Gemini 2.0 Flash is now production ready!

Learn more (https://developers.googleblog.com/en/gemini-2-family-expands/)

Understand and count tokens

Python ✓ Node.js Go

Gemini and other generative AI models process input and output at a granularity called a *token*.

About tokens

Tokens can be single characters like `z` or whole words like `cat`. Long words are broken up into several tokens. The set of all tokens used by the model is called the vocabulary, and the process of splitting text into tokens is called *tokenization*.

For Gemini models, a token is equivalent to about 4 characters. 100 tokens is equal to about 60-80 English words.

When billing is enabled, the cost of a call to the Gemini API (/pricing) is determined in part by the number of input and output tokens, so knowing how to count tokens can be helpful.

Count tokens

All input to and output from the Gemini API is tokenized, including text, image files, and other non-text modalities.

You can count tokens in the following ways:

- **Call `countTokens`** (/api/rest/v1/models/countTokens) **with the input of the request.**
  This returns the total number of tokens in *the input only*. You can make this call before sending the input to the model to check the size of your requests.
- **Use the `usageMetadata` attribute on the `response` object after calling `generate_content`.**
  This returns the total number of tokens in *both the input and the output*: `totalTokenCount`.
  It also returns the token counts of the input and output separately: `promptTokenCount` (input tokens) and `candidatesTokenCount` (output tokens).

Count text tokens

If you call `countTokens` with a text-only input, it returns the token count of the text in *the input only* (`totalTokens`). You can make this call before calling `generateContent` to check the size of your requests.

Another option is calling `generateContent` and then using the `usageMetadata` attribute on the `response` object to get the following:

- The separate token counts of the input (`promptTokenCount`) and the output (`candidatesTokenCount`)
- The total number of tokens in *both the input and the output* (`totalTokenCount`)

```
// Make sure to include these imports:
// import { GoogleGenerativeAI } from "@google/generative-ai";
const genAI = new GoogleGenerativeAI(process.env.API_KEY);
const model = genAI.getGenerativeModel({
  model: "gemini-1.5-flash",
});

// Count tokens in a prompt without calling text generation.
const countResult = await model.countTokens(
  "The quick brown fox jumps over the lazy dog.",
);

console.log(countResult.totalTokens); // 11

const generateResult = await model.generateContent(
  "The quick brown fox jumps over the lazy dog.",
);

// On the response for `generateContent`, use `usageMetadata`
// to get separate input and output token counts
// (`promptTokenCount` and `candidatesTokenCount`, respectively),
// as well as the combined token count (`totalTokenCount`).
console.log(generateResult.response.usageMetadata);
// candidatesTokenCount and totalTokenCount depend on response, may vary
// { promptTokenCount: 11, candidatesTokenCount: 124, totalTokenCount: 135 }
```
om/google-gemini/generative-ai-js/blob/53bf9b6d996be72b5f52c8397de34c5abf1f1be7/samples/count_tokens.js#L33-L57)

Count multi-turn (chat) tokens

If you call `countTokens` with the chat history, it returns the total token count of the text from each role in the chat (`totalTokens`).

Another option is calling `sendMessage` and then using the `usageMetadata` attribute on the `response` object to get the following:

- The separate token counts of the input (`promptTokenCount`) and the output (`candidatesTokenCount`)
- The total number of tokens in *both the input and the output* (`totalTokenCount`)

To understand how big your next conversational turn will be, you need to append it to the history when you call `countTokens`.

```
// Make sure to include these imports:
// import { GoogleGenerativeAI } from "@google/generative-ai";
const genAI = new GoogleGenerativeAI(process.env.API_KEY);
const model = genAI.getGenerativeModel({
  model: "gemini-1.5-flash",
});

const chat = model.startChat({
  history: [
    {
      role: "user",
      parts: [{ text: "Hi my name is Bob" }],
    },
    {
      role: "model",
      parts: [{ text: "Hi Bob!" }],
    },
  ],
});

const countResult = await model.countTokens({
  generateContentRequest: { contents: await chat.getHistory() },
});
console.log(countResult.totalTokens); // 10

const chatResult = await chat.sendMessage(
  "In one sentence, explain how a computer works to a young child.",
);

// On the response for `sendMessage`, use `usageMetadata`
// to get separate input and output token counts
// (`promptTokenCount` and `candidatesTokenCount`, respectively),
// as well as the combined token count (`totalTokenCount`).
console.log(chatResult.response.usageMetadata);
// candidatesTokenCount and totalTokenCount depend on response, may vary
// { promptTokenCount: 25, candidatesTokenCount: 25, totalTokenCount: 50 }
```
om/google-gemini/generative-ai-js/blob/53bf9b6d996be72b5f52c8397de34c5abf1f1be7/samples/count_tokens.js#L63-L98)

Count multimodal tokens

All input to the Gemini API is tokenized, including text, image files, and other non-text modalities. Note the following high-level key points about tokenization of multimodal input during processing by the Gemini API:

- With Gemini 2.0, image inputs with both dimensions <=384 pixels are counted as 258 tokens. Images larger in one or both dimensions are cropped and scaled as needed into tiles of 768x768 pixels, each counted as 258 tokens. Prior to Gemini 2.0, images used a fixed 258 tokens.
- Video and audio files are converted to tokens at the following fixed rates: video at 263 tokens per second and audio at 32 tokens per second.

Image files

If you call `countTokens` with a text-and-image input, it returns the combined token count of the text and the image in *the input only* (`totalTokens`). You can make this call before calling `generateContent` to check the size of your requests. You can also optionally call `countTokens` on the text and the file separately.

Another option is calling `generateContent` and then using the `usageMetadata` attribute on the `response` object to get the following:

- The separate token counts of the input (`promptTokenCount`) and the output (`candidatesTokenCount`)
- The total number of tokens in *both the input and the output* (`totalTokenCount`)

Note: You'll get the same token count if you use a file uploaded using the File API or you provide the file as inline data.

Example that uses an uploaded image from the File API:

```
// Make sure to include these imports:
// import { GoogleAIFileManager } from "@google/generative-ai/server";
// import { GoogleGenerativeAI } from "@google/generative-ai";
const fileManager = new GoogleAIFileManager(process.env.API_KEY);

const uploadResult = await fileManager.uploadFile(
  `${mediaPath}/jetpack.jpg`,
  { mimeType: "image/jpeg" },
);

const imagePart = {
  fileData: {
    fileUri: uploadResult.file.uri,
    mimeType: uploadResult.file.mimeType,
  },
};

const genAI = new GoogleGenerativeAI(process.env.API_KEY);
const model = genAI.getGenerativeModel({
  model: "gemini-1.5-flash",
});

const prompt = "Tell me about this image.";

// Call `countTokens` to get the input token count
// of the combined text and file (`totalTokens`).
// An image's display or file size does not affect its token count.
// Optionally, you can call `countTokens` for the text and file separately.
const countResult = await model.countTokens([prompt, imagePart]);

console.log(countResult.totalTokens); // 265

const generateResult = await model.generateContent([prompt, imagePart]);

// On the response for `generateContent`, use `usageMetadata`
// to get separate input and output token counts
// (`promptTokenCount` and `candidatesTokenCount`, respectively),
// as well as the combined token count (`totalTokenCount`).
console.log(generateResult.response.usageMetadata);
// candidatesTokenCount and totalTokenCount depend on response, may vary
// { promptTokenCount: 265, candidatesTokenCount: 157, totalTokenCount: 422 }
```
ı/google-gemini/generative-ai-js/blob/53bf9b6d996be72b5f52c8397de34c5abf1f1be7/samples/count_tokens.js#L148-L188)

Example that provides the image as inline data:

```
// Make sure to include these imports:
// import { GoogleGenerativeAI } from "@google/generative-ai";
const genAI = new GoogleGenerativeAI(process.env.API_KEY);
const model = genAI.getGenerativeModel({
  model: "gemini-1.5-flash",
});

function fileToGenerativePart(path, mimeType) {
  return {
    inlineData: {
      data: Buffer.from(fs.readFileSync(path)).toString("base64"),
      mimeType,
    },
  };
}

const imagePart = fileToGenerativePart(
  `${mediaPath}/jetpack.jpg`,
  "image/jpeg",
);

const prompt = "Tell me about this image.";

// Call `countTokens` to get the input token count
// of the combined text and file (`totalTokens`).
// An image's display or file size does not affect its token count.
// Optionally, you can call `countTokens` for the text and file separately.
const countResult = await model.countTokens([prompt, imagePart]);
console.log(countResult.totalTokens); // 265

const generateResult = await model.generateContent([prompt, imagePart]);

// On the response for `generateContent`, use `usageMetadata`
// to get separate input and output token counts
// (`promptTokenCount` and `candidatesTokenCount`, respectively),
// as well as the combined token count (`totalTokenCount`).
console.log(generateResult.response.usageMetadata);
// candidatesTokenCount and totalTokenCount depend on response, may vary
// { promptTokenCount: 265, candidatesTokenCount: 157, totalTokenCount: 422 }
```

ı/google-gemini/generative-ai-js/blob/53bf9b6d996be72b5f52c8397de34c5abf1f1be7/samples/count_tokens.js#L104-L142)

Video or audio files

Audio and video are each converted to tokens at the following fixed rates:

- Video: 263 tokens per second
- Audio: 32 tokens per second

If you call `countTokens` with a text-and-video/audio input, it returns the combined token count of the text and the video/audio file in *the input only* (`totalTokens`). You can make this call before calling `generateContent` to check the size of your requests. You can also optionally call `countTokens` on the text and the file separately.

Another option is calling `generateContent` and then using the `usageMetadata` attribute on the `response` object to get the following:

- The separate token counts of the input (`promptTokenCount`) and the output (`candidatesTokenCount`)
- The total number of tokens in *both the input and the output* (`totalTokenCount`)

Note: You'll get the same token count if you use a file uploaded using the File API or you provide the file as inline data.

```
// Make sure to include these imports:
// import { GoogleAIFileManager, FileState } from "@google/generative-ai/server";
// import { GoogleGenerativeAI } from "@google/generative-ai";
const fileManager = new GoogleAIFileManager(process.env.API_KEY);

const uploadVideoResult = await fileManager.uploadFile(
  `${mediaPath}/Big_Buck_Bunny.mp4`,
  { mimeType: "video/mp4" },
);

let file = await fileManager.getFile(uploadVideoResult.file.name);
process.stdout.write("processing video");
while (file.state === FileState.PROCESSING) {
  process.stdout.write(".");
  // Sleep for 10 seconds
  await new Promise((resolve) => setTimeout(resolve, 10_000));
  // Fetch the file from the API again
  file = await fileManager.getFile(uploadVideoResult.file.name);
}

if (file.state === FileState.FAILED) {
  throw new Error("Video processing failed.");
} else {
  process.stdout.write("\n");
}

const videoPart = {
  fileData: {
    fileUri: uploadVideoResult.file.uri,
    mimeType: uploadVideoResult.file.mimeType,
  },
};

const genAI = new GoogleGenerativeAI(process.env.API_KEY);
const model = genAI.getGenerativeModel({
  model: "gemini-1.5-flash",
});

const prompt = "Tell me about this video.";

// Call `countTokens` to get the input token count
// of the combined text and file (`totalTokens`).
// A video or audio file is converted to tokens at a fixed rate of tokens
// per second.
// Optionally, you can call `countTokens` for the text and file separately.
const countResult = await model.countTokens([prompt, videoPart]);

console.log(countResult.totalTokens); // 302
```

```
const generateResult = await model.generateContent([prompt, videoPart]);

// On the response for `generateContent`, use `usageMetadata`
// to get separate input and output token counts
// (`promptTokenCount` and `candidatesTokenCount`, respectively),
// as well as the combined token count (`totalTokenCount`).
console.log(generateResult.response.usageMetadata);
// candidatesTokenCount and totalTokenCount depend on response, may vary
// { promptTokenCount: 302, candidatesTokenCount: 46, totalTokenCount: 348 }
```

ı/google-gemini/generative-ai-js/blob/53bf9b6d996be72b5f52c8397de34c5abf1f1be7/samples/count_tokens.js#L195-L252)

System instructions and tools

System instructions and tools also count towards the total token count for the input.

If you use system instructions, the `totalTokens` count increases to reflect the addition of `systemInstruction`.

```
// Make sure to include these imports:
// import { GoogleGenerativeAI } from "@google/generative-ai";
const genAI = new GoogleGenerativeAI(process.env.API_KEY);
const prompt = "The quick brown fox jumps over the lazy dog.";
const modelNoInstructions = genAI.getGenerativeModel({
  model: "models/gemini-1.5-flash",
});

const resultNoInstructions = await modelNoInstructions.countTokens(prompt);

console.log(resultNoInstructions);
// { totalTokens: 11 }

const modelWithInstructions = genAI.getGenerativeModel({
  model: "models/gemini-1.5-flash",
  systemInstruction: "You are a cat. Your name is Neko.",
});

const resultWithInstructions =
  await modelWithInstructions.countTokens(prompt);

// The total token count includes everything sent to the
// generateContent() request. When you use system instructions, the
// total token count increases.
console.log(resultWithInstructions);
// { totalTokens: 23 }
```

ı/google-gemini/generative-ai-js/blob/53bf9b6d996be72b5f52c8397de34c5abf1f1be7/samples/count_tokens.js#L325-L350)

If you use function calling, the `totalTokens` count increases to reflect the addition of `tools`.

```
// Make sure to include these imports:
// import { GoogleGenerativeAI } from "@google/generative-ai";
```

```
const genAI = new GoogleGenerativeAI(process.env.API_KEY);
const prompt =
  "I have 57 cats, each owns 44 mittens, how many mittens is that in total?";

const modelNoTools = genAI.getGenerativeModel({
  model: "models/gemini-1.5-flash",
});

const resultNoTools = await modelNoTools.countTokens(prompt);

console.log(resultNoTools);
// { totalTokens: 23 }

const functionDeclarations = [
  { name: "add" },
  { name: "subtract" },
  { name: "multiply" },
  { name: "divide" },
];

const modelWithTools = genAI.getGenerativeModel({
  model: "models/gemini-1.5-flash",
  tools: [{ functionDeclarations }],
});

const resultWithTools = await modelWithTools.countTokens(prompt);

// The total token count includes everything sent to the
// generateContent() request. When you use tools (like function calling),
// the total token count increases.
console.log(resultWithTools);
// { totalTokens: 99 }
```
ı/google-gemini/generative-ai-js/blob/53bf9b6d996be72b5f52c8397de34c5abf1f1be7/samples/count_tokens.js#L356-L389)