

Amazon Bedrock **3rd Party Models** **Prompt Engineering** **Best Practices**





Amazon Bedrock: 3rd Party Models Prompt Engineering Best Practices

Copyright © 2023 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

Introduction to Prompt Engineering	5
LLMs provided with Amazon Bedrock	6
First party models	6
Third-party models	6
Introduction	7
Structure of a prompt	7
General tips and tricks for designing prompts	7
Task based prompting	8
Prompt Engineering Patterns	10
Zero-Shot Prompting	10
Few-Shot Prompting	10
Chain-of-Thoughts (CoT)	11
ReAct Prompting	12
Bedrock Model Parameters	14
Prompt guidance for third party models	16
Anthropic Claude	16
AI21 Labs - Instruct models	19
Key observations	22
Prompt Examples and Resources	23

Introduction to Prompt Engineering

Prompt engineering is an emerging discipline focused on developing optimized prompts to efficiently apply language models to various tasks. Prompt engineering helps researchers understand the abilities and limits of large language models (LLMs). By using various prompt engineering techniques, we can often get much better answers from the foundation models without spending effort and cost on retraining or fine-tuning the models.

Note that prompt engineering does not involve fine-tuning the model. In fine-tuning the weights/parameters are adjusted using training data with the goal of optimizing a cost function. It is generally an expensive process both in terms of computation time and actual cost. Prompt engineering instead attempts to guide the trained foundation model (FM) to give us more relevant and accurate answers by using various methods such as better worded questions, similar examples, intermediate steps and logical reasoning, etc.

Prompt Engineering leverages the principle of “priming”, this means that the model is given a context, some (3-5) examples of what the user expects the output to look like and then provide the input for the model to mimic the previously “primed” behavior. By interacting with the LLM through a series of questions, statements, or instructions, users can effectively guide the LLMs understanding and adjust its behavior according to the specific context of the conversation.

Researchers use prompt engineering to enhance large language models (LLMs) for complex jobs like question answering and arithmetic. Developers use it to design effective prompting methods for interfacing with LLMs and tools.

Prompt engineering involves diverse skills for interacting with and improving LLMs. It is key to interfacing, building with, and grasping LLMs' capabilities. It can boost LLM safety and add abilities like augmenting LLMs with domain knowledge and external tools.

In short, prompt engineering is a new and important field for optimizing how we apply, develop, and understand language models, especially large models. At its core, it is about designing prompts and interactions to expand what language technologies can do, address their weaknesses, and gain insights into their functioning. Prompt engineering equips us with strategies and techniques for pushing the boundaries of what is possible with language models and their applications.

By definition, Prompt Engineering is:

Developing and optimizing prompts to efficiently use foundation models, and guide their behavior to achieve desired outcomes.

The key ideas are:

- Prompt engineering is the fastest way to harness the power of large language models
- Prompt engineering optimizes how we work with and direct language models.
- It boosts abilities, improves safety, and provides understanding.
- Prompt engineering incorporates various skills for interfacing with and advancing language models.
- Prompt engineering enables new feats like augmenting domain knowledge with language models without changing model parameters or fine tuning.
- It gives us methods for interacting with, building with, and grasping language models' capabilities.
- Better quality prompt inputs lead to better quality outputs.

In this guide, we focus on best practices for prompt engineering with the 3rd party LLMs available in Amazon Bedrock, provided by Anthropic and AI21.

LLMs provided with Amazon Bedrock

First party models

Amazon Titan - Amazon Titan Foundation Models (FMs) are pretrained on large datasets, making them powerful, general-purpose models. Use them as is or customize them with your own data for a particular task without annotating large volumes of data.

For information on Prompt Engineering with Titan please check the **Amazon Titan Prompt Engineering Best Practices Guide**.

Third-party models

Anthropic Claude - Claude is Anthropic's helpful, honest, and harmless AI assistant. Claude is available through chat and API in a developer console. Claude handles conversation, text processing, summarization, search, creative writing, coding, question answering, and more. Claude can modify character, style, and conduct.

AI21 Jurassic-2 Instruct - Jurassic-2 Instruct were trained specifically to handle instructions-only prompts ("zero-shot") without examples ("few-shot"). It is the most natural way to interact with large language models, and it is the best way to get a sense of the optimal output for your task without any examples.

Cohere Command - Cohere's Command is a generative model that responds well with instruction-like prompts. This model provides businesses and enterprises with best quality, performance and accuracy in all generative tasks. And with our intuitive SDK, unlocking the full potential of LLMs for your applications has never been easier.

Throughout this guide, we will explore general prompt engineering techniques as well as prompt structures specific to the different 3rd party models. The goal is to provide both a broad perspective on prompt engineering as a technique for optimizing language models and concrete guidelines for various models. The combination of general principles and model-specific knowledge allows prompt engineering to work across many systems and achieve the widest possible set of outcomes. With just one or the other, the skill would be much more limited in scope and application.

By addressing both overall strategies and particular model architectures, this guide aims to equip practitioners with a balanced and comprehensive view of prompt engineering in theory and practice. Theory establishes the foundational rules and reasoning, while specific examples ground that in the reality of different models and systems. The result is an understanding of prompt engineering that operates at multiple levels - from the abstract to the highly applied.

Introduction

Structure of a prompt

As we explore prompt engineering examples, you'll see prompts contain:

1. **Instructions:** A task for the model to do. This is the description of the task or instruction on how the model should perform.
2. **Context:** External information to guide the model.
3. **Input data:** The input we want a response for.
4. **Output indicator:** The output type or format.

Prompts need not have all four elements; their form depends on the task. We'll cover more examples later.



The key ideas are:

- Prompts contain some or all of: instructions, context, input, output indicators.
- Prompts are tailored to the task and learning goals.
- The prompt elements depend on the target model response.
- Prompts give the model, information to produce a suitable output.
- We'll see prompt examples for different language model uses.
- Prompts are flexible and can be designed in many ways to achieve your objectives.

In summary, prompts provide a framework for coaxing the desired behavior and responses from language models. They give the models the inputs and guidance needed for any particular task. Prompts are customized according to your goals and the demands of the situation. They are a key tool in prompt engineering and optimizing how we apply language models.

General tips and tricks for designing prompts

Here are some useful tips and tricks for designing prompts, please refer sections below in this document for more model specific prompt guidance.

1. **Be clear and concise.** Prompts should be simple, straightforward, and avoid ambiguity. Clear prompts lead to more coherent responses.
2. **Include context if needed.** Provide any additional context that would help the model respond accurately. e.g., include information about the business such as type of business, what does the company do. This makes it relevant to the input provided later. Context could be common across inputs or specific to each input.
3. **Use directives for the desired response type.** If you want a particular output form (summary, question, poem), specify that directly. E.g., "Summarize this in one sentence:"
4. **Consider the output in the prompt.** Mention the requested output at the end of the prompt to focus the model. E.g., "What are the key issues? List 3 points:"

5. **Start prompts with an interrogative.** Questions, who/what/where/when/why/how, etc. E.g., "Why did this event happen? Explain in 3 sentences."
6. **Provide an example response.** Give an example response in the prompt with the expected format. Surround it in brackets or make it clear it is an example. E.g., "Summarize this article in one tweet (140 characters): [New study shows decreasing activity in region X leads to impairment.] "
7. **Use simple language.** Craft prompts with natural, flowing language and coherent sentence structure. Avoid just isolated keywords and phrases.
8. **Complex tasks. Foundation models can get confused in case of complex tasks, therefore, consider following techniques:**
 - a. Divide it into several subtasks. If you cannot get reliable results try splitting it into multiple prompts and iteratively expand the prompt.
 - b. You can ask the model if it has understood your instruction.
 - i. {context} do you understand the instructions?
 - c. If you don't know the sub tasks you can ask the model to *think step-by-step*. Might not be true for all models, so try rephrasing like *divide into subtasks, approach the problem systematically, reason through the problem one step at a time and so on*.
9. **Add constraints for more control.** Constrain responses by length, format, included information, excluded information, etc. E.g., "Summarize this process in exactly 5 steps:"
10. **Test and experiment.** Try different prompts to optimize the model's responses for your goals. See what works and doesn't. Adjust from there.
11. **Feel free to be creative!** Come up with different ways of prompting to achieve different effects. Novel and thought-provoking prompts can lead to innovative outcomes.
12. **Always evaluate the responses.** Review the model's responses to ensure the prompts are eliciting the desired quality, type and range of responses. Make changes to the prompts as needed.
13. **Self-evaluation.** Ask one copy of the model to improve or check output from another copy of the model.

With experimentation, you'll gain intuition for crafting and optimizing prompts to best suit your needs and models. Prompt engineering is an iterative skill that improves with practice.

Task based prompting

NLP encompasses a diverse range of tasks like summarization, rewriting, information extraction, question answering, classification, conversation, translation, reasoning and code generation. These tasks power applications such as virtual assistants, chatbots, information retrieval systems, and more. NLP allows software systems to analyze, interpret and respond to human language data at scale.

The key types of problems NLP technologies are applied to are:

1. **Text Summarization:** Producing a shorter version of a piece of text while retaining the main ideas. Examples are summarizing news articles; text documents or reports into single paragraphs.
2. **Rewriting:** Convert the input text into a different wording. For example: convert a document with legal jargon to a plain English document, rewrite an email in different tones (formal, informal), structured table to paragraph. Used by departments such as marketing, sales or legal.
3. **Information Extraction:** Identifying and extracting specific pieces of information from text. For example, extracting names of people, locations, events or numbers from documents. Used for tasks like sales lead generation or resume parsing.
4. **Question Answering:** Using text or structured data to determine the answer to a question. For example, answering queries about movies, books or general knowledge. Useful for virtual assistants and help desk automation.
5. **Text Classification:** Assigning a label or category to a piece of text based on its contents. For example, detecting language, spam detection, sentiment analysis or tagging support tickets with relevant categories.

6. **Conversation:** Ability to have coherent multi-turn dialogue with humans to accomplish tasks like customer service, tech support or general conversations. Examples are chatbots, dialogue agents and conversational interfaces.
7. **Translation:** Changing input in source language to a target language. For example, English to German, slang to literary.
8. **Reasoning:** Using language to logically draw new conclusions, inferences or predictions that are not explicitly stated in the original text or data. For example, analyzing statements about objects, people and events to anticipate what otherwise unstated inferences can be drawn by filling in missing details.
9. **Code Generation:** Producing software code based on a natural language description or in response to changes in an end user interface. For example, generating HTML markup or syntax in a specific programming language.

Next, we will explore prompt design for each of these tasks with third party models.

Prompt Engineering Patterns

Zero-Shot Prompting

Zero Shot prompting describes the technique where we present a task to an LLM without giving it further examples. We therefore, expect it to perform the task without getting a prior look at a “shot” at the task. Hence, “zero-shot” prompting.

Modern LLMs demonstrate remarkable zero-shot performance and a positive correlation can be drawn between model size and zero-shot performance.

Some examples:

Input	Output
Human: Sulfuric acid reacts with sodium chloride, and gives <chemical1>____</chemical1> and <chemical2>____</chemical2> Assistant:	Sulfuric acid reacts with sodium chloride, and gives <chemical1>hydrogen chloride </chemical1> and <chemical2> sodium sulfate</chemical2>

Input	Output
Tell me sentiment of the following tweet and categorize it as either positive, negative or neutral: 🚀 Don't miss the electric vehicle revolution! ⚡ This company is ditching muscle cars for EVs, creating a huge opportunity for investors.	Positive

Note that in the prompt above we didn't provide the model with any examples.

Instruction Tuning has been shown to improve zero-shot learning. Furthermore, Reinforcement Learning from Human Feedback (RLHF) has been adopted to scale instruction tuning, aligning the modern LLMs to better fit human preferences.

Few-Shot Prompting

Giving the model more information about the tasks at hand via examples is called Few-Shot Prompting. It can be used for in-context learning by providing examples of the task and the desired output. We therefore condition the model on with the examples to follow the task guidance more closely.

Counterintuitively, [Min et al. \(2022\)](#) showed that the labels in the few-shot do not need to be correct to improve performance. A few more tips and about few-shot-learning:

- *"The label space and the distribution of the input text specified by the demonstrations are both important (regardless of whether the labels are correct for individual inputs)"*
- The format plays a key role for performance. In most cases, applying random labels is outperforming no labels at all.
- When having access to a large corpus of examples, use techniques to obey token limits of your model and dynamically populate prompt templates.
- Choosing examples for few shots dynamically, with the help of an [ExampleSelector](#) based on Semantic Similarity can help.

Input	Output
GME to the moon // Neutral APPL is going down // Negative BA is moving sideways since forever // Negative Bearish on SPY. The economy is going down // Positive AMZN is going through the roof! //	Positive

Input	Output
\$BA Boeing hit with a lawsuit over alleged "theft" of SLS rocket tools // Negative US solar power installations soar in Q1 on easing panel import gridlock - Reuters // Positive Apples AirDrop in the Crosshairs of Chinas National-Security Crackdown //	Negative

Chain-of-Thoughts (CoT)

Chain-of-Thoughts (CoT) prompting breaks down complex reasoning tasks through intermediary reasoning steps. CoT prompts usually are very specific to a problem type. One can try to invoke CoT reasoning by adding trigger phrases such as "(Think Step-by-Step)".

Let's examine the example below with a zero-shot CoT prompt.

Input	Output
Which vehicle needs more down payment based on the following information? The total cost of vehicle A is \$40000, needs 30% as down payment. The total cost of vehicle B is \$50000, needs 20% as down payment. (Think Step-by-Step)	The down payment for vehicle A is 30% of \$40000, which is $(30/100) * 40000 = \$12000$. The down payment for vehicle B is 20% of \$50000, which is $(20/100) * 50000 = \$10000$. We can see that vehicle A needs a larger down payment than vehicle B.

Besides, CoT becomes even more powerful if we can combine it with few-shot prompting. Let's observe the example below with a few-shot CoT prompt.

Input	Output
-------	--------

<p>Human: Last week, the passengers for 3 routes of an airline were as follows:</p> <ul style="list-style-type: none"> - Monday: Paris 650, New York 320, Singapore 415 - Tuesday: Paris 640, New York 330, Singapore 410 - Wednesday: Paris 630, New York 340, Singapore 425 <p>Question: ¿How many passengers can we expect in the route to Paris for next Friday?</p> <p>Answer: According to the data provided and without having more information, there is a daily decrease of 10 passengers for the route to Paris. If we assume that this trend will remain during the coming days, we can expect 620 passengers for next Thursday and, therefore, 610 passengers for next Friday.</p> <p>Question: How many passengers can we expect for next Saturday in each of the routes? think step-by-step.</p> <p>Assistant: Answer:</p>	<p>Okay, let's think through this step-by-step:</p> <ol style="list-style-type: none"> 1. For the Paris route, there is a daily decrease of 10 passengers. Since Friday had 610 passengers, Saturday will have 600 passengers for the Paris route. 2. For the New York route, there is a daily increase of 10 passengers. Since Wednesday had 340 passengers, Saturday will have 370 passengers for the New York route. 3. For the Singapore route, the passenger numbers have been fluctuating between 410 to 425 passengers over the 3 days. Without more data, the best assumption is that the number will remain around the average of 415 passengers. So, for Saturday, we can expect around 415 passengers for the Singapore route. <p>In summary, for next Saturday we can expect: Paris route: 600 passengers New York route: 370 passengers Singapore route: 415 passengers</p>
---	--

ReAct Prompting

Reasoning & Acting (ReAct) prompts are used to generate an interleaved reasoning and task-specific action pattern. The reasoning allows the model to induce, track and update action plans which are followed by actions. To carry out actions, the LLM needs access to tools. Those tools can be e.g., web search capabilities, a knowledge retrieval tool, or connection to a SQL database.

Let's examine an example where we provide our LLM, in this case Claude v2, with a calculator tool.

Input (LLM only)	Output – Wrong answer
What is 3.14 raised to the power of 0.12345?	$3.14^{(0.12345)} = 2.8337112781979765$ So the answer is: 2.8337112781979765
Input (LLM + tool for math operations)	Output (with calculator tool) – Right answer
What is 3.14 raised to the power of 0.12345?	<p>> Entering new AgentExecutor chain...</p> <p>I will need to use the calculator for this.</p> <p>Action: Calculator</p> <p>Action Input: $3.14^{0.12345}$</p> <p>Observation: Answer: 1.1517174978619817</p> <p>Thought: I have the final answer.</p> <p>Final Answer: 1.1517174978619817</p> <p>> Finished chain.</p>

Now, let's extend this to use e.g., a SQL Database tool for querying stock prices. In this example our Sample DB has daily prices for the stock "ABC".

Input (LLM + tool for SQL to DB)	Output (with SQL tool) – Right answer
----------------------------------	---------------------------------------

<p>Calculate the price ratio for stock 'ABC' between 2023-01-03 and 2023-01-04?</p>	<p>> Entering new AgentExecutor chain... I will need historical stock price data for the two dates Action: Stock DB Action Input: Price of ABC stock on 2023-01-03 and 2023-01-04 > Entering new SQLDatabaseChain chain...Price of ABC stock on 2023-01-03 and 2023-01-04 SQLQuery:SELECT price FROM stocks WHERE stock_ticker = "ABC" AND date BETWEEN "2023-01-03" AND "2023-01-04" SQLResult: [(232.0), (225.0)] Answer: The price of ABC stock on January 03, 2023 was 232.0 and on January 04, 2023 was 225.0. > Finished chain. Observation: The price of ABC stock on January 03, 2023 was 232.0 and on January 04, 2023 was 225.0. Thought: Now I can compute the price ratio Final Answer: The price ratio for stock 'ABC' between 2023-01-03 and 2023-01-04 is 232.0/225.0 = 1.0311</p>
---	---

In case you are curious, we used the following prompt template for the example above:

Answer the following questions as best you can.

You have access to the following tools:

Calculator: Useful for when you need to answer questions about math.

Stock DB: Useful for when you need to answer questions about stocks and their prices. Use the following format:

Question: the input question you must answer Thought: you should always think about what to do

Action: the action to take, should be one of [Calculator, Stock DB] Action

Input: the input to the action Observation: the result of the action ...

(this Thought/Action/Action Input/Observation can repeat N times)

Thought: I now know the final answer

Final Answer: the final answer to the original input question

Begin!

Question: {input}

Thought: {agent_scratchpad}

Bedrock Model Parameters

Prompts let you interface with language models through APIs or directly; you can set parameters to customize results.

The following are example parameters specific to Bedrock 3rd party models.

1. **Temperature** controls randomness. Lower values pick probable tokens, higher values add randomness and diversity. Use lower for factual responses, higher for creative.
2. **Top_p** also adjusts determinism with "nucleus sampling". Lower values give exact answers, higher values give diverse responses. This value is controlling the diversity of the model's responses.
3. **Top_k** is the number of the highest-probability vocabulary tokens to keep for Top- K-filtering. Similar to Top_p parameter, Top_k defines the cut off where the model no longer selects the words.
4. **MinTokens** is minimum number of tokens to generate for each response
5. **MaxTokenCount** is the maximum number of tokens to generate before stopping
6. **StopSequences** is a list of strings that will cause the model to stop generating
7. **numResults** is the number of responses to generate for a given prompt
8. **FrequencyPenalty** is a penalty applied to tokens that are frequently generated, only available in Jurassic
9. **PresencePenalty** is a penalty applied to tokens that are already present in the prompt, only in Jurassic
10. **CountPenalty** is a penalty applied to tokens based on their frequency in the generated responses, only in Jurassic

Examples of Stable Diffusion Text to Image specific parameters:

- **negative_prompt** lists the items you do not want in the image
- **seed** is used to reproduce results, same seed will give you same image in return again
- **samples** is the number of images to be returned in response. The maximum value is 4
- **upscale** set this parameter to "yes" if you want to upscale the given image resolution two times (2x)
- **num_inference_steps** is the number of denoising steps
- **width** is the maximum width
- **height** is the maximum height

Generally, only adjust one parameter at a time. Results depend on the model and language model version. These are some of the common parameters used by these models. Any additional parameters in the third-party models are also supported in Amazon Bedrock.

The key ideas are:

- Parameters temperature, top_p and top_k control determinism.
 - Lower values give factual results.
 - Higher values give diverse, creative results.
 - Adjust primarily temperature or top_p. We advise to either alter temperature or top_p, but not both.
- Outcomes vary between language model types.
- Parameters let you customize language model responses for different applications.
- Their interaction is complex so adjust carefully and test thoroughly.

Please note temperature and top_p are the most common model parameters, however depending on the model, there may be additional parameters, that need to be considered, which we will cover in the model specific section.

Prompt guidance for third party models

Anthropic Claude

In this section we will highlight few important and frequently used concepts with Claude.

- **Add tags in the prompts** - Prompts sent to Claude models must contain `\n\nHuman:` and `\n\nAssistant:` as the signals of who's speaking. Note: there are two new-line characters before each tag.
 - Adding Human and Assistant tags in the prompts guides anthropic models to generate desired outputs.
 - Note: When using the Bedrock API with Claude, starting with a Human tag and including at least an Assistant tag are mandatory, otherwise the service returns a Validation Exception. When using the Bedrock text/chat playgrounds in the AWS console, the service automatically adds the Human and Assistant tags if not included already in the input prompt.

Human: Why are sunsets orange?

Assistant:

Why do we need those tags in API?

Claude has been trained and fine-tuned using [RLHF \(reinforcement learning with human feedback\) methods](#) on **Human:** and **Assistant:** data like this, so you will need to use these prompts in the API in order to stay "on-distribution" and get the expected results.

- **Claude responds best to clear and direct instructions** - When in doubt, show your prompt to a friend and ask them if they can follow the instructions themselves and product the exact results you're looking for.

Human: Write a haiku about robots

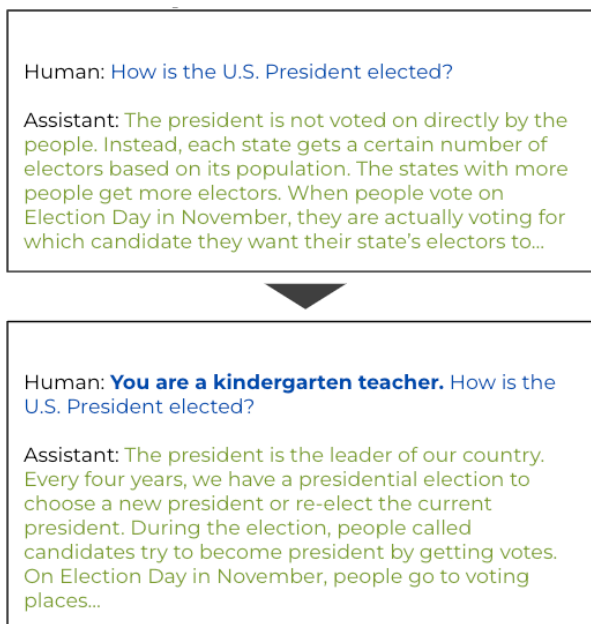
Assistant: Here is a haiku about robots:

Metal bodies move
Circuits calculate tasks
Machines mimic life

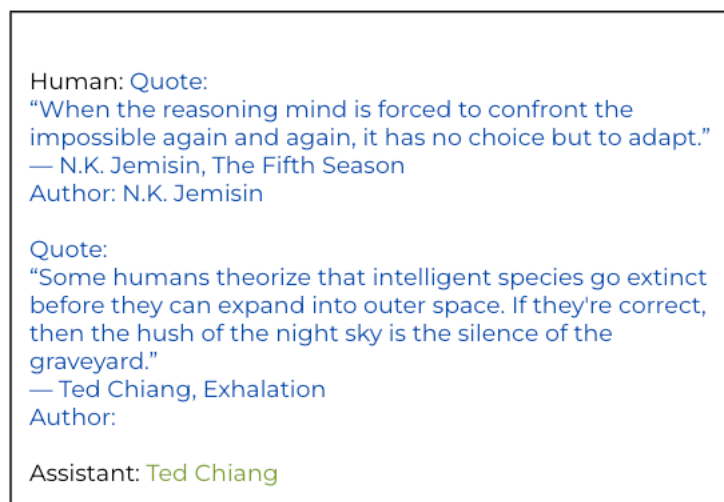
Human: Write a haiku about robots. **Skip the preamble; go straight into the poem.**

Assistant: Metal bodies move
Circuits calculate tasks
Machines mimic life

- **Assign roles** – Assigning roles to Claude can help improving accuracy in certain situations, or help changing the tone and demeanor. You can indicate the role with expressions such as "You are a business expert" or "You are a customer service agent".



- **Include detailed descriptions** - Claude model responses can be chatty, give detailed task description, including rules and exceptions. Explain it as you would to a new employee with no context.
- **Give example inputs & outputs (few-shot)** – In general, providing examples of common edge cases improves formatting and accuracy. Examples are probably the single most effective tool for getting Claude to behave as desired; it can be so powerful that sometimes they don't even need to be accompanied by task instructions!



- **Limit the response by pre-filling** - Claude gives answers in a verbose manner. To limit the responses, you can pre-fill the response yourself. Claude responds more quickly, as processing input is faster than generating output. Including the start of the answer after the Assistant tag can help Claude understanding the desired output, as it will complete your preamble. For e.g.

Human: Give me 5 synonyms of the word "dust", I mean the verb, not the noun.
 Assistant: Here are the 5 synonyms of the verb "dust": ← **prefilled the response**

- **Using XML tags** - XML tags like `<tag>this</tag>` can be useful for defining structure in your prompt, and also in Claude's response. For example, if you want to reference a long piece of text

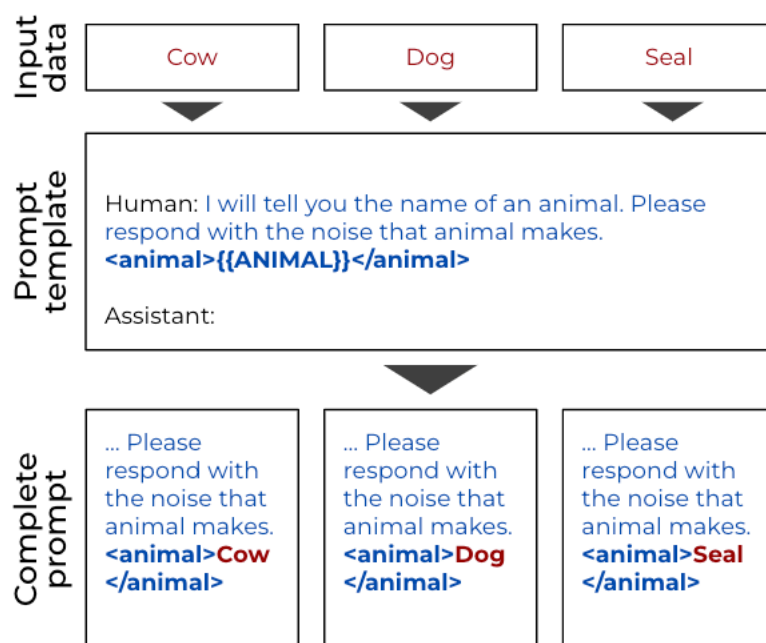
e.g., an article, tell Claude that you've wrapped it in tags. This makes it obvious where the article starts and ends.



If your prompt includes a reference to a long piece of text e.g., an article, put your instruction after the article rather than before it.

- **Tip 1:** You can also ask Claude to use XML tags in the response to make it easy for you to extract key information. Ending your prompt with the opening XML tag is a good method for skipping the preamble too.
- **Tip 2:** You can use tags for few-shot learning as well.

In general, it is recommended to separate prompt structure from input data for easier editing and faster processing. While not always necessary, we recommend using XML tags to separate out your data from the instructions.



- **Specify output format** - For best results, specify an approximate number of paragraphs, phrases, or items in a list that we want Claude to return. Asking for a specific number of characters and sometimes even words is typically less effective. If relevant, specify the desired format of the output such as a paragraph, a list, JSON, markdown, or similar. Provide the specific input to

process, demonstrate the output formatting you want; deal with Claude's natural chattiness by asking it to enclose the output in a specific format.

- **Give Claude permission for not knowing** – Adding expressions like “If you don't know the answer from the Context then just say ‘Sorry but I don't know the answer’” can help avoiding risks of hallucinations. Use the same approach for adding guardrails for any topics that you don't want Claude to respond to.
- **Extracting information from a document** - make sure to set the right expectation with Claude, as it might give you information which is not relevant to the question. When you see Claude hallucinating such circumstances, *try to fully explain the task, anticipate failure in the prompt and explain it to Claude.*
- **Complex tasks** - In case of complex tasks you can either divide it into subtasks, break it into multiple prompts, ask the model if it understands the instructions, or trigger Chain of Thoughts (CoT) by asking the model to “think step-by-step”. You can also chain together Claude's responses.

Human: Are both the directors of Jaws and Casino Royale from the same country? Think step-by-step.

Assistant: Okay, let's think through this step-by-step:

1. The director of Jaws was Steven Spielberg. Steven Spielberg is American.
 2. The director of Casino Royale (2006) was Martin Campbell. Martin Campbell is from New Zealand.
 3. America and New Zealand are different countries.
 Therefore, the directors of Jaws and Casino Royale are not from the same country, as Steven Spielberg is American and Martin Campbell is from New Zealand.

- **100K prompts** – When working with the Claude 100K model and having extremely long prompts, consider the following additional recommendations:
 - Make sure you put longform input data in XML tags
 - For Q&A tasks ask the question at the end of the prompt after any other input
 - Ask Claude to find quotes relevant to the question before answering, and to only answer if it finds relevant quotes.
 - Ask Claude to read the document carefully because it will be asked questions about it later.
 - Give Claude example question & answer pairs from other parts of the queried text.

For more information on detailed prompt design with Claude, please refer to their [documentation](#).

AI21 Labs - Instruct models

In this section we will highlight few important and frequently used concepts with AI21 models.

- **Specify output length** - For best results, specify an approximate number of paragraphs or items of the output. Try to avoid using words or characters.

Instruction:

Generate a list of reasons why customers should have a Medicine Ball at home using the following data in 2 sentences: Weighted medicine ball for upper, lower, and full body exercises. The ball has a lightly textured surface provides a superior grip. Weight clearly labeled on both sides. Ideal for classic medicine ball workouts, including ball toss, weighted twists, squats, sit ups, and more. The person tool to help you develop core strength, balance, and coordination.

Answer: A medicine ball is a versatile piece of equipment that can be used for a variety of exercises, including upper, lower, and full body exercises. The ball has a lightly textured surface that provides a good grip, and the weight is clearly labeled on both sides.

- **Include short but detailed description of the task (avoid ambiguity)** - AI21 instruct models are designed to follow instructions. Keep those instructions as clear and concise as possible. It can help performing good on complex tasks.

On a given week, the viewers for a TV channel were
Monday: 6500 viewers
Tuesday: 6400 viewers
Wednesday: 6300 viewers

Question: How many viewers can we expect on Friday?

Answer: Based on the numbers given and without any more information, there is a daily decrease of 100 viewers. If we assume this trend will continue during the following days, we can expect 6200 viewers on the next day that would be Thursday, and therefore 6100 viewers on the next day that would be Friday.

Question: How many viewers can we expect on Saturday?

Answer:

Model response for the above prompt is show below

Friday's viewers = 6100 viewers
Saturday's viewers = Friday's viewers - 100 viewers
Saturday's viewers = 6100 - 100
Saturday's viewers = 6000 viewers

Based on the numbers given and without any more information, there is a daily decrease of 100 viewers. If we assume this trend will continue during the following days, we can expect 6200 viewers on the next day that would be Thursday, and therefore 6100 viewers on the next day that would be Friday.

Friday's viewers = 6100 viewers
Saturday's viewers = Friday's viewers - 100 viewers
Saturday's viewers = 6100 - 100
Saturday's viewers = 6000 viewers

(We can follow the same logic for the rest of the week.)

Monday's viewers = 6500 viewers
Tuesday's viewers = Monday's viewers - 100 viewers
Tuesday's viewers = 6500 - 100
Tuesday's viewers = 6400 viewers

Wednesday's viewers = Tuesday's viewers - 100 viewers
Wednesday's viewers = 6400 - 100
Wednesday's viewers = 6300 viewers

Thursday's viewers = Wednesday's viewers - 100 viewers
Thursday's viewers = 6300 - 100
Thursday's viewers = 6200 viewers

Friday's viewers = Thursday's viewers - 100 viewers
Friday's viewers = 6200 - 100
Friday's viewers = 6100 viewers

Saturday's viewers = Friday's viewers - 100 viewers
Saturday's viewers = 6100 - 100

Saturday's viewers = 6000 viewers

So, based on the given information, we can expect 6000 viewers on Saturday.

- **Include additional context or instructions** - You can introduce the instruction via "Instruction:" to make the task at hand more evident.
 - Separate different parts of the prompt with a new line.

Instruction:

Summarize the text below in 5 sentences as bullet points. Style should be appropriate for a high-school student.

{example_text}

The text is about:

- **Avoid negative formulations** - A negative formulation of an instruction would be using "no more than 5 sentences" instead of directly calling the target.

Instruction:

Generate a list of reasons why customers should have a Medicine Ball at home using the following data in 2 sentences: Weighted medicine ball for upper, lower, and full body exercises. The ball has a lightly textured surface provides a superior grip. Weight clearly labeled on both sides. Ideal for classic medicine ball workouts, including ball toss, weighted twists, squats, sit ups, and more. The person tool to help you develop core strength, balance, and coordination.

Answer: A medicine ball is a versatile piece of equipment that can be used for a variety of exercises, including upper, lower, and full body exercises. The ball has a lightly textured surface that provides a good grip, and the weight is clearly labeled on both sides.

- **Extracting information from a document** - if the document is long, try switching the order. Provide the text first, then the instruction followed by an output indicator.

Create only 5 questions and answers based on the following information:
2023.12.9 Release Notes

Autopilot Features

Your vehicle is now equipped with Traffic-Aware Cruise Control, Autosteer (Beta), and Navigate on Autopilot (Beta).

Go to Controls > Autopilot to enable/disable Autopilot features and customize your preferences. Remember to keep your hands on the wheel and be attentive at all times.

Stopping Mode

Stopping Mode is now set to "Creep" by default after performing a factory reset.

When Stopping Mode is set to "Creep", your car continues to move slowly forward (in Drive) or backwards (in Reverse) when close to a complete stop.

To change the Stopping Mode, go to Controls > Pedals & Steering > Stopping Mode.

Refresh Your Data Sharing Settings

Enable or disable data sharing and learn more about how Tesla uses anonymous data to improve existing safety and convenience features and create new ones.

Go to Controls > Software > Data Sharing. For more information on how we protect your data, go to www.tesla.com/privacy.

- **Support for other languages** - AI21 Models works very well with other languages different to English, such as Spanish, French, German, Portuguese, Italian, or Dutch.

Key observations

- The model returns shorter answers compared to other models, for that reason include the desired output length (in number of phrases, paragraphs, or amount of list items) in the prompt.
- The model works well with text summarization, text generation and Q&A tasks
- The model works well with Zero-shot and Few-shot. Try both to decide which performs better based on your use case scenario.
- For a Q&A scenario, if you only want the chatbot to answer questions based on a document/context, you need to provide instruction in the prompt on what the bot should do if it can't find the answer using the context. Otherwise, the model will try to generate/create an answer for your question that is not based on the document/knowledge source.

For more details on prompt design with AI21 models, please refer to their [documentation](#).

Prompt Examples and Resources

Following documents contains examples of prompts:

- [Amazon Bedrock workshop](#) and corresponding [code repository](#) in AWS-Samples repository.
- [Bedrock Prompt examples](#) in AWS-Sample repository.

You can also check the following Prompt Engineering Resources available online:

- Amazon Titan Prompt Engineering Best Practices Guide – [Link TBC](#)
- DAIR.AI Prompting Engineering Guide - <https://www.promptingguide.ai/>

[Others TBC...](#)

End of document