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).

Break down complex tasks into simpler prompts

For complex tasks that require multiple instructions or steps, you can improve the model's responses by breaking your prompts into subtasks. Smaller prompts can help you improve controllability, debugging, and accuracy.

There are two ways to break down complex prompts and ingest them into a model:

- **Chain prompts:** split a task into subtasks and run the subtaks sequentially.
- **Aggregate responses:** split a task into subtasks and run the subtasks in parallel.

Chain prompts

For complex tasks that involve multiple *sequential* steps, make each step a prompt and chain the prompts together in a sequence. In this sequential chain of prompts, the output of one prompt in the sequence becomes the input of the next prompt. The output of the last prompt in the sequence is the final output.

Example

For example, suppose you run a telecommunications business and want to use a model to help you analyze customer feedback to identify common customer issues, classify issues into categories, and generate solutions for categories of issues.

Task 1: identify customer issues

The first task you want the model to complete is extracting meaningful data from raw customer feedback. A prompt that achieves this task might be similar to the following, where *`CUSTOMER_FEEDBACK`* is a file that contains the customer feedback:

t data

:

```
act the main issues and sentiments from the customer feedback on our telecom services
s on comments related to service disruptions, billing issues, and customer support
actions.
se format the output into a list with each issue/sentiment in a sentence, separated by
colon.

: CUSTOMER_FEEDBACK
```

We would expect the model's response to contain a list of extracted issues and sentiment from the customer feedback.

Task 2: classify issues into categories

Next, you want to prompt the model to classify the data into categories so that you can understand the types of issues customers face, using the response from the previous task. A prompt that achieves this task might look similar to the following, where *TASK_1_RESPONSE* is the response from the previous task:

fy data

:

```
sify the extracted issues into categories such as service reliability, pricing concer
omer support quality, and others.
se organize the output into JSON format with each issue as the key, and category as th
.

: TASK_1_RESPONSE
```

We would expect the model's response to contain categorized issues.

Task 3: generate solutions

Now, you want to prompt the model to generate actionable recommendations based on the categorized issues to improve customer satisfaction, using the response from the previous task. A prompt that achieves this might look similar to the following, where *`TASK_2_RESPONSE`* is the response from the previous task:

ate suggestions

:

```
ate detailed recommendations for each category of issues identified from the feedbacl
est specific actions to address service reliability, improving customer support, and
sting pricing models, if necessary.
e organize the output into a JSON format with each category as the key, and
mendation as the value.

: TASK_2_RESPONSE
```

We would expect the model's response to contain recommendations for each category, aimed at improving customer experience and service quality, which satifies our overall objective.

Aggregate responses

In cases where you have complex tasks but you don't need to perform the tasks in a specific order, you can run parallel prompts and aggregate the model's responses.

Example

For example, suppose you own a record store and want to use a model to help you decide which records to stock based on music streaming trends and your store's sales data.

Task 1: analyze data

The first thing you need to do is analyze the two datasets, streaming data and sales data. You can run the prompts to complete these tasks in parallel. Prompts that achieve these tasks might be similar to the following, where *`STORE_SALES_DATA`* is a file that contains the sales data and *`STREAMING_DATA`* is a file that contains the streaming data:

a: analyze sales data

:

```
yze the sales data to identify the number of sales of each record.
se organize the output into a JSON format with each record as the key, and sales as th
e.

:: STORE_SALES_DATA
```

We would expect the output to contain the number of sales for each record, formatted in JSON.

b: analyze streaming data

:

```
yze the streaming data to provide a the number of streams for each album.
se organize the output into a JSON format with each album as the key, and streams as t
e.

:: STREAMING_DATA
```

We would expect the output to contain the number of streams for each album, formatted in JSON.

Task 2: aggregate data

Now you can aggregate the data from both datasets to help you plan your purchasing decisions. To aggregate the data, include the output from both tasks as the input. A prompt that achieves this might look similar to the following, where *TASK_1A_RESPONSE* and *TASK_1B_RESPONSE* are the responses from the previous tasks:

gate sales and streaming data

:

```
ımend a stocklist of about 20 records based on the most sold and most streamed records
ıly three quarters of the stock list should be based on record sales, and the rest on
ıming.
```

`:: TASK_1A_RESPONSE and TASK_1B_RESPONSE`

We would expect the output to contain a suggested stocklist of about 20 records, based on record sales and streams, with more favor given to records with proven sales history than to those with more streaming popularity.

What's next

- Explore examples of prompts in the Prompt gallery (/vertex-ai/generative-ai/docs/prompt-gallery).

Last updated 2025-07-30 UTC.