Starting April 29, 2025, Gemini 1.5 Pro and Gemini 1.5 Flash models are not available in projects that have no prior usage of these models, including new projects. For details, see Model versions and lifecycle (/vertex-ai/generative-ai/docs/learn/model-versions#legacy-stable).

Overview of prompting strategies

To see an example of prompt design, run the "Intro to Prompt Design" notebook in one of the following environments:

Open in Colab (https://colab.research.google.com/github/GoogleCloudPlatform/generative-ai/blob/main/gemini/prompts/intro_prompt_design.ipynb)

|

Open in Colab Enterprise (https://console.cloud.google.com/vertex-ai/colab/import/https%3A%2F%2Fraw.githubusercontent.com%2FGoogleCloudPlatform%2Fgenerative-ai%2Fmain%2Fgemini%2Fprompts%2Fintro_prompt_design.ipynb)

|

Open in Vertex AI Workbench (https://console.cloud.google.com/vertex-ai/workbench/deploy-notebook?download_url=https%3A%2F%2Fraw.githubusercontent.com%2FGoogleCloudPlatform%2Fgenerative-ai%2Fmain%2Fgemini%2Fprompts%2Fintro_prompt_design.ipynb)

|

View on GitHub (https://github.com/GoogleCloudPlatform/generative-ai/blob/main/gemini/prompts/intro_prompt_design.ipynb)

While there's no right or wrong way to design a prompt, there are common strategies that you can use to affect the model's responses. Rigorous testing and evaluation remain crucial for optimizing model performance.

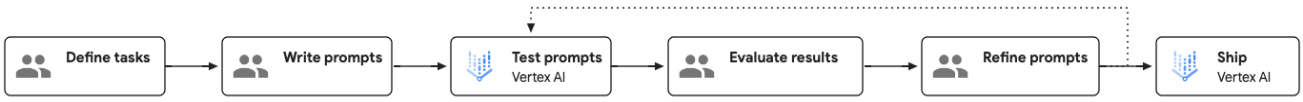
Large language models (LLM) are trained on vast amounts of text data to learn the patterns and relationships between units of language. When given some text (the prompt), language models can predict what is likely to come next, like a sophisticated autocompletion tool. Therefore, when

designing prompts, consider the different factors that can influence what a model predicts comes next.

Prompt engineering workflow

Prompt engineering is a test-driven and iterative process that can enhance model performance. When creating prompts, it is important to clearly define the objectives and expected outcomes for each prompt and systematically test them to identify areas of improvement.

The following diagram shows the prompt engineering workflow:



How to create an effective prompt

There are two aspects of a prompt that ultimately affect its effectiveness: *content* and *structure*.

- **Content:**

  In order to complete a task, the model needs all of the relevant information associated with the task. This information can include instructions, examples, contextual information, and so on. For details, see Components of a prompt (#components-of-a-prompt).

- **Structure:**

  Even when all the required information is provided in the prompt, giving the information structure helps the model parse the information. Things like the ordering, labeling, and the use of delimiters can all affect the quality of responses. For an example of prompt structure, see Sample prompt template (#sample-prompt-template).

Components of a prompt

The following table shows the essential and optional components of a prompt:

Component	Description	Example
Objective	What you want the model to achieve. Be specific and include any overarching objectives. Also called "mission" or "goal."	Your objective is to help students with math problems without directly giving them the answer.
Instructions	Step-by-step instructions on how to perform the task at hand. Also called "task," "steps," or "directions."	1. Understand what the problem is asking. 2. Understand where the student is stuck. 3. Give a hint for the next step of the problem.
Optional components		
System instructions	Technical or environmental directives that may involve controlling or altering the model's behavior across a set of tasks. For many model APIs, system instructions are specified in a dedicated parameter. System instructions are available in Gemini 2.0 Flash and later models.	You are a coding expert that specializes in rendering code for front-end interfaces. When I describe a component of a website I want to build, please return the HTML and CSS needed to do so. Do not give an explanation for this code. Also offer some UI design suggestions.
Persona	Who or what the model is acting as. Also called "role" or "vision."	You are a math tutor here to help students with their math homework.
Constraints	Restrictions on what the model must adhere to when generating a response, including what the model can and can't do. Also called "guardrails," "boundaries," or "controls."	Don't give the answer to the student directly. Instead, give hints at the next step towards solving the problem. If the student is completely lost, give them the detailed steps to solve the problem.
Tone	The tone of the response. You can also influence the style and tone by specifying a persona. Also called "style," "voice," or "mood."	Respond in a casual and technical manner.
Context	Any information that the model needs to refer to in order to perform the task at hand. Also called "background," "documents," or "input data."	A copy of the student's lesson plans for math.
Few-shot examples	Examples of what the response should look like for a given prompt. Also called "exemplars" or "samples."	**`input:`** I'm trying to calculate how many golf balls can fit into a box that has a one cubic meter volume. I've converted one cubic meter into cubic centimeters and divided it by the volume of a golf ball in cubic centimeters, but the system says my answer is wrong. **`output:`** Golf balls are spheres and cannot be packed into a space with perfect efficiency. Your calculations take into account the maximum packing efficiency of spheres.

Component	Description	Example
Reasoning steps	Tell the model to explain its reasoning. This can sometimes improve the model's reasoning capability. Also called "thinking steps."	Explain your reasoning step-by-step.
Response format	The format that you want the response to be in. For example, you can tell the model to output the response in JSON, table, Markdown, paragraph, bulleted list, keywords, elevator pitch, and so on. Also called "structure," "presentation," or "layout."	Format your response in Markdown.
Recap	Concise repeat of the key points of the prompt, especially the constraints and response format, at the end of the prompt.	Don't give away the answer and provide hints instead. Always format your response in Markdown format.
Safeguards	Grounds the questions to the mission of the bot. Also called "safety rules."	N/A

Depending on the specific tasks at hand, you might choose to include or exclude some of the optional components. You can also adjust the ordering of the components and check how that can affect the response.

Sample prompt template

The following prompt template shows you an example of what a well-structured prompt might look like:

prompt template:

```
:CTIVE_AND_PERSONA>
ıre a [insert a persona, such as a "math teacher" or "automotive expert"]. Your task :

)ECTIVE_AND_PERSONA>

TRUCTIONS>
›mplete the task, you need to follow these steps:



STRUCTIONS>
```

```
-------- Optional Components ------------

STRAINTS>
nd don'ts for the following aspects
s
n'ts
STRAINTS>

EXT>
rovided context
TEXT>

UT_FORMAT>
utput format must be



PUT_FORMAT>

_SHOT_EXAMPLES>
 we provide some examples:
ample #1
::
hts:
t:

_SHOT_EXAMPLES>

P>
phasize the key aspects of the prompt, especially the constraints, output format, etc
AP>
```

Best practices

Prompt design best practices include the following:

- Give clear and specific instructions (/vertex-ai/generative-ai/docs/learn/prompts/clear-instructions)
- Include few-shot examples (/vertex-ai/generative-ai/docs/learn/prompts/few-shot-examples)
- Assign a role (/vertex-ai/generative-ai/docs/learn/prompts/assign-role)
- Add contextual information (/vertex-ai/generative-ai/docs/learn/prompts/contextual-information)

- Use system instructions (/vertex-ai/generative-ai/docs/learn/prompts/system-instructions)
- Structure prompts (/vertex-ai/generative-ai/docs/learn/prompts/structure-prompts)
- Instruct the model to explain its reasoning (/vertex-ai/generative-ai/docs/learn/prompts/explain-reasoning)
- Break down complex tasks (/vertex-ai/generative-ai/docs/learn/prompts/break-down-prompts)
- Experiment with parameter values (/vertex-ai/generative-ai/docs/learn/prompts/adjust-parameter-values)
- Prompt iteration strategies (/vertex-ai/generative-ai/docs/learn/prompts/prompt-iteration)

Prompt health checklist

If a prompt is not performing as expected, use the following checklist to identify potential issues and improve the prompt's performance.

Writing issues

- **Typos:** Check keywords that define the task (for example, *sumarize* instead of *summarize*), technical terms, or names of entities, as misspellings can lead to poor performance.
- **Grammar:** If a sentence is difficult to parse, contains run-on fragments, has mismatched subjects and verbs, or feels structurally awkward, the model may not properly understand the prompt.
- **Punctuation:** Check your use of commas, periods, quotes, and other separators, as incorrect punctuation can cause the model to misinterpret the prompt.
- **Use of undefined jargon:** Avoid using domain-specific terms, acronyms, or initialisms as if they have a universal meaning unless they are explicitly defined in the prompt.
- **Clarity:** If you find yourself wondering about the scope, the specific steps to take, or the implicit assumptions being made, the prompt is likely unclear.
- **Ambiguity:** Avoid using subjective or relative qualifiers that lack a concrete, measurable definition. Instead, provide objective constraints (for example, "write a summary of 3 sentences or less" instead of "write a brief summary").
- **Missing key information:** If the task requires knowledge of a specific document, company policy, user history, or dataset, make sure that information is explicitly included within the prompt.

- **Poor word choice:** Check the prompt for unnecessarily complex, vague, or verbose phrasing, as it could confuse the model.
- **Secondary review:** If the model continues to perform poorly, have another person review your prompt.

Issues with instructions and examples

- **Overt manipulation:** Remove language outside of the core task from the prompt that attempts to influence performance using emotional appeals, flattery, or artificial pressure. While first generation foundation models showed improvement in some circumstances with instructions like "very bad things will happen if you don't get this correct", foundation model performance will no longer improve and in many cases will get worse.
- **Conflicting instructions and examples:** Check for this by auditing the prompt for logical contradictions or mismatches between instructions or an instruction and an example.
- **Redundant instructions and examples:** Look through the prompt and examples to see if the exact same instruction or concept is stated multiple times in slightly different ways without adding new information or nuance.
- **Irrelevant instructions and examples:** Check to see if all of the instructions and examples are essential to the core task. If any instructions or examples can be removed without diminishing the model's ability to perform the core task, they might be irrelevant.
- **Use of "few-shot"** (/vertex-ai/generative-ai/docs/learn/prompts/few-shot-examples) **examples:** If the task is complex, requires a specific format, or has a nuanced tone, make sure there are concrete, illustrative examples that show a sample input and the corresponding output.
- **Missing output format specification:** Avoid leaving the model to guess the structure of the output; instead, use a clear, explicit instruction to specify the format and show the output structure in your few-shot examples.
- **Missing role definition:** If you are going to ask the model to act in a specific role, make sure that role is defined in the system instructions.

Prompt and system design issues

- **Underspecified task:** Ensure that the prompt's instructions provide a clear path for handling edge cases and unexpected inputs, and provide instructions for handling missing data rather than assuming inserted data will always be present and well-formed.

- **Task outside of model capabilities:** Avoid using prompts that ask the model to perform a task for which it has a known, fundamental limitation.
- **Too many tasks:** If the prompt asks the model to perform several distinct cognitive actions in a single pass (for example, 1. Summarize, 2. Extract entities, 3. Translate, and 4. Draft an email), it is likely trying to accomplish too much. Break the requests into separate prompts.
- **Non-standard data format:** When model outputs must be machine-readable or follow a specific format, use a widely recognized standard like JSON, XML, Markdown or YAML that can be parsed by common libraries. If your use case requires a non-standard format, consider asking the model to output to a common format and then using code to convert the output.
- **Incorrect Chain of Thought (CoT) order:** Avoid providing examples that show the model generating its final, structured answer before it has completed its step-by-step reasoning.
- **Conflicting internal references:** Avoid writing a prompt with non-linear logic or conditionals that require the model to piece together fragmented instructions from multiple different places in the prompt.
- **Prompt injection risk:** Check if there are explicit safeguards surrounding untrusted user input that is inserted into the prompt, as this can be a major security risk.

What's next

- Explore examples of prompts in the Prompt gallery (/vertex-ai/generative-ai/docs/prompt-gallery).
- Learn how to optimize prompts for use with Google models (/vertex-ai/generative-ai/docs/learn/models) by using the Vertex AI prompt optimizer (Preview) (/vertex-ai/generative-ai/docs/learn/prompts/prompt-optimizer).
- Learn about responsible AI best practices and Vertex AI's safety filters (/vertex-ai/generative-ai/docs/learn/responsible-ai).

Last updated 2025-07-30 UTC.