Prompt engineering

Visit our prompt engineering tutorial to learn prompting via an interactive course.

Claude offers high-level baseline performance out of the box. However, prompt engineering can help you enhance its performance further and fine-tune its responses to better suit your specific use case. These techniques are not necessary for achieving good results with Claude, but you may find them useful in upleveling your inputs & outputs.

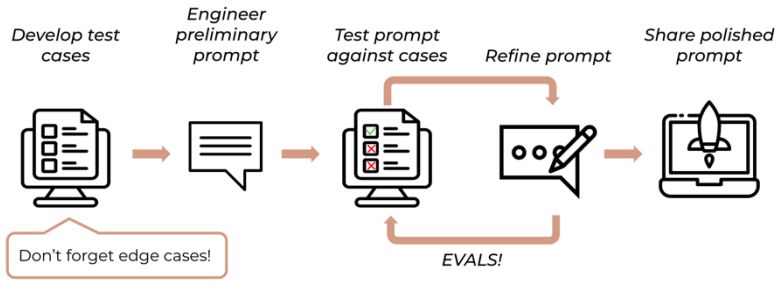
To quickly get up and running with a prompt or get introduced to prompting as a concept, see intro to prompting.

What is prompt engineering?

Prompt engineering is an empirical science that involves iterating and testing prompts to optimize performance. Most of the effort spent in the prompt engineering cycle is not actually in writing prompts. Rather, the majority of prompt engineering time is spent developing a strong set of evaluations, followed by testing and iterating against those evals.

The prompt development lifecycle

We recommend a principled, test-driven-development approach to ensure optimal prompt performance. Let's walk through the key high level process we use when developing prompts for a task, as illustrated in the accompanying diagram.



1. **Define the task and success criteria**: The first and most crucial step is to clearly define the specific task you want Claude to perform. This could be anything from entity extraction, question answering, or text summarization to more complex tasks like code generation or creative writing. Once you have a

well-defined task, establish the success criteria that will guide your evaluation and optimization process.

Key success criteria to consider include:

- **Performance and accuracy**: How well does the model need to perform on the task?
- **Latency**: What is the acceptable response time for the model? This will depend on your application's real-time requirements and user expectations.
- **Price**: What is your budget for running the model? Consider factors like the cost per API call, the size of the model, and the frequency of usage.

Having clear, measurable success criteria from the outset will help you make informed decisions throughout the adoption process and ensure that you're optimizing for the right goals.

2. **Develop test cases**: With your task and success criteria defined, the next step is to create a diverse set of test cases that cover the intended use cases for your application. These should include both typical examples and edge cases to ensure your prompts are robust. Having well-defined test cases upfront will enable you to objectively measure the performance of your prompts against your success criteria.
3. **Engineer the preliminary prompt**: Next, craft an initial prompt that outlines the task definition, characteristics of a good response, and any necessary context for Claude. Ideally you should add some examples of canonical inputs and outputs for Claude to follow. This preliminary prompt will serve as the starting point for refinement.
4. **Test prompt against test cases**: Feed your test cases into Claude using the preliminary prompt. Carefully evaluate the model's responses against your expected outputs and success criteria. Use a consistent grading rubric, whether it's human evaluation, comparison to an answer key, or even another instance of Claude's judgement based on a rubric. The key is to have a systematic way to assess performance.
5. **Refine prompt**: Based on the results from step 4, iteratively refine your prompt to improve performance on the test cases and better meet your success criteria. This may involve adding clarifications, examples, or constraints to guide Claude's behavior. Be cautious not to overly optimize for a narrow set of inputs, as this can lead to overfitting and poor generalization.
6. **Ship the polished prompt**: Once you've arrived at a prompt that performs well across your test cases and meets your success criteria, it's time to deploy it in your application. Monitor the model's performance in the wild and be prepared to make further refinements as needed. Edge cases may crop up that weren't anticipated in your initial test set.

Throughout this process, it's worth starting with the most capable model and unconstrained prompt length to establish a performance ceiling. Once you've achieved the desired output quality, you can then experiment with optimizations like shorter prompts or smaller models to reduce latency and costs as needed.

By following this test-driven methodology and carefully defining your task and success criteria upfront, you'll be well on your way to harnessing the power of Claude for your specific use case. If you invest time in designing robust test cases and prompts, you'll reap the benefits in terms of model performance and maintainability.

Prompt engineering techniques

Across your prompt development cycle, there are some techniques you can use to enhance Claude's performance, such as:

- Be clear & direct: Provide clear instructions and context to guide Claude's responses
- Use examples: Include examples in your prompts to illustrate the desired output format or style
- Give Claude a role: Prime Claude to inhabit a specific role (like that of an expert) in order to increase performance for your use case
- Use XML tags: Incorporate XML tags to structure prompts and responses for greater clarity
- Chain prompts: Divide complex tasks into smaller, manageable steps for better results
- Let Claude think: Encourage step-by-step thinking to improve the quality of Claude's output
- Prefill Claude's response: Start Claude's response with a few words to guide its output in the desired direction
- Control output format: Specify the desired output format to ensure consistency and readability
- Ask Claude for rewrites: Request revisions based on a rubric to get Claude to iterate and improve its output
- Long context window tips: Optimize prompts that take advantage of Claude's longer context windows

We also provide an experimental helper metaprompt that prompts Claude to create a prompt for you based on guidelines you provide. The metaprompt is experimental, but may be helpful for drafting an initial prompt or quickly creating many prompt variations for testing.

Note: Models older than the Claude 3 family may require more prompt engineering. For more information, see our legacy model guide.

Be clear & direct

When interacting with Claude, providing clear and direct instructions is essential for achieving the best results. Think of Claude like a smart but new employee who has no context on what to do aside from what you explicitly tell them. Just as when you instruct a human for the first time on a task, the more you explain exactly what you want in a straightforward manner, the better and more accurate Claude's response will be.

Provide detailed context and instructions

To ensure Claude understands your task, provide as much context and detail as possible. Include any specific rules or requirements for completing the task correctly. Consider the following example where we ask Claude to remove personally identifiable information (PII) from a given text:

Role	Prompt
User	Please remove all personally identifiable information from this text: {{TEXT}}

While this prompt may work for simple cases, it lacks the necessary details for Claude to consistently deliver the desired output. Here is a more detailed and clearly written version.

Role	Prompt
User	We want to anonymize some text by removing all personally identifiable information (PII). Please follow these steps: 1. Replace all instances of names, phone numbers, and home and email addresses with 'XXX'. 2. If the text contains no PII, copy it word-for-word without replacing anything. 3. Output only the processed text, without any additional commentary. Here is the text to process: {{TEXT}}

In this improved prompt, we:

1. Provide context upfront about the nature of the task
2. Define what constitutes PII (names, phone numbers, addresses)
3. Give specific, enumerated step-by-step instructions on how Claude should carry out its task
4. Give guidelines to Claude for how it should format its output

You can also use this technique to control other aspects of Claude's output. For example, by default, Claude often includes a friendly preamble before providing the requested output. If you prefer to receive only the essential information, you can instruct Claude to skip the preamble and get straight to the point.

Here is an example where Claude outputs a preamble:

Role	Prompt
User	Write a haiku about robots. Here is a haiku about robots:
Assistant (Claude's response)	Metal bodies move Circuits calculate tasks Machines mimic life

Here is an example where the prompt asks for Claude to skip the preamble:

Role	Prompt
User	Write a haiku about robots. Skip the preamble and provide only the poem.
Assistant (Claude's response)	Metal bodies move Circuits calculate tasks Machines mimic life

By explicitly asking Claude to skip the preamble, you can receive a more concise response that focuses solely on the requested output.

Use numbered lists or bullet points

When providing instructions for complex tasks, consider breaking them down into numbered steps or bullet points. This format makes it easier for Claude to follow the instructions and ensures that all requirements are met.

Here is an example prompt without this technique:

Role	Prompt
User	Turn this textbook chapter into a lesson plan: {{CHAPTER}}

Here is the same prompt with numbered steps:

Role	Prompt
User	Your task is turn the given textbook chapter into a comprehensive lesson plan. To do this task, you should: 1. Extract and summarize key concepts by section 2. Convert each extracted concept into a 30 minute lesson plan, including a list of questions to check comprehension. 3. Design an accompanying homework assignment that reinforces learnings from this chapter. Here is the chapter: <chapter> {{CHAPTER}} </chapter>

By presenting the instructions as numbered steps, you will both naturally write with more detail and better ensure that Claude executes its task to your standards.

Be specific about what you want

If you want Claude to provide a definitive answer or take a specific stance on a topic, make that clear in your prompt. When instructions are vague or open-ended, Claude may provide a more general response.

Here is an example of an open-ended question that causes Claude to equivocate:

Role	Prompt
User	Who is the best basketball player of all time?
Assistant (Claude's response)	There is no definitive answer to who is the best basketball player of all time. Reasonable basketball fans and analysts can disagree on this topic. Some of the players most frequently cited as possibilities for the GOAT (Greatest Of All Time) include Michael Jordan, LeBron James, Kareem Abdul-Jabbar, Wilt Chamberlain, Bill Russell, and Magic Johnson. It's a debate

Role	Prompt
	that will likely continue, as different eras, stats, accomplishments, and playing styles can be compared and interpreted in different ways.

Here is the same request, but with a more detailed and nuanced ask that successfully pushes Claude to have an opinion:

Role	Prompt
User	Who is the best basketball player of all time? Yes, there are differing opinions, but if you absolutely had to pick one player, who would it be?
Assistant (Claude's response)	There is no definitive answer to who is the best basketball player of all time, as reasonable people can disagree on this. However, if forced to choose just one player, I would say Michael Jordan is most commonly considered the GOAT (Greatest Of All Time) based on his exceptional talent, championships, awards, and cultural impact on the game of basketball. But others like LeBron James, Kareem Abdul-Jabbar, Wilt Chamberlain and Bill Russell are also frequently mentioned in the debate. It's a subjective call without a clear consensus.

The golden rule of clear prompting

When crafting your prompts, follow the golden rule of clear prompting: show your prompt to a friend or colleague and ask them to follow the instructions themselves to see if they can produce the exact result you want. If your friend is confused, Claude will likely be confused as well.

Remember, Claude is a powerful tool, but it relies on your guidance to deliver the best results. By providing clear, direct, and well-structured prompts, you can unlock Claude's full potential and achieve your desired outcomes more consistently.

Use examples

Examples are one of the most powerful tools for enhancing Claude's performance and guiding it to produce your desired output. By providing a few well-crafted examples in your prompt, you can significantly improve the accuracy, consistency, and quality of Claude's responses. This technique is particularly effective for tasks that are highly detailed or require structured outputs or adherence to specific formats.

This technique is also known as few-shot prompting (or one-shot prompting if only one example is provided).

Why use examples?

Examples serve as a learning tool for Claude, demonstrating exactly what kind of output you expect. They often are far more effective than simply describing or providing instructions, as they allow Claude to generalize patterns and apply them to new inputs. Just as humans often learn best by observing and imitating, Claude can quickly grasp the desired behavior by studying relevant examples.

Here are some key benefits of using examples in your prompts:

- **Improved accuracy:** Examples help Claude understand precisely what you want, reducing the likelihood of misinterpretation or irrelevant responses.
- **Increased consistency:** By providing a template for Claude to follow, examples ensure that responses maintain a consistent structure and style across different inputs.
- **Enhanced performance:** Well-chosen examples can significantly boost Claude's ability to handle complex or nuanced tasks, as it learns from the patterns and logic demonstrated in the examples.

Crafting effective examples

To get the most out of using examples in your prompts, consider the following guidelines on how to provide the most effective examples:

- **Relevance:** Ensure that your examples closely resemble the types of inputs and outputs you expect Claude to handle. The more similar the examples are to your actual use case, the better Claude will perform.
- **Diversity:** Include a variety of examples that cover different scenarios, edge cases, and potential challenges. This helps Claude generalize better and handle a wider range of inputs.
- **Clarity:** Make your examples clear, concise, and easy to understand. Use formatting tags like <example> to structure your examples and distinguish them from the rest of the prompt. Give Claude context as to what kind of example it's about to encounter when possible (e.g., Here are some examples of proper APA citations or Here are some examples of emails I've written).
- **Quantity:** While there's no hard rule for the optimal number of examples, aim to provide at least 3-5 examples to start to give Claude a solid foundation. You can always add more targeted examples if Claude's performance isn't meeting your expectations.

To provide examples, simply include them in your prompt, clearly distinguishing them from the actual task. We recommend using <example></example> tags to wrap your examples, making it easy for Claude to differentiate between the examples and the rest of the prompt.

Here's a prompt that demonstrates the use of examples to guide Claude's response:

Role	Content
User	I will give you some quotes. Please extract the author from the quote block. Here is an example: <example> Quote: "When the reasoning mind is forced to confront the impossible again and again, it has no choice but to adapt." — N.K. Jemisin, The Fifth Season Author: N.K. Jemisin </example> Quote: "Some humans theorize that intelligent species go extinct before they can expand into outer space. If they're correct, then the hush of the night sky is the silence of the graveyard." — Ted Chiang, Exhalation Author:
Assistant (Claude's response)	Ted Chiang

In this prompt, the example provides Claude with guidance on how to extract the author's name from a given quote, making it easy for Claude to replicate the process on a new input.

Formatting outputs

Examples are particularly effective for tasks that require structured or formatted outputs. Sometimes, instead of providing step-by-step formatting instructions, you can simply include a few examples of the desired output format (although we recommend having both instructions and examples together, as that is likely to be more effective than one without the other).

Suppose you want Claude to extract names and professions from a given text and format them as a list. Here's how you might prompt Claude with examples:

Role	Content
User	<example> Text: Sarah Martinez, a dedicated nurse, was known for her compassionate care at the local hospital. David Thompson, an innovative software engineer, worked tirelessly on groundbreaking projects. Output: 1. Sarah Martinez [NURSE] 2. David Thompson [SOFTWARE ENGINEER] </example> <example> Text: Chef Oliver Hamilton has transformed the culinary scene with his farm-to-table restaurant. Just down the street, you'll find the library, where head librarian Elizabeth Chen has worked diligently to create a welcoming space for all. Output: 1. Oliver Hamilton [CHEF] 2. Elizabeth Chen [LIBRARIAN] </example> Text: At the town's bustling farmer's market, you'll find Laura Simmons, a passionate organic farmer known for her delicious produce. In the community center, Kevin Alvarez, a skilled dance instructor, has brought the joy of movement to people of all ages. Output:
Assistant (Claude's response)	1. Laura Simmons [ORGANIC FARMER] 2. Kevin Alvarez [DANCE INSTRUCTOR]

By observing these examples, Claude learns to extract the relevant information and format it as a numbered list with names and professions in the desired style.

A word of caution

While examples are incredibly powerful, it's important to be mindful of potential pitfalls. Claude may sometimes pick up on unintended patterns in your examples, leading to overfitting or unexpected behaviors.

For instance, if all your example emails end with "Sincerely," Claude might learn to always sign off that way for emails it generates, even if it's not appropriate for every case. To mitigate this, ensure your examples are diverse and representative of the full range of desired outputs.

Iterating and refining

Crafting the perfect set of examples often involves iteration and refinement. If Claude's performance isn't quite meeting your expectations, consider the following:

- **Analyze the output:** Look for patterns in Claude's responses that deviate from what you want. This can help you identify areas where your examples might be unclear or misleading, or where more examples might help.
- **Add more examples:** If Claude struggles with certain types of inputs, provide additional examples that specifically address those scenarios.
- **Revise existing examples:** Sometimes, even small tweaks to your examples can make a big difference. Experiment with different wordings, formats, or structures to see what works best.
- **Get Claude's help:** Writing good examples is hard! You can ask Claude to evaluate the diversity or relevance of your examples for a given task, or generate new examples given a set of existing examples to reference.

Remember, prompt engineering is an iterative process. Don't be discouraged if your initial examples don't yield perfect results - with a bit of tweaking and experimentation, you'll be able to unlock Claude's full potential and achieve exceptional results for your applications.

Give Claude a role

Claude is a highly capable AI assistant, but sometimes it benefits from having additional context to understand the role it should play in a given conversation. By assigning a role to Claude, you can prime it to respond in a specific way, improve its accuracy and performance, and tailor its tone and demeanor to match the desired context. This technique is also known as role prompting.

How role prompting works

Role prompting is a powerful technique that can enhance Claude's performance in two main ways:

1. **Improved accuracy and performance:** Particularly in certain complex situations, such as solving complex math problems or writing code, assigning a relevant role to Claude can significantly improve its performance and understanding of the task at hand.
2. **Tailored tone and style:** By specifying a role, you can influence Claude's tone, demeanor, and communication style to better suit your needs. For example, asking Claude to act as a kindergarten teacher will result in a very different response compared to asking it to be a graduate professor.

When to use role prompting

While role prompting is not always necessary, it can be incredibly useful in the following scenarios:

- **Highly technical tasks:** If you need Claude to perform complex tasks related to logic, mathematics, or coding, assigning an appropriate role can help it excel at the task, even if it might have struggled without the role prompt. Even if Claude isn't struggling, role prompting might still improve performance to new levels.
- **Specific communication styles:** When you require a particular tone, style, or level of complexity in Claude's responses, role prompting can be an effective way to achieve the desired output.
- **Enhancing baseline performance:** Unless you are severely limited by token count, there is rarely a reason not to use role prompting if you want to try improving Claude's performance beyond its baseline capabilities.

Role prompting examples

Here are a few examples that demonstrate the power of role prompting:

Solving a logic puzzle

Let's consider the following logic puzzle:

There are two ducks in front of a duck, two ducks behind a duck and a duck in the middle. How many ducks are there?

Without role prompting, Claude might struggle to provide the correct answer or make errant assumptions that result in only a partial answer:

Role	Content
User	There are two ducks in front of a duck, two ducks behind a duck and a duck in the middle. How many ducks are there?
Assistant (Claude's response)	[Automatically outputting thinking step by step as part of its training...] The total number of ducks is 5.

However, by assigning the role of a logic bot, Claude's performance improves significantly and it is able to catch the nuance that multiple answers are possible:

Role	Content
User	You are a master logic bot designed to answer complex logic problems. Solve this logic puzzle. There are two ducks in front of a duck, two ducks behind a duck and a duck in the middle. How many ducks are there?
Assistant (Claude's response)	[Automatically outputting thinking step by step as part of its training...] Without additional clarification or assumptions, the logic puzzle does not provide enough information to determine the exact number of ducks. The answer could range from a minimum of 3 ducks to a maximum of 5 ducks, depending on the interpretation of the given information.

Explaining a concept to different audiences

Role prompting can be used to adjust Claude's communication style based on the intended audience. Consider the following prompts and how Claude's output differs depending on the assigned role:

Role	Content
User	You are a kindergarten teacher. Succinctly explain why the sky is blue to your students.
Assistant (Claude's response)	Have you ever wondered why the sky looks so pretty and blue? The reason is because of the sunlight! When the sun's rays shine down through the Earth's atmosphere, they hit lots of tiny gas molecules in the air. The blue colors in the sunlight get scattered and bounced all around by these molecules. And that's what makes the sky look blue to our eyes! Isn't that neat? The bright blue sky is like nature's big colorful painting just for us.

Role	Content

Role	Content
User	You are a graduate physics professor. Succinctly explain why the sky is blue to your students.
Assistant (Claude's response)	The sky appears blue due to the phenomenon of Rayleigh scattering of sunlight by the molecules in the Earth's atmosphere. Sunlight, which is composed of a spectrum of wavelengths, interacts with the gaseous molecules in the atmosphere. The shorter wavelengths of visible light, corresponding to the blue and violet colors, are scattered more efficiently by these molecules than the longer wavelengths. This selective scattering of the shorter blue wavelengths in all directions by the atmospheric molecules is what causes the sky to appear blue when viewed from the Earth's surface. The longer wavelengths of red and orange, being less scattered, tend to pass through the atmosphere more directly, giving rise to the reddish hues we see during sunrise and sunset when the sunlight has to travel through a greater depth of the atmosphere.

Tips for effective role prompting

To get the most out of role prompting, keep these tips in mind:

1. **Be specific:** Provide clear and detailed context about the role you want Claude to play. The more information you give, the better Claude can understand and embody the desired role.
2. **Experiment and iterate:** Try different roles and variations of your prompts to find the best approach for your specific use case. Prompt engineering often involves experimentation and iteration to achieve optimal results.

Use XML tags

XML tags are a powerful tool for structuring prompts and guiding Claude's responses. Claude is particularly familiar with prompts that have XML tags as Claude was exposed to such prompts during training. By wrapping key parts of your prompt (such as instructions, examples, or input data) in XML tags, you can help Claude better understand the context

and generate more accurate outputs. This technique is especially useful when working with complex prompts or variable inputs.

Looking for more advanced techniques? Check out long context window tips to learn how XML tags can help you make the most of Claude's extended context capabilities.

What are XML tags?

XML tags are angle-bracket tags like <tag></tag>. They come in pairs and consist of an opening tag, such as <tag>, and a closing tag marked by a /, such as </tag>. XML tags are used to wrap around content, like this: <tag>content</tag>.

Opening and closing XML tags should share exactly the same name. The tag name can be anything you like, as long as it's wrapped in angle brackets, although we recommend naming your tags something contextually relevant to the content it's wrapped around.

XML tags should always be referred to in pairs and never as just as the first half of a set (e.g., Using the document in <doc></doc> tags, answer this question.).

XML tag names

There is no canonical best set of XML tag names that Claude performs particularly well with. For example, <doc> works just as well as <document>. The only time you need very specific XML tag names is in the case of function calling.

Why use XML tags?

There are several reasons why you might want to incorporate XML tags into your prompts:

1. **Improved accuracy:** XML tags help Claude distinguish between different parts of your prompt, such as instructions, examples, and input data. This can lead to more precise parsing of your prompt and thus more relevant and accurate responses, particularly in domains like mathematics or code generation.
2. **Clearer structure:** Just as headings and sections make documents easier for humans to follow, XML tags help Claude understand the hierarchy and relationships within your prompt.

3. **Easier post-processing:** You can also ask Claude to use XML tags in its responses, making it simpler to extract key information programmatically.

How to use XML tags

You can use XML tags to structure and delineate parts of your prompt from one another, such as separating instructions from content, or examples from instructions.

Role	Content
User	Please analyze this document and write a detailed summmary memo according to the instructions below, following the format given in the example: <document> {{DOCUMENT}} </document> <instructions> {{DETAILED_INSTRUCTIONS}} </instructions> <example> {{EXAMPLE}} </example>

Handling variable inputs

When working with prompt templates that include variable inputs, use XML tags to indicate where the variable content should be inserted, such as in the following example:

Role	Content
User	I will tell you the name of an animal. Please respond with the noise that animal makes. <animal>{{ANIMAL}}</animal>

As a general rule, you should ways separate your variable inputs from the rest of your prompt using XML tags. This makes it clear to Claude where the examples or data begin and end, leading to more accurate responses.

Requesting structured output

You can ask Claude to use XML tags in its responses to make the output easier to parse and process:

Role	Content
User	Please extract the key details from the following email and return them in XML tags: - Sender name in <sender></sender> tags - Main topic in <topic></topic> tags - Any deadlines or dates mentioned in <deadline></deadline> tags <email> From: John Smith To: Jane Doe Subject: Project X Update Hi Jane, I wanted to give you a quick update on Project X. We've made good progress this week and are on track to meet the initial milestones. However, we may need some additional resources to complete the final phase by the August 15th deadline. Can we schedule a meeting next week to discuss the budget and timeline in more detail? Thanks, John </email>

Claude's response:

Role	Content
Assistant (Claude's response)	<sender>John Smith</sender> <topic>Project X Update</topic> <deadline>August 15th</deadline>

XML tags make it easier to retrieve targeted details from Claude's response by allowing for programmatic extraction of content between specific tags.

When calling Claude via the API, you can pass closing XML tags (e.g., </json>) to the stop_sequences parameter to have Claude stop generating once it reaches the desired endpoint. This can save both money and time by eliminating any concluding remarks after the core response. The same is true of skipping Claude's friendly preamble by prefilling Claude's response with an opening XML tag.

XML best practices

To get the most out of XML tags, keep these tips in mind:

- Use descriptive tag names that reflect the content they contain (e.g., <instructions>, <example>, <input>).
- Be consistent with your tag names throughout your prompts.
- Always include both the opening (<tag>) and closing (</tag>) tags, including when you reference them, such as Using the document in <doc></doc> tags, answer this question.
- You can and should nest XML tags, although more than five layers of nesting may decrease performance depending on the complexity of the use case.

Chain prompts

You can think of working with large language models like juggling. The more tasks you have Claude handle in a single prompt, the more liable it is to drop something or perform any single task less well. Thus, for complex tasks that require multiple steps or subtasks, we recommend breaking those tasks down into subtasks and chaining prompts to ensure highest quality performance at every step.

What is prompt chaining?

Prompt chaining involves using the output from one prompt as the input for another prompt. By chaining prompts together, you can guide Claude through a series of smaller, more manageable tasks to ultimately achieve a complex goal.

Prompt chaining offers several advantages:

- Improved accuracy and consistency in the generated output at each distinct step
- Easier troubleshooting by isolating specific subtasks that may be particularly error-prone or challenging to handle

---

When to use prompt chaining

Consider using prompt chaining in the following scenarios:

1. **Multi-step tasks:** If your task requires multiple distinct steps, such as researching a topic, outlining an essay, writing the essay, then formatting the essay, chaining prompts can help ensure each step of the task has Claude's full focus and is executed at a high level of performance.
2. **Complex instructions:** When a single prompt contains too many instructions or details, Claude may struggle to follow them consistently. Breaking the task into a series of chained subtasks can improve performance for each subtask.
3. **Verifying outputs:** You can use chaining to ask Claude to double-check its own outputs with a given rubric and improve its response if needed, ensuring higher quality results. For example, after generating a list of items, you can feed that list back to Claude and ask it to verify the list's accuracy or completeness.
4. **Parallel processing:** If your task has multiple independent subtasks, you can create separate prompts for each subtask and run them in parallel to save time.

Tips for effective prompt chaining

1. **Keep subtasks simple and clear:** Each subtask should have a well-defined objective and simple instructions. This makes it easier for Claude to understand and follow.
2. **Use XML tags:** Enclosing inputs and outputs in XML tags can help structure the data and make it easier to extract and pass on to the next step when chaining prompts.

Examples

Here are a few examples showcasing how to use chaining prompts and breaking tasks into subtasks:

Answering questions using a document and quotes

Here we want Claude to, given a document and a question, generate an answer using relevant quotes from the document.

Prompt 1: Extracting the quotes

Role	Content
User	Here is a document, in <document></document> XML tags: <document> {{DOCUMENT}} </document> Please extract, word-for-word, any quotes relevant to the question {{QUESTION}}. Please enclose the full list of quotes in <quotes></quotes> XML tags. If there are no quotes in this document that seem relevant to this question, please say "I can't find any relevant quotes".

Prompt 2 (using {{QUOTES}} output from Prompt 1): Answering the question

Role	Content
User	I want you to use a document and relevant quotes from the document to answer a question. Here is the document: <document> {{DOCUMENT}} </document> Here are direct quotes from the document that are most relevant to the question: <quotes> {{QUOTES}} </quotes> Please use these to construct an answer to the question "{{QUESTION}}" Ensure that your answer is accurate and doesn't contain any information not directly supported by the quotes.

Validating outputs

In this example, the goal is to have Claude identify grammatical errors in an article, then double-check that the list of errors is complete.

Prompt 1: Generating a list of errors

Role	Prompt 1
User	Here is an article: <article> {{ARTICLE}} </article> Please identify any grammatical errors in the article. Please only respond with the list of errors, and nothing else. If there are no grammatical errors, say "There are no errors."

Prompt 2 (using {{ERRORS}} output from Prompt 1): Double checking that the list is comprehensive

Role	Prompt 2
User	Here is an article: <article> {{ARTICLE}} </article> Please identify any grammatical errors in the article that are missing from the following list: <list> {{ERRORS}} </list> If there are no errors in the article that are missing from the list, say "There are no additional errors."

Parallel processing

In this example, the goal is to have Claude explain a concept to readers at three different levels (1st grade, 8th grade, college freshman) by first creating an outline, then expanding it into a full explanation.

Prompt 1 (create three different versions, one for each reading level): Create an outline

Role	Prompt 1
User	Here is a concept: {{CONCEPT}} I want you to write a three sentence outline of an essay about this concept that is appropriate for this level of reader: {{LEVEL}} Please only respond with your outline, one sentence per line, in <outline></outline> XML tags. Don't say anything else.

Prompt 2 (using {{OUTLINE}} output from Prompt 1, one per reading level):

Create full explanations using the outline

Role	Prompt 2
User	Here is an outline: <outline> {{OUTLINE}} </outline> Please expand each sentence in the outline into a paragraph. Use each sentence word-for-word as the first sentence in its corresponding paragraph. Make sure to write at a level appropriate for this type of reader: {{LEVEL}}.

Let Claude think

When faced with a complex question or task, it's often beneficial to let Claude think through the problem step-by-step before providing a final answer. This technique, also known as chain of thought (CoT) prompting, can significantly improve the accuracy and nuance of Claude's responses.

Why let Claude think?

Imagine being woken up and immediately asked to answer a series of complicated questions. You'd probably perform better if given some time to think through your answers first, right? The same applies to Claude!

Allowing Claude to reason through a problem before responding can lead to more accurate and intelligent outputs, particularly for multifaceted or challenging queries. By

explicitly instructing Claude to think step-by-step, you encourage a more methodical and thorough approach to problem-solving.

Furthermore, examining Claude's thought process can be invaluable for troubleshooting and identifying areas where your prompt instructions may be unclear or easily misinterpreted.

It's important to note that thinking cannot happen without output! Claude must output its thinking in order to actually "think."

How to prompt for thinking step-by-step

The simplest way to encourage thinking step-by-step is to include the phrase "Think step by step" in your prompt. For example:

Role	Content
User	Are both the directors of Jaws and Casino Royale from the same country? Think step by step.

Claude will then break down its thought process before providing a final answer:

Role	Content
Assistant (Claude's response)	To determine if the directors of the movies "Jaws" and "Casino Royale" are from the same country, let's break it down into steps: Step 1: Identify the director of the movie "Jaws." The director of "Jaws" is Steven Spielberg. Step 2: Identify the country of origin for Steven Spielberg. Steven Spielberg was born in Cincinnati, Ohio, United States. Step 3: Identify the director of the movie "Casino Royale." The director of the 2006 movie "Casino Royale" is Martin Campbell. Step 4: Identify the country of origin for Martin Campbell. Martin Campbell was born in Hastings, New Zealand. Step 5: Compare the countries of origin for both directors. Steven Spielberg is from the United States, while Martin Campbell is from New Zealand.

Role	Content
	Therefore, the directors of "Jaws" and "Casino Royale" are not from the same country.

For more complex queries, you can guide Claude's thinking by specifying the steps it should take. Here's an example:

Role	Content
User	Use the following clues to answer the multiple-choice question below, using this procedure: 1. Go through the clues one by one and consider whether each is potentially relevant 2. Combine the relevant clues to reason out the answer to the question 3. Map the answer to one of the multiple choice options: (a), (b), or (c) Clues: 1. Miss Scarlett was the only person in the lounge. 2. The person with the pipe was in the kitchen. 3. Colonel Mustard was the only person in the observatory. 4. Professor Plum was not in the library nor the billiard room. 5. The person with the candlestick was in the observatory. Question: Was Colonel Mustard in the observatory with the candlestick? (a) Yes; Colonel Mustard was in the observatory with the candlestick (b) No; Colonel Mustard was not in the observatory with the candlestick (c) Unknown; there is not enough information to determine whether Colonel Mustard was in the observatory with the candlestick

By outlining a clear thinking process, you help Claude focus its reasoning on the most relevant information and ensure it thinks through all the necessary factors to perform well at its given task.

Capturing Claude's thought process

To make it easier to separate Claude's step-by-step reasoning from its final response, consider using XML tags like <thinking> and <answer>. You can instruct Claude to place its thought process inside <thinking> tags and its ultimate answer within <answer> tags.

Here's an example prompt with this method:

Role	Content
User	[Rest of prompt] Before answering the question, please think about it step-by-step within <thinking></thinking> tags. Then, provide your final answer within <answer></answer> tags.

You can even prefill the <thinking> tag in the Assistant role to guide Claude as to where to begin:

Role	Content
Assistant (prefill)	<thinking>

Claude will then complete its thought process within the tags and provide its final answer:

Role	Content
Assistant (Claude's response)	[Reasoning through the problem step-by-step] </thinking> <answer> [Final answer] </answer>

Using tags makes it simple to extract just the final answer within <answer></answer> tags during post-processing if desired.

Some considerations

While encouraging step-by-step thinking can greatly enhance Claude's responses, keep these points in mind:

- Thinking cannot occur unless Claude is allowed to output its thought process. There's no way to have Claude think privately and only return the final answer.
- Prompting for step-by-step reasoning will increase the length of Claude's outputs, which can impact latency. Consider this tradeoff when deciding whether to use this technique.

Prefill Claude's response

When using Claude, you have the unique ability to guide its responses by prefilling the Assistant message. This powerful technique allows you to direct Claude's actions, control

the output format, and even help Claude stay in character during role-play scenarios. In some cases where Claude is not performing as expected, a few prefilled sentences can vastly improve Claude's performance.

Check out our blog post Long context prompting for Claude 2.1 to see an example of highly effective prefilling.

Why prefill Claude's response?

Prefilling Claude's response offers several key benefits:

1. **Increased steerability**: By providing some initial text for Claude to continue from, you can steer Claude's response in a desired direction. This is particularly useful when you want Claude to focus on a specific topic, generate a particular type of content, or act a certain way.
2. **Control output format**: Prefilling allows you to specify the exact format you want Claude to use for its output. This is especially handy when working with structured data formats like JSON or XML. For more details on this, see our guide on controlling output format.
3. **Maintain character consistency**: In role-play scenarios, prefilling Claude's response can help Claude stay in character throughout a long conversation. By consistently reminding Claude of its role in the Assistant message, you can better ensure that Claude maintains the desired persona. Check out keep Claude stay in character for more details.

How to prefill Claude's response

To prefill Claude's response, simply include the desired initial text in the Assistant message when making an API request. Here's an example prompt:

Role	Good Prompt
User	Please extract the name, size, price, and color from this product description and output it within a JSON object. <description>The SmartHome Mini is a compact smart home assistant available in black or white for only $49.99. At just 5 inches wide, it lets you control lights, thermostats, and other connected devices via voice or app—no matter where you place it in your home. This affordable little hub brings convenient hands-free control to your smart devices.</description>

Role	Good Prompt
Assistant (prefill)	{

In this example, by starting the Assistant message with {, we constrain Claude's output to be the rest of the requested JSON schema.

Role	Response
Assistant (Claude's response)	"name": "SmartHome Mini", "size": "5 inches wide", "price": "$49.99", "colors": ["black", "white"] }

Here is how the above prompt would be written in code in the Messages API format:

```
import anthropic

client = anthropic.Anthropic(
    # defaults to os.environ.get("ANTHROPIC_API_KEY")
    api_key="my_api_key",
)
message = client.messages.create(
    model="claude-2.1",
    max_tokens=1000,
    temperature=0,
    messages=[
        {
            "role": "user",
            "content": "Please extract the name, size, price, and color from this product description and output it within a JSON object.\n\n<description>The SmartHome Mini is a compact smart home assistant available in black or white for only $49.99. At just 5 inches wide, it lets you control lights, thermostats, and other connected devices via voice or app—no matter where you place it in your home. This affordable little hub brings convenient hands-free control to your smart devices.\n</description>"
        }
        {
            "role": "assistant",
            "content": "{"
        }
    ]
```

)
print(message.content)

Control output format (JSON mode)

Claude is highly capable of producing output in a wide variety of formats. By providing clear instructions, examples, and prefilled responses, you can guide Claude to generate responses that adhere to your desired structure and style.

Specifying the desired format

One of the simplest ways to control Claude's output is to simply state the format you want. Claude can understand and follow instructions related to formatting, and format outputs such as:

- JSON
- XML
- HTML
- Markdown
- CSV
- Custom formats

For example, if you want Claude to generate a haiku in JSON format, you can use a prompt like this:

Role	Content
User	Please write a haiku about a cat. Use JSON format with the keys "first_line", "second_line", and "third_line".

Assistant (Claude's response)	{
	"first_line": "Soft furry feline",
	"second_line": "Napping in the warm sunlight",
	"third_line": "Purring contentedly"
	}

Prefilling Claude's response

Another effective way to control Claude's output format is to prefill the Assistant field with the beginning of the desired format. This technique is particularly useful when working with structured formats like JSON or HTML.

For example, to generate a recipe in JSON format:

Role	Content
User	Please generate a recipe for chocolate chip cookies in JSON format.
Assistant (prefill)	{
Assistant (Claude's response)	"title": "Chocolate Chip Cookies", "ingredients": ["2 1/4 cups all-purpose flour", "1 tsp baking soda", "1 tsp salt", "1 cup unsalted butter, softened", "3/4 cup granulated sugar", "3/4 cup brown sugar", "1 tsp vanilla extract", "2 large eggs", "2 cups semisweet chocolate chips"], "instructions": ["Preheat the oven to 375° F (190° C) and line baking sheets with parchment paper.", "In a medium bowl, whisk together the flour, baking soda, and salt. Set aside.", "In a large bowl, beat the butter and sugars with an electric mixer until light and fluffy, about 2-3 minutes.", "Beat in the vanilla and eggs one at a time until well combined.", "Gradually stir the flour mixture into the butter mixture until just combined.", "Fold in the chocolate chips.", "Drop rounded tablespoons of the dough onto the prepared baking sheets, about 2 inches apart.", "Bake for 9-11 minutes, until the edges are golden brown.", "Allow the cookies to cool on the baking sheets for 5 minutes before transferring to a wire rack to cool completely."], "yield": "About 3 dozen cookies" }

By prefilling the response with the opening JSON curly brace, you give Claude a clear starting point for generating the recipe in the desired format. This also is another technique you can use to have Claude skip its friendly preamble text.

Providing examples

In addition to explicit instructions, providing examples of the desired output format can help Claude better understand your requirements. When including examples, make it clear that Claude should follow the formatting of the examples provided (otherwise Claude may pick up other details from the provided examples, such as content or writing style).

Here is an example prompt showcasing this technique:

Role	Content
User	Your task is to write a poem. Here are some examples of ideal formatting for the poem: <poem> Title: "Autumn Leaves" Verse 1: Crisp autumn leaves dance In the gentle, chilly breeze A colorful sight Verse 2: Red, orange, and gold Painting the world with beauty Before winter comes </poem> <poem> Title: "Moonlit Night" Verse 1: Moonlight casts shadows Across the tranquil garden A peaceful retreat Verse 2: Stars twinkle above As crickets sing their nightsong Nature's lullaby </poem> Now, please write a poem about a sunset, following the formatting of the examples above.
Assistant (prefill)	<poem>

Role	Content
Assistant (Claude's response)	Title: "Sunset Symphony" Verse 1: Fiery sun descends Painting the sky with wonder A masterpiece forms Verse 2: Pinks, purples, and reds Blending in pure harmony Day bids night hello </poem>

Tips for better output control

- Be as specific as possible in your instructions
- Use clear and consistent formatting in your prompts
- Provide multiple examples when possible to reinforce the desired format
- Experiment with different combinations of techniques to find what works best for your use case

Ask Claude for rewrites

Sometimes, even with a well-crafted prompt, Claude's response might not quite hit the mark or be completely accurate. In these cases, you can leverage Claude's ability to revise its own work by asking for a rewrite. By providing clear instructions and a rubric for the desired output, you can guide Claude to generate content that better aligns with your goals.

When to ask for a rewrite

Consider requesting a rewrite when:

- The response is close to what you need but lacks specific details or formatting
- The output doesn't fully adhere to the constraints outlined in your initial prompt
- You are unsure if the result is accurate and want Claude to double check to make sure (consider it a mini one-off LLM-graded eval)

Rewrites are particularly useful when your prompt contains multiple constraints, such as a specific format (e.g., a poem or list) and specific content topics. Sometimes, Claude may prioritize one constraint over another, especially if there are many complex steps to follow. Asking for a rewrite allows you to emphasize the importance of whatever constraints Claude initially missed.

How to request a rewrite

When sending requests through the API (including the functions in Claude for Sheets), you must always include the previously generated text in your prompt. Here's an example prompt template:

Role	Prompt
	Here's a paragraph: <paragraph> {{PARAGRAPH}} </paragraph>
User	I'd like you to rewrite it using the following instructions: <instructions> {{INSTRUCTIONS}} </instructions> Please put your rewrite in <rewrite></rewrite> tags.
Assistant (prefill)	<rewrite>

For lower volume use cases, you can paste the paragraph to be rewritten in place of {{PARAGRAPH}}. To automate this process, consider setting up a prompt chain so that Claude's response to a previous prompt gets substituted for the {{PARAGRAPH}} placeholder.

When using Claude for Sheets, you can structure your Sheet so that each row contains a different set of instructions to be substituted for the {{INSTRUCTIONS}} placeholder. This allows you to generate multiple types of rewrites simultaneously. Check out our Claude for Sheets prompting examples spreadsheet for a sample Sheet structure that demonstrates this and other techniques.

Crafting effective rewrite instructions

To get the most out of Claude's rewrite capabilities, provide clear and specific instructions that outline your expectations for the revised content. Consider including:

- The desired format (e.g., paragraph, list, dialogue)
- The target tone or style (e.g., formal, conversational, humorous)
- Any additional information or context to include
- Specific areas of the original response to focus on or modify
- A rubric for Claude to asses the quality and accuracy of its previous response, and to understand what a high quality result entails

Here's an example of how you might request a rewrite to emphasize certain constraints:

Role	Prompt
User	Tell me about all the ways I can get fitter.
Assistant (Claude's response)	Here are some ways you can get fitter: [list-style response]
User	Rewrite this as a single paragraph of text, focusing on the most effective strategies.
Assistant (Claude's response)	If you're looking to get fitter, some of the most effective strategies include: [brief paragraph-style response]

By breaking down the prompt into sequential steps and providing targeted rewrite instructions, you can guide Claude to generate output that better meets your specific needs.

A note on fact-checking

If you're asking Claude to rewrite a response to ensure factual accuracy, consider utilizing the "I don't know" technique to prevent hallucinations. This approach encourages Claude to acknowledge when it lacks the necessary information to make a correction rather than generating a potentially inaccurate "fix."

Iterative rewrites

Sometimes, a single rewrite may not be enough to achieve the perfect output. In these cases, consider engaging in an iterative process, requesting multiple revisions based on

evolving criteria. With each round of feedback, Claude can learn more about your preferences and refine its responses accordingly.

Remember, the key to successful rewrites is providing clear, actionable guidance. The more specific your instructions, the more likely Claude is to generate content that aligns with your vision.

Long context window tips

Claude's extended context window (200K tokens for Claude 3 models) enables it to handle complex tasks that require processing large amounts of information. Claude's extended context window also enables you to simplify workflows that previously required splitting inputs to fit within shorter context windows. By combining inputs into a single prompt, you can streamline your process and take full advantage of Claude's capabilities.

For example, if your previous application required splitting a long document into multiple parts and processing each part separately, you can now provide the entire document to Claude in a single prompt. This not only simplifies your code but also allows Claude to have a more comprehensive understanding of the context, potentially leading to better results.

Looking for general prompt engineering techniques? Check out our prompt engineering guide.

Structuring long documents

When working with long documents (particularly 30K+ tokens), it's essential to structure your prompts in a way that clearly separates the input data from the instructions. We recommend using XML tags to encapsulate each document. This structure is how Claude was trained to take long documents, and is thus the structure that Claude is most familiar with:

Here are some documents for you to reference for your task:

```
<documents>
<document index="1">
<source>
(a unique identifying source for this item - could be a URL, file name, hash, etc)
</source>
<document_content>
(the text content of the document - could be a passage, web page, article, etc)
```

```
</document_content>
</document>
<document index="2">
<source>
(a unique identifying source for this item - could be a URL, file name, hash, etc)
</source>
<document_content>
(the text  content of the document - could be a passage, web page, article, etc)
</document_content>
</document>
...
</documents>

[Rest of prompt]
```

This structure makes it clear to Claude which parts of the prompt are input data and which are instructions, improving its ability to process the information accurately. You can also add tags to house other metadata, such as <title> or <author>.

Document-query placement

Notice in the above example of long document prompt structure that the documents come first and the rest of the prompt comes after. For situations with long documents or a lot of additional background content, Claude generally performs noticeably better if the documents and additive material are placed up top, above the detailed instructions or user query.

This is true of all Claude models, from legacy models to the Claude 3 family.

Tips for document q&a

When using Claude for document question-answering tasks, keep these tips in mind:

- Place the question at the end of the prompt, after the input data. As mentioned, this has been shown to significantly improve the quality of Claude's responses.
- Ask Claude to find quotes relevant to the question before answering, and to only answer if it finds relevant quotes. This encourages Claude to ground its responses in the provided context and reduces hallucination risk.

- Instruct Claude to read the document carefully, as it will be asked questions later. This primes Claude to pay close attention to the input data with an eye for the task it will be asked to execute.

Here's an example prompt that incorporates these tips:

Role	Content
User	I'm going to give you a document. Read the document carefully, because I'm going to ask you a question about it. Here is the document: <document>{{TEXT}}</document> First, find the quotes from the document that are most relevant to answering the question, and then print them in numbered order in <quotes></quotes> tags. Quotes should be relatively short. If there are no relevant quotes, write "No relevant quotes" instead. Then, answer the question in <answer></answer> tags. Do not include or reference quoted content verbatim in the answer. Don't say "According to Quote [1]" when answering. Instead make references to quotes relevant to each section of the answer solely by adding their bracketed numbers at the end of relevant sentences. Thus, the format of your overall response should look like what's shown between the <examples></examples> tags. Make sure to follow the formatting and spacing exactly. <examples> [Examples of question + answer pairs, with answers written exactly like how Claude's output should be structured] </examples> If the question cannot be answered by the document, say so. Here is the first question: {{QUESTION}}

Multiple choice question generation

When using Claude to generate multiple choice questions based on a given text, providing example question-answer pairs from other parts of the same text can significantly improve the quality of the generated questions. It's important to note that generic multiple choice examples based on external knowledge or generated from an unrelated document do not seem to be nearly as effective.

Here's an example prompt for multiple choice question generation:

Role Content

Your task is to generate multiple choice questions based on content from the following document:
<document>
{{DOCUMENT}}
</document>

Here are some example multiple choice questions and answers based on other parts of the text:

<examples>
Q1: [Example question 1, created from information within the document]
A. [Answer option A]
B. [Answer option B]
C. [Answer option C]
D. [Answer option D]
User Answer: [Correct answer letter]

Q2: [Example question 2, created from information within the document]
A. [Answer option A]
B. [Answer option B]
C. [Answer option C]
D. [Answer option D]
Answer: [Correct answer letter]
</examples>

Instructions:
1. Generate 5 multiple choice questions based on the provided text.
2. Each question should have 4 answer options (A, B, C, D).
3. Indicate the correct answer for each question.
4. Make sure the questions are relevant to the text and the answer options are all plausible.

By providing example questions and answers from the same text, you give Claude a better understanding of the desired output format and the types of questions that can be generated from the given content.

For more information on this specific task, see Anthropic's blog post Prompt engineering for a long context window.

Helper metaprompt (experimental)

Sometimes, the hardest part of using an AI model is figuring out how to prompt it effectively. To help with this, we've created an experimental helper "meta"-prompt that can guide Claude to generate a high-quality prompts tailored to your specific tasks.

To use the metaprompt, follow the instructions in our metaprompt Google Colab notebook. There, you can easily run the code to have Claude construct prompts on your behalf.

Note that to run the Colab notebook, you will need an API key.

The metaprompt is particularly useful as a "getting started" tool or as a method to generate multiple prompt versions for a given task, making it easier to test a variety of initial prompt variations for your use case.
