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).

Structured output

Release Notes

To see an example of structured output, run the "Intro to structured output" notebook in one of the following environments:

Open in Colab (https://colab.research.google.com/github/GoogleCloudPlatform/generative-ai/blob/main/gemini/controlled-generation/intro_controlled_generation.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%2Fcontrolled-generation%2Fintro_controlled_generation.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%2Fcontrolled-generation%2Fintro_controlled_generation.ipynb)

|

View on GitHub (https://github.com/GoogleCloudPlatform/generative-ai/blob/main/gemini/controlled-generation/intro_controlled_generation.ipynb)

You can guarantee that a model's generated output always adheres to a specific schema so that you receive consistently formatted responses. For example, you might have an established data schema that you use for other tasks. If you have the model follow the same schema, you can directly extract data from the model's output without any post-processing.

To specify the structure of a model's output, define a *response schema*, which works like a blueprint for model responses. When you submit a prompt and include the response schema (/vertex-ai/docs/reference/rest/v1/projects.locations.cachedContents#Schema), the model's response always follows your defined schema.

You can control generated output when using the following models:

- Gemini models:
  - Vertex AI Model Optimizer (/vertex-ai/generative-ai/docs/model-reference/vertex-ai-model-optimizer)
  - Gemini 2.5 Pro (/vertex-ai/generative-ai/docs/models/gemini/2-5-pro)
  - Gemini 2.5 Flash (/vertex-ai/generative-ai/docs/models/gemini/2-5-flash)
  - Gemini 2.0 Flash (/vertex-ai/generative-ai/docs/models/gemini/2-0-flash)
  - Gemini 2.0 Flash-Lite (/vertex-ai/generative-ai/docs/models/gemini/2-0-flash-lite)
- Open models:
  - DeepSeek R1-0528 (/vertex-ai/generative-ai/docs/maas/deepseek/r1-0528)
  - Llama 4 Maverick (/vertex-ai/generative-ai/docs/partner-models/llama/llama4-maverick)
  - Llama 4 Scout (/vertex-ai/generative-ai/docs/partner-models/llama/llama4-scout)
  - Llama 3.3 (/vertex-ai/generative-ai/docs/partner-models/llama/llama3-3)

For Open Models, follow this user guide (/vertex-ai/generative-ai/docs/maas/capabilities/structured-output).

Note: Using structured output on tuned Gemini models (/vertex-ai/generative-ai/docs/models/gemini-supervised-tuning) can result in decreased model quality (/vertex-ai/generative-ai/docs/models/gemini-supervised-tuning#known_issues).

Example use cases

One use case for applying a response schema is to ensure that a model's response produces valid JSON and conforms to your schema. Generative model outputs can have some degree of variability, so including a response schema ensures that you always receive valid JSON. Consequently, your downstream tasks can reliably expect valid JSON input from generated responses.

Another example is to constrain how a model can respond. For example, you can have a model annotate text with user-defined labels, not with labels that the model produces. This constraint is useful when you expect a specific set of labels such as `positive` or `negative` and don't want to receive a mixture of other labels that the model might generate like `good`, `positive`, `negative`, or `bad`.

Considerations

The following considerations discuss potential limitations if you plan on using a response schema:

- You must use the API to define and use a response schema. There's no console support.
- The size of your response schema counts towards the input token limit.
- Only certain output formats are supported, such as `application/json` or `text/x.enum`. For more information, see the `responseMimeType` parameter in the Gemini API reference (/vertex-ai/generative-ai/docs/model-reference/inference#parameters).
- Structured output supports a subset of the Vertex AI schema reference (/vertex-ai/docs/reference/rest/v1/projects.locations.cachedContents#Schema). For more information, see Supported schema fields (#fields).
- A complex schema can result in an `InvalidArgument: 400` error. Complexity might come from long property names, long array length limits, enums with many values, objects with lots of optional properties, or a combination of these factors.

  If you get this error with a valid schema, make one or more of the following changes to resolve the error:

  - Shorten property names or enum names.
  - Flatten nested arrays.
  - Reduce the number of properties with constraints, such as numbers with minimum and maximum limits.
  - Reduce the number of properties with complex constraints, such as properties with complex formats like `date-time`.
  - Reduce the number of optional properties.
  - Reduce the number of valid values for enums.

Supported schema fields

Structured output supports the following fields from the Vertex AI schema (/vertex-ai/docs/reference/rest/v1/projects.locations.cachedContents#Schema). If you use an unsupported field, Vertex AI can still handle your request but ignores the field.

- `anyOf`
- `enum`: only `string` enums are supported
- `format`
- `items`
- `maximum`
- `maxItems`
- `minimum`
- `minItems`
- `nullable`
- `properties`
- `propertyOrdering`* (#note)
- `required`

* `propertyOrdering` is specifically for structured output and not part of the Vertex AI schema. This field defines the order in which properties are generated. The listed properties must be unique and must be valid keys in the `properties` dictionary.

For the `format` field, Vertex AI supports the following values: `date`, `date-time`, `duration`, and `time`. The description and format of each value is described in the OpenAPI Initiative Registry (https://spec.openapis.org/registry/format/)

Before you begin

Define a response schema to specify the structure of a model's output, the field names, and the expected data type for each field. Use only the supported fields as listed in the Considerations (#considerations) section. All other fields are ignored.

Include your response schema as part of the `responseSchema` field only. Don't duplicate the schema in your input prompt. If you do, the generated output might be lower in quality.

For sample schemas, see the Example schemas and model responses (#examples) section.

Model behavior and response schema

When a model generates a response, it uses the field name and context from your prompt. As such, we recommend that you use a clear structure and unambiguous field names so that your intent is clear.

By default, fields are optional, meaning the model can *populate* the fields or *skip* them. You can set fields as required to force the model to provide a value. If there's insufficient context in the associated input prompt, the model generates responses mainly based on the data it was trained on.

If you aren't seeing the results you expect, add more context to your input prompts or revise your response schema. For example, review the model's response without structured output to see how the model responds. You can then update your response schema that better fits the model's output.

Send a prompt with a response schema

To see an example of a response schema and structured output, run the "Introduction to structured output" notebook in one of the following environments:

Open in Colab (https://colab.research.google.com/github/GoogleCloudPlatform/generative-ai/blob/main/gemini/controlled-generation/intro_controlled_generation.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%2Fcontrolled-generation%2Fintro_controlled_generation.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%2Fcontrolled-generation%2Fintro_controlled_generation.ipynb)

|

View on GitHub
(https://github.com/GoogleCloudPlatform/generative-ai/blob/main/gemini/controlled-generation/intro_controlled_generation.ipynb)

By default, all fields are optional, meaning a model might generate a response to a field. To force the model to always generate a response to a field, set the field as required.

Python Gen AI SDK | Go Gen AI SDK | REST

Before using any of the request data, make the following replacements:

- ***GENERATE_RESPONSE_METHOD***: The type of response that you want the model to generate. Choose a method that generates how you want the model's response to be returned:
  - `streamGenerateContent`: The response is streamed as it's being generated to reduce the perception of latency to a human audience.
  - `generateContent`: The response is returned after it's fully generated.
- ***LOCATION***: The region to process the request.
- ***PROJECT_ID***: Your project ID (/resource-manager/docs/creating-managing-projects#identifiers).
- ***MODEL_ID***: The model ID of the multimodal model that you want to use.
- ***ROLE***: The role in a conversation associated with the content. Specifying a role is required even in singleturn use cases. Acceptable values include the following:
  - `USER`: Specifies content that's sent by you.
- ***TEXT***: The text instructions to include in the prompt.
- ***RESPONSE_MIME_TYPE***: The format type of the generated candidate text. For a list of supported values, see the `responseMimeType` parameter in the Gemini API (/vertex-ai/generative-ai/docs/model-reference/inference).
- ***RESPONSE_SCHEMA***: Schema for the model to follow when generating responses. For more information, see the Schema (/vertex-ai/docs/reference/rest/v1/projects.locations.cachedContents#Schema) reference.

HTTP method and URL:

```
POST https://LOCATION-aiplatform.googleapis.com/v1/projects/PROJECT_ID/location
```

Request JSON body:

```
{
  "contents": {
    "role": "ROLE",
    "parts": {
      "text": "TEXT"
    }
  },
  "generation_config": {
    "responseMimeType": "RESPONSE_MIME_TYPE",
    "responseSchema": RESPONSE_SCHEMA,
  }
}
```

To send your request, choose one of these options:

curlPowerShell (#powershell)
(#curl)

★ **Note:** The following command assumes that you have logged in to the **gcloud** CLI with your user account by running **gcloud init** (/sdk/gcloud/reference/init) or **gcloud auth login** (/sdk/gcloud/reference/auth/login) , or by using Cloud Shell (/shell/docs), which automatically logs you into the **gcloud** CLI . You can check the currently active account by running **gcloud auth list** (/sdk/gcloud/reference/auth/list).

Save the request body in a file named **request.json**, and execute the following command:

```
curl -X POST \
     -H "Authorization: Bearer $(gcloud auth print-access-token)" \
     -H "Content-Type: application/json; charset=utf-8" \
```

```
-d @request.json \
"https://LOCATION-aiplatform.googleapis.com/v1/projects/PROJECT_ID/l
```

You should receive a JSON response similar to the following.

Response


```
{
  "candidates": [
    {
      "content": {
        "role": "model",
        "parts": [
          {
            "text": "[{\"recipe_name\": \"Chocolate Chip Cookies\"}, {\"recipe_
          }
        ]
      },
      "finishReason": "STOP",
      "safetyRatings": [
        {
          "category": "HARM_CATEGORY_HATE_SPEECH",
          "probability": "NEGLIGIBLE",
          "probabilityScore": 0.08021325,
          "severity": "HARM_SEVERITY_NEGLIGIBLE",
          "severityScore": 0.0921962
        },
        {
          "category": "HARM_CATEGORY_DANGEROUS_CONTENT",
          "probability": "NEGLIGIBLE",
          "probabilityScore": 0.14730969,
          "severity": "HARM_SEVERITY_NEGLIGIBLE",
          "severityScore": 0.08866235
        },
        {
          "category": "HARM_CATEGORY_HARASSMENT",
          "probability": "NEGLIGIBLE",
          "probabilityScore": 0.13432105,
          "severity": "HARM_SEVERITY_NEGLIGIBLE",
          "severityScore": 0.07172113
        },
        {
          "category": "HARM_CATEGORY_SEXUALLY_EXPLICIT",
```

```
          "probability": "NEGLIGIBLE",
          "probabilityScore": 0.12787028,
          "severity": "HARM_SEVERITY_NEGLIGIBLE",
          "severityScore": 0.10017223
        }
      ]
    }
  ],
  "usageMetadata": {
    "promptTokenCount": 7,
    "candidatesTokenCount": 55,
    "totalTokenCount": 62
  }
}
```

Example curl command

```
LOCATION="us-central1"
MODEL_ID="gemini-2.5-flash"
PROJECT_ID="test-project"
GENERATE_RESPONSE_METHOD="generateContent"

cat << EOF > request.json
{
  "contents": {
    "role": "user",
    "parts": {
      "text": "List a few popular cookie recipes."
    }
  },
  "generation_config": {
    "maxOutputTokens": 2048,
    "responseMimeType": "application/json",
    "responseSchema": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "recipe_name": {
            "type": "string",
          },
        },
        "required": ["recipe_name"],
      },
```

```
    }
  }
}
EOF

curl \
-X POST \
-H "Authorization: Bearer $(gcloud auth print-access-token)" \
-H "Content-Type: application/json" \
https://${LOCATION}-aiplatform.googleapis.com/v1/projects/${PROJECT_ID}/locatio
-d '@request.json'
```

Example schemas for JSON output

The following sections demonstrate a variety of sample prompts and response schemas. A sample model response is also included after each code sample.

- Forecast the weather for each day of the week in an array (#forecast)
- Classify a product with a well-defined enum (#classify)

Forecast the weather for each day of the week

The following example outputs a `forecast` object for each day of the week that includes an array of properties such as the expected temperature and humidity level for the day. Some properties are set to nullable so the model can return a null value when it doesn't have enough context to generate a meaningful response. This strategy helps reduce hallucinations.

Python Gen AI SDK | Node.js Gen AI SDK | Go Gen AI SDK

Install

```
pip install --upgrade google-genai
```

To learn more, see the SDK reference documentation (https://googleapis.github.io/python-genai/).

Set environment variables to use the Gen AI SDK with Vertex AI:

```
# Replace the `GOOGLE_CLOUD_PROJECT` and `GOOGLE_CLOUD_LOCATION` values
# with appropriate values for your project.
export GOOGLE_CLOUD_PROJECT=GOOGLE_CLOUD_PROJECT
export GOOGLE_CLOUD_LOCATION=global
export GOOGLE_GENAI_USE_VERTEXAI=True
```

```
from google import genai
from google.genai.types import GenerateContentConfig, HttpOptions

response_schema = {
    "type": "OBJECT",
    "properties": {
        "forecast": {
            "type": "ARRAY",
            "items": {
                "type": "OBJECT",
                "properties": {
                    "Day": {"type": "STRING", "nullable": True},
                    "Forecast": {"type": "STRING", "nullable": True},
                    "Temperature": {"type": "INTEGER", "nullable": True},
                    "Humidity": {"type": "STRING", "nullable": True},
                    "Wind Speed": {"type": "INTEGER", "nullable": True},
                },
                "required": ["Day", "Temperature", "Forecast", "Wind Speed"],
            },
        }
    },
}

prompt = """
    The week ahead brings a mix of weather conditions.
    Sunday is expected to be sunny with a temperature of 77°F and a humidity le
    Monday will see partly cloudy skies with a slightly cooler temperature of 7
    Tuesday brings rain showers, with temperatures dropping to 64°F and humidit
    Wednesday may see thunderstorms, with a temperature of 68°F.
    Thursday will be cloudy with a temperature of 66°F and moderate humidity at
    Friday returns to partly cloudy conditions, with a temperature of 73°F and
    Finally, Saturday rounds off the week with sunny skies, a temperature of 80
"""

client = genai.Client(http_options=HttpOptions(api_version="v1"))
response = client.models.generate_content(
    model="gemini-2.5-flash",
```

```
    contents=prompt,
    config=GenerateContentConfig(
        response_mime_type="application/json",
        response_schema=response_schema,
    ),
)

print(response.text)
# Example output:
# {"forecast": [{"Day": "Sunday", "Forecast": "sunny", "Temperature": 77, "Wind
#   {"Day": "Monday", "Forecast": "partly cloudy", "Temperature": 72, "Wind Spe
#   {"Day": "Tuesday", "Forecast": "rain showers", "Temperature": 64, "Wind Spe
#   {"Day": "Wednesday", "Forecast": "thunderstorms", "Temperature": 68, "Wind
#   {"Day": "Thursday", "Forecast": "cloudy", "Temperature": 66, "Wind Speed":
#   {"Day": "Friday", "Forecast": "partly cloudy", "Temperature": 73, "Wind Spe
#   {"Day": "Saturday", "Forecast": "sunny", "Temperature": 80, "Wind Speed": 8
```

Classify a product

The following example includes enums where the model must classify an object's type and condition from a list of given values.

Python Gen AI SDK | Python Gen AI SDK | Go Gen AI SDK | Node.js Gen AI SDK | Java Gen AI SDK

Install

```
pip install --upgrade google-genai
```

To learn more, see the SDK reference documentation (https://googleapis.github.io/python-genai/).

Set environment variables to use the Gen AI SDK with Vertex AI:

```
# Replace the `GOOGLE_CLOUD_PROJECT` and `GOOGLE_CLOUD_LOCATION` values
# with appropriate values for your project.
export GOOGLE_CLOUD_PROJECT=GOOGLE_CLOUD_PROJECT
export GOOGLE_CLOUD_LOCATION=global
export GOOGLE_GENAI_USE_VERTEXAI=True
```

```
from google import genai
from google.genai.types import GenerateContentConfig, HttpOptions

client = genai.Client(http_options=HttpOptions(api_version="v1"))
response = client.models.generate_content(
    model="gemini-2.5-flash",
    contents="What type of instrument is an oboe?",
    config=GenerateContentConfig(
        response_mime_type="text/x.enum",
        response_schema={
            "type": "STRING",
            "enum": ["Percussion", "String", "Woodwind", "Brass", "Keyboard"],
        },
    ),
)

print(response.text)
# Example output:
# Woodwind
```


Last updated 2025-07-26 UTC.