Veo 3 is now available in the Gemini API!

Learn more (https://developers.googleblog.com/en/veo-3-now-available-gemini-api/)

Text generation

The Gemini API can generate text output from various inputs, including text, images, video, and audio, leveraging Gemini models.

Here's a basic example that takes a single text input:

Python JavaScript (#javascript) Go (#go) REST (#rest) Apps Script (#apps-script)
(#python)

```
from google import genai

client = genai.Client()

response = client.models.generate_content(
    model="gemini-2.5-flash",
    contents="How does AI work?"
)
print(response.text)
```

Thinking with Gemini 2.5

2.5 Flash and Pro models have "thinking" (/gemini-api/docs/thinking) enabled by default to enhance quality, which may take longer to run and increase token usage.

When using 2.5 Flash, you can disable thinking by setting the thinking budget to zero.

For more details, see the thinking guide (/gemini-api/docs/thinking#set-budget).

Python JavaScript (#javascript) Go (#go) REST (#rest) Apps Script (#apps-script)
(#python)

```
from google import genai
from google.genai import types

client = genai.Client()

response = client.models.generate_content(
    model="gemini-2.5-flash",
    contents="How does AI work?",
    config=types.GenerateContentConfig(
        thinking_config=types.ThinkingConfig(thinking_budget=0) # Disables
    ),
)
print(response.text)
```

System instructions and other configurations

You can guide the behavior of Gemini models with system instructions. To do so, pass a `GenerateContentConfig` (/api/generate-content#v1beta.GenerationConfig) object.

Python JavaScript (#javascript) Go (#go) REST (#rest) Apps Script (#apps-script)
(#python)

```
from google import genai
from google.genai import types

client = genai.Client()

response = client.models.generate_content(
    model="gemini-2.5-flash",
    config=types.GenerateContentConfig(
        system_instruction="You are a cat. Your name is Neko."),
    contents="Hello there"
)

print(response.text)
```

The `GenerateContentConfig` (/api/generate-content#v1beta.GenerationConfig) object also lets you override default generation parameters, such as temperature (/api/generate-content#v1beta.GenerationConfig).

Python (#python) JavaScript (#javascript) Go (#go) REST (#rest) Apps Script (#apps-script)

```
from google import genai
from google.genai import types

client = genai.Client()

response = client.models.generate_content(
    model="gemini-2.5-flash",
    contents=["Explain how AI works"],
    config=types.GenerateContentConfig(
        temperature=0.1
    )
)
print(response.text)
```

Refer to the `GenerateContentConfig` (/api/generate-content#v1beta.GenerationConfig) in our API reference for a complete list of configurable parameters and their descriptions.

Multimodal inputs

The Gemini API supports multimodal inputs, allowing you to combine text with media files. The following example demonstrates providing an image:

Python (#python) JavaScript (#javascript) Go (#go) REST (#rest) Apps Script (#apps-script)

```
from PIL import Image
from google import genai

client = genai.Client()

image = Image.open("/path/to/organ.png")
response = client.models.generate_content(
```

```
    model="gemini-2.5-flash",
    contents=[image, "Tell me about this instrument"]
)
print(response.text)
```

For alternative methods of providing images and more advanced image processing, see our image understanding guide (/gemini-api/docs/image-understanding). The API also supports document (/gemini-api/docs/document-processing), video (/gemini-api/docs/video-understanding), and audio (/gemini-api/docs/audio) inputs and understanding.

Streaming responses

By default, the model returns a response only after the entire generation process is complete.

For more fluid interactions, use streaming to receive `GenerateContentResponse` (/api/generate-content#v1beta.GenerateContentResponse) instances incrementally as they're generated.

Python JavaScript (#javascript) Go (#go) REST (#rest) Apps Script (#apps-script)
(#python)

```
from google import genai

client = genai.Client()

response = client.models.generate_content_stream(
    model="gemini-2.5-flash",
    contents=["Explain how AI works"]
)
for chunk in response:
    print(chunk.text, end="")
```

Multi-turn conversations (Chat)

Our SDKs provide functionality to collect multiple rounds of prompts and responses into a chat, giving you an easy way to keep track of the conversation history.

Note: Chat functionality is only implemented as part of the SDKs. Behind the scenes, it still uses the **`generateContent`** (/api/generate-content#method:-models.generatecontent) API. For multi-turn conversations, the full conversation history is sent to the model with each follow-up turn.

Python JavaScript (#javascript) Go (#go) REST (#rest) Apps Script (#apps-script)
(#python)

```
from google import genai

client = genai.Client()
chat = client.chats.create(model="gemini-2.5-flash")

response = chat.send_message("I have 2 dogs in my house.")
print(response.text)

response = chat.send_message("How many paws are in my house?")
print(response.text)

for message in chat.get_history():
    print(f'role - {message.role}',end=": ")
    print(message.parts[0].text)
```

Streaming can also be used for multi-turn conversations.

Python JavaScript (#javascript) Go (#go) REST (#rest) Apps Script (#apps-script)
(#python)

```
from google import genai

client = genai.Client()
chat = client.chats.create(model="gemini-2.5-flash")

response = chat.send_message_stream("I have 2 dogs in my house.")
for chunk in response:
    print(chunk.text, end="")

response = chat.send_message_stream("How many paws are in my house?")
for chunk in response:
    print(chunk.text, end="")
```

```
for message in chat.get_history():
    print(f'role - {message.role}', end=": ")
    print(message.parts[0].text)
```

Supported models

All models in the Gemini family support text generation. To learn more about the models and their capabilities, visit the Models (/gemini-api/docs/models) page.

Best practices

Prompting tips

For basic text generation, a zero-shot (/gemini-api/docs/prompting-strategies#few-shot) prompt often suffices without needing examples, system instructions or specific formatting.

For more tailored outputs:

- Use System instructions (#system-instructions) to guide the model.
- Provide few example inputs and outputs to guide the model. This is often referred to as few-shot (/gemini-api/docs/prompting-strategies#few-shot) prompting.

Consult our prompt engineering guide (/gemini/docs/prompting-strategies) for more tips.

Structured output

In some cases, you may need structured output, such as JSON. Refer to our structured output (/gemini-api/docs/structured-output) guide to learn how.

What's next

- Try the Gemini API getting started Colab (https://colab.research.google.com/github/google-gemini/cookbook/blob/main/quickstarts/Get_started.ipynb)

.

- Explore Gemini's image (/gemini-api/docs/image-understanding), video (/gemini-api/docs/video-understanding), audio (/gemini-api/docs/audio) and document (/gemini-api/docs/document-processing) understanding capabilities.
- Learn about multimodal file prompting strategies (/gemini-api/docs/files#prompt-guide).

Last updated 2025-06-27 UTC.