Learn more (https://developers.googleblog.com/en/gemini-2-5-flash-pro-live-api-veo-2-gemini-api/)

Document understanding

Python ✓ JavaScript Go REST

The Gemini API supports PDF input, including long documents (up to 1000 pages). Gemini models process PDFs with native vision, and are therefore able to understand both text and image contents inside documents. With native PDF vision support, Gemini models are able to:

- Analyze diagrams, charts, and tables inside documents
- Extract information into structured output formats
- Answer questions about visual and text contents in documents
- Summarize documents
- Transcribe document content (e.g. to HTML) preserving layouts and formatting, for use in downstream applications

This tutorial demonstrates some possible ways to use the Gemini API to process PDF documents.

PDF input

For PDF payloads under 20MB, you can choose between uploading base64 encoded documents or directly uploading locally stored files.

As inline data

You can process PDF documents directly from URLs. Here's a code snippet showing how to do this:

```
import { GoogleGenAI } from "@google/genai";

const ai = new GoogleGenAI({ apiKey: "GEMINI_API_KEY" });

async function main() {
    const pdfResp = await fetch('https://discovery.ucl.ac.uk/id/eprint/10089234/1/34
        .then((response) => response.arrayBuffer());

    const contents = [
        { text: "Summarize this document" },
        {
            inlineData: {
                mimeType: 'application/pdf',
                data: Buffer.from(pdfResp).toString("base64")
            }
        }
    ];

    const response = await ai.models.generateContent({
        model: "gemini-2.0-flash",
        contents: contents
    });
    console.log(response.text);
}

main();
```

Technical details

Gemini 1.5 Pro and 1.5 Flash support a maximum of 3,600 document pages. Document pages must be in one of the following text data MIME types:

- PDF - **`application/pdf`**
- JavaScript - **`application/x-javascript`**, **`text/javascript`**
- Python - **`application/x-python`**, **`text/x-python`**
- TXT - **`text/plain`**
- HTML - **`text/html`**
- CSS - **`text/css`**
- Markdown - **`text/md`**
- CSV - **`text/csv`**
- XML - **`text/xml`**
- RTF - **`text/rtf`**

Each document page is equivalent to 258 tokens.

While there are no specific limits to the number of pixels in a document besides the model's context window, larger pages are scaled down to a maximum resolution of 3072x3072 while preserving their original aspect ratio, while smaller pages are scaled up to 768x768 pixels. There is no cost reduction for pages at lower sizes, other than bandwidth, or performance improvement for pages at higher resolution.

For best results:

- Rotate pages to the correct orientation before uploading.
- Avoid blurry pages.
- If using a single page, place the text prompt after the page.

Locally stored PDFs

For locally stored PDFs, you can use the following approach:

```
import { GoogleGenAI } from "@google/genai";
import * as fs from 'fs';

const ai = new GoogleGenAI({ apiKey: "GEMINI_API_KEY" });

async function main() {
    const contents = [
        { text: "Summarize this document" },
        {
            inlineData: {
                mimeType: 'application/pdf',
                data: Buffer.from(fs.readFileSync("content/343019_3_art_0_py4t4l_con
            }
        }
    ];

    const response = await ai.models.generateContent({
        model: "gemini-2.0-flash",
        contents: contents
    });
    console.log(response.text);
}

main();
```

Large PDFs

You can use the File API to upload larger documents. Always use the File API when the total request size (including the files, text prompt, system instructions, etc.) is larger than 20 MB.

Note: The File API lets you store up to 50 MB of PDF files. Files are stored for 48 hours. They can be accessed in that period with your API key, but cannot be downloaded from the API. The File API is available at no cost in all regions where the Gemini API is available.

Call **`media.upload`** (/api/rest/v1beta/media/upload) to upload a file using the File API. The following code uploads a document file and then uses the file in a call to **`models.generateContent`** (/api/generate-content#method:-models.generatecontent).

Large PDFs from URLs

Use the File API for large PDF files available from URLs, simplifying the process of uploading and processing these documents directly through their URLs:

```
import { createPartFromUri, GoogleGenAI } from "@google/genai";

const ai = new GoogleGenAI({ apiKey: "GEMINI_API_KEY" });

async function main() {

    const pdfBuffer = await fetch("https://www.nasa.gov/wp-content/uploads/static/hi
        .then((response) => response.arrayBuffer());

    const fileBlob = new Blob([pdfBuffer], { type: 'application/pdf' });

    const file = await ai.files.upload({
        file: fileBlob,
        config: {
            displayName: 'A17_FlightPlan.pdf',
        },
    });

    // Wait for the file to be processed.
    let getFile = await ai.files.get({ name: file.name });
    while (getFile.state === 'PROCESSING') {
        getFile = await ai.files.get({ name: file.name });
        console.log(`current file status: ${getFile.state}`);
        console.log('File is still processing, retrying in 5 seconds');
        await new Promise((resolve) => {
            setTimeout(resolve, 5000);
        });
    }
    if (file.state === 'FAILED') {
        throw new Error('File processing failed.');
    }

    // Add the file to the contents.
    const content = [
        'Summarize this document',
    ];

    if (file.uri && file.mimeType) {
        const fileContent = createPartFromUri(file.uri, file.mimeType);
        content.push(fileContent);
    }

    const response = await ai.models.generateContent({
        model: 'gemini-2.0-flash',
        contents: content,
    });

    console.log(response.text);

}

main();
```

Large PDFs stored locally

```
import { createPartFromUri, GoogleGenAI } from "@google/genai";

const ai = new GoogleGenAI({ apiKey: "GEMINI_API_KEY" });

async function main() {
    const file = await ai.files.upload({
        file: 'path-to-localfile.pdf'
        config: {
            displayName: 'A17_FlightPlan.pdf',
        },
    });
    // Wait for the file to be processed.
    let getFile = await ai.files.get({ name: file.name });
    while (getFile.state === 'PROCESSING') {
        getFile = await ai.files.get({ name: file.name });
        console.log(`current file status: ${getFile.state}`);
        console.log('File is still processing, retrying in 5 seconds');

        await new Promise((resolve) => {
            setTimeout(resolve, 5000);
        });
    }
    if (file.state === 'FAILED') {
        throw new Error('File processing failed.');
    }

    // Add the file to the contents.
    const content = [
        'Summarize this document',
    ];

    if (file.uri && file.mimeType) {
        const fileContent = createPartFromUri(file.uri, file.mimeType);
        content.push(fileContent);
    }

    const response = await ai.models.generateContent({
        model: 'gemini-2.0-flash',
        contents: content,
    });

    console.log(response.text);

}

main();
```

You can verify the API successfully stored the uploaded file and get its metadata by calling `files.get` (/api/rest/v1beta/files/get). Only the `name` (and by extension, the `uri`) are unique.

Multiple PDFs

The Gemini API is capable of processing multiple PDF documents in a single request, as long as the combined size of the documents and the text prompt stays within the model's context window.

```
import { createPartFromUri, GoogleGenAI } from "@google/genai";

const ai = new GoogleGenAI({ apiKey: "GEMINI_API_KEY" });

async function uploadRemotePDF(url, displayName) {
    const pdfBuffer = await fetch(url)
        .then((response) => response.arrayBuffer());

    const fileBlob = new Blob([pdfBuffer], { type: 'application/pdf' });

    const file = await ai.files.upload({
        file: fileBlob,
        config: {
            displayName: displayName,
        },
    });

    // Wait for the file to be processed.
    let getFile = await ai.files.get({ name: file.name });
    while (getFile.state === 'PROCESSING') {
        getFile = await ai.files.get({ name: file.name });
        console.log(`current file status: ${getFile.state}`);
        console.log('File is still processing, retrying in 5 seconds');

        await new Promise((resolve) => {
            setTimeout(resolve, 5000);
        });
    }
    if (file.state === 'FAILED') {
        throw new Error('File processing failed.');
    }

    return file;
}

async function main() {
    const content = [
        'What is the difference between each of the main benchmarks between these tw
    ];

    let file1 = await uploadRemotePDF("https://arxiv.org/pdf/2312.11805", "PDF 1")
    if (file1.uri && file1.mimeType) {
        const fileContent = createPartFromUri(file1.uri, file1.mimeType);
        content.push(fileContent);
    }
    let file2 = await uploadRemotePDF("https://arxiv.org/pdf/2403.05530", "PDF 2")
    if (file2.uri && file2.mimeType) {
        const fileContent = createPartFromUri(file2.uri, file2.mimeType);
        content.push(fileContent);
    }

    const response = await ai.models.generateContent({
        model: 'gemini-2.0-flash',
        contents: content,
    });

    console.log(response.text);
}

main();
```

What's next

To learn more, see the following resources:

- File prompting strategies (/gemini-api/docs/files#prompt-guide): The Gemini API supports prompting with text, image, audio, and video data, also known as multimodal prompting.
- System instructions (/gemini-api/docs/text-generation#system-instructions): System instructions let you steer the behavior of the model based on your specific needs and use cases.