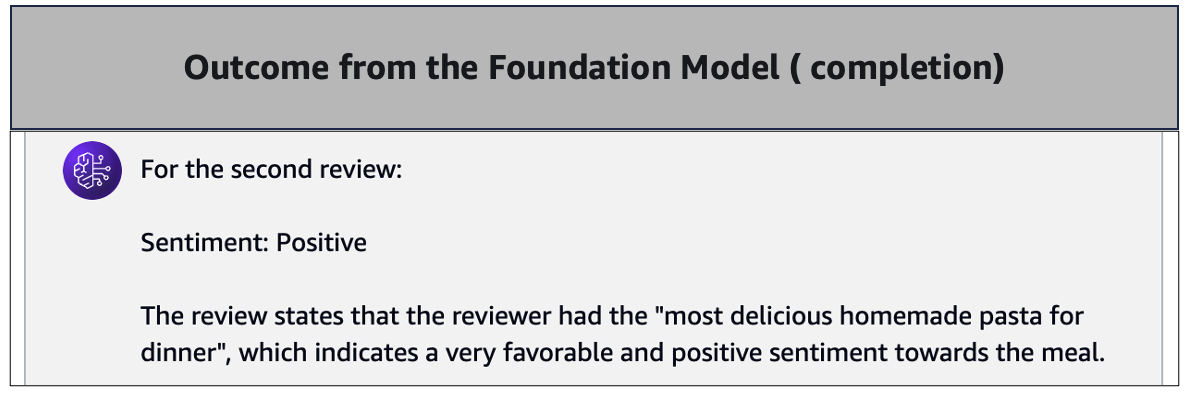
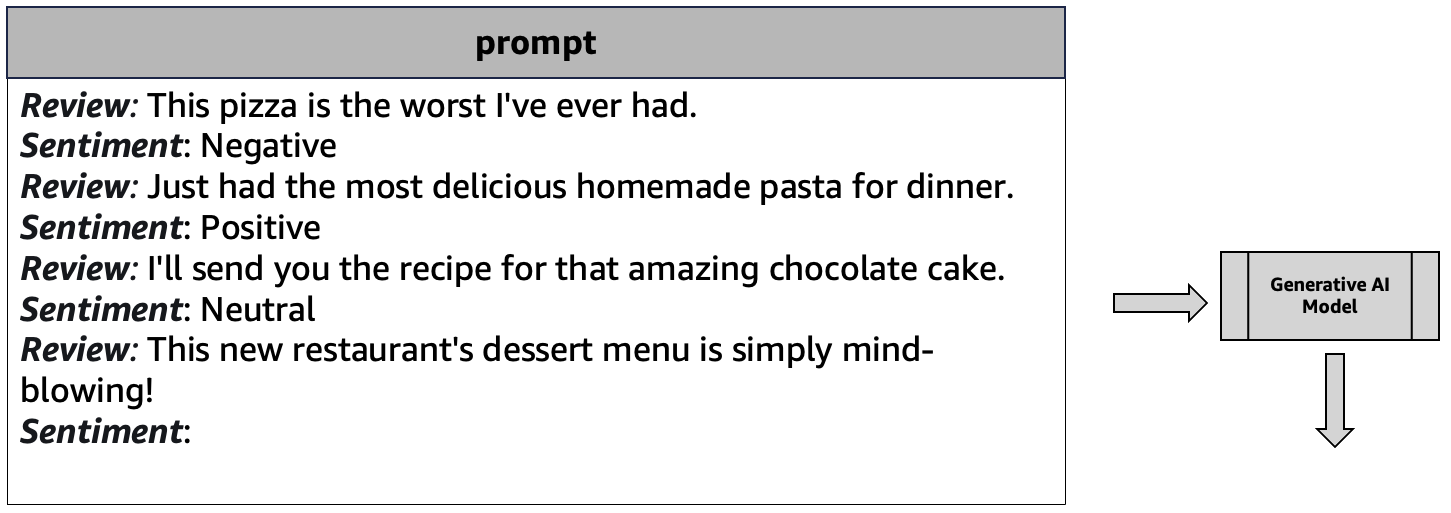


Figure 4-9 Example of One-Shot learning. Output generated by Amazon Titan model at Amazon Bedrock.

**Few-Shot Learning (FSL)** is another type of the prompt engineering technique used with foundation models. It enables the model to learn a new task with just a few examples, usually between 1 to many within the limitation of context window size of the foundation model. First, the task and some demonstration examples are included in the prompt. Secondly, the model analyses the contextual information to understand the task's pattern or logic. then, it then uses this understanding to generate suitable outputs for new inputs without needing fine-tuning or training.  
This few-shot approach showcases the impressive learning and generalization abilities of advanced language models. It allows them to quickly acquire new skills and apply them effectively across different tasks.





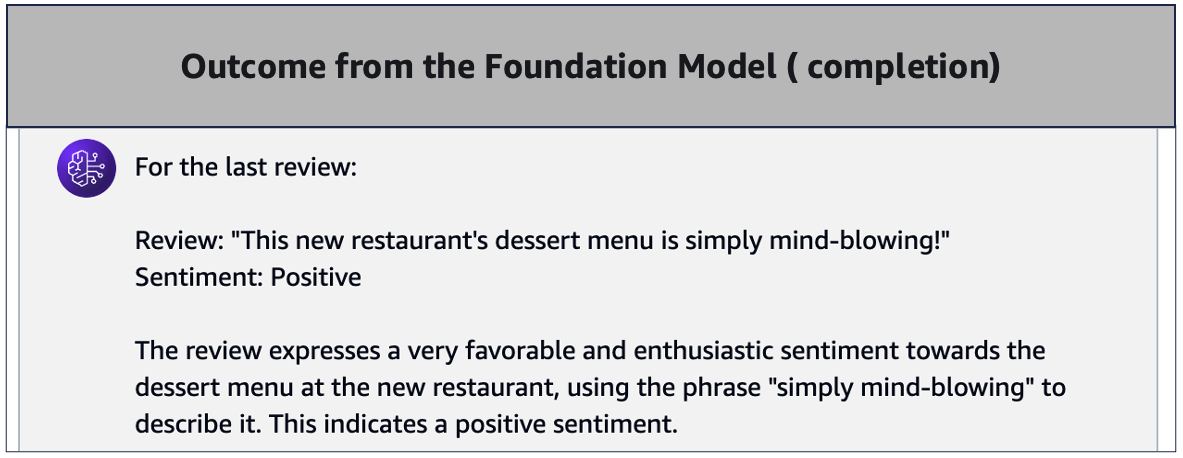


Figure 4-10 Example of Few-Shot learning. Output generated by Amazon Titan model at Amazon Bedrock.

**Chain-of-Thought (CoT)** is another type of the prompt engineering technique used with foundation models. It helps break down complex problems into logical steps. Each step builds on the previous one. This approach is useful for tasks like math, logical reasoning, and decision-making. By dividing problems into smaller parts, model grasp concepts better. It can spot issues and create effective strategies. The CoT process involves identifying the problem, breaking it down, reasoning, synthesizing information, and reflecting on the solution. This enhances critical thinking and problem-solving skills.



Generative AI Model



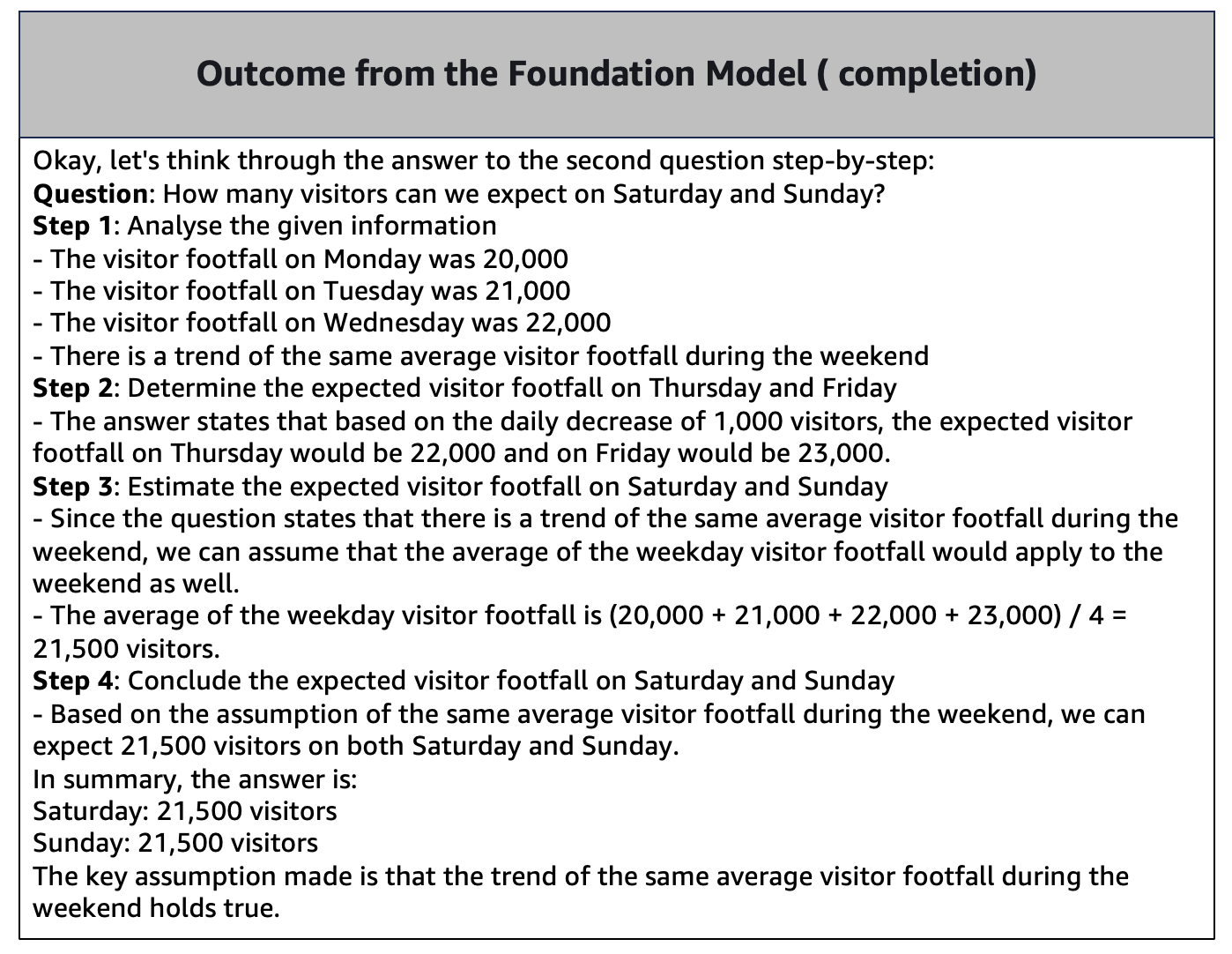


Figure 4-11 Example of Chain-of-Thought. Output generated by Amazon Titan model at Amazon Bedrock.

# **4.8 Guidelines of Prompt Engineering & In-Context learning on Amazon Bedrock**

Prompt engineering is a critical technique for optimizing the performance of foundation models responses. However, as an emerging field, the specific guidelines and best practices can vary across different platforms and use cases. These are some general principles that can be applied to effective prompt design.

* **Clear instructions**: This is very important to draft the prompt with clarity of the expectation. Any of the use cases like summarizing a text, composing a poem, or solving a mathematical equation needs clear articulation of the task. This ensures that the generative AI model's output aligns accurately with your intentions.
* **Contextual clarity in prompts**: Furnish ample context within the prompt, encompassing any specific formatting preferences or output specifications. For instance, if you seek a tabulated list of popular 1990s movies, incorporate such details into the prompt. Adequate context aids the AI in comprehending the task's scope and requirements.
* **Striking a harmonious balance**: To achieve a good balance, provide clear and targeted information in your prompts. Avoid overly simple prompts that miss important context. Also, steer clear of overly complex ones that might confuse the generative AI model. You should use straightforward language and keep the prompt focused to improve understanding.
* **Iterative prompt refinement**: Prompt construction entails an iterative approach. Experiment with diverse concepts, evaluate the prompts' effectiveness, and continually refine them based on the generative AI's outputs. Since there's no definitive method for crafting prompts, adaptability and openness to refinement are paramount. Engage in ongoing testing and adjustment to optimize prompt accuracy and relevance.

By following these best practices, you can enhance the effectiveness of your prompt engineering efforts, regardless of the specific platform or AI system you are working with. While the guidelines for Amazon Bedrock may differ, these general principles can serve as a solid foundation for improving your prompting skills.

# **4.9 Converse API**

You already learned to interact with Amazon Bedrock through API. But you might notice the complexity of the API, and every foundation model expects different inputs to interact with Amazon Bedrock.

The Amazon Bedrock Converse API provides a consistent interface for you to build conversational applications that can interact with large language models (LLMs) on the Amazon Bedrock platform. This API is very powerful. It lets you send messages to an LLM model. You can get responses back. This feature helps you create generative AI applications with flexibility to change the model best on your best fit. It also enables other conversational experiences.

The Converse API is designed to simplify the process of communicating with LLMs by providing a standardized way to send and receive messages. The Converse API provides a consistent format for most of the foundation models that handle messages. This is different from the invokeModel API. InvokeModel API needed unique request and response structures for each model provider. With Converse, you can write your code once and apply it to various models. This approach saves time and simplifies the process and increases operational efficiency.

There are several key benefits to using the Converse API, like a consistent interface and simplified message handling. You will learn some examples of the GitHub code under chapter 4.

# **4.10 Sample Application: Using Converse API**

To get the GitLab details, refer to the appendix section of this book. In GitLab, locate the repository named **genai-bedrock-book-samples** and click on it.

Inside the **genai-bedrock-book-samples** repository is an AWS CloudFormation template that resides in the **cloudformation** folder. If you already executed the AWS CloudFormation template in Chapter 3 and didn't delete the stack afterward, you can skip the paragraph highlighted in grey below.

The task requires the execution of an AWS CloudFormation template, which should be performed once for all exercises in this book. A detailed guidance on how to manually execute the AWS CloudFormation template can be found in a file called **README** located within a directory named **cloudformation**. For more information about AWS CloudFormation template refer <https://aws.amazon.com/cloudformation/>.

**Disclaimer**: It is advisable to delete the AWS CloudFormation template if you are not actively participating in any exercises for some longer duration. Clear instructions for deleting the AWS CloudFormation template are provided within the README file itself.

However, in the **genai-bedrock-book-samples** folder there’s another subfolder titled **chapter4**. The **README** file within **chapter4** folder provides clear instructions on launching a **Notebook** on Amazon SageMaker.

|  |  |
| --- | --- |
| **File Name** | **File Description** |
| simple\_converse\_api.ipynb | 1. Understanding Amazon Bedrock client and Amazon Bedrock runtime client. 2. Example of Amazon Titan LLM foundation model with and without parameters using the Converse API. 3. Example of Anthropic LLM foundation model with and without parameters using the Converse API. 4. Example of Amazon Titan LLM foundation model with streaming API with and without parameters using the Converse API.   **Dependency**:  simple-sageMaker-bedrock.ipynb at Chapter 3 should work properly. |

# 3.8 Bedrock Interaction Sample Application

**Disclaimer**: Charges will apply upon executing above files. Therefore, it is important not to forget to clean up the kernel after studying the topic. Refer to the clean-up section for instructions on how to properly clean up the kernel.

# **4.11 Summary**

This chapter gives an overview of prompts in generative AI. Prompts are natural language inputs that guide generative AI systems to perform tasks. As generative AI models can create diverse content, including stories and images, the quality of the output relies on the prompt's clarity.

Prompt engineering is about creating effective prompts for better results. Good prompts lead to clear and high-quality outputs. In contrast, vague prompts often result in poor outcomes. Important elements of prompts are instructions, context, roles, input data, negative prompts, and output indicators.

The chapter also covers important concepts like text, tokens, and embeddings for generative AI language processing. It introduces prompt templates to assist you in crafting effective prompts. Additionally, it discusses in-context learning. It helps models generate relevant outputs by using extra context and examples.

In-context learning has different types. These include zero-shot, one-shot, few-shot, and chain-of-thought learning. They help models adjust to new tasks. This chapter also provides guidelines for effective prompt engineering. It emphasizes the importance of clear instructions and contextual clarity. Balanced prompts and iterative refinement are also crucial. These principles can enhance prompts across various AI applications. Lastly, it covers the Converse API, a very important API for your development.