(https://developers.googleblog.com/en/introducing-veo-3-1-and-new-creative-controls-in-the-gemini-api/)
and **documentation** (https://ai.google.dev/gemini-api/docs/video).

Document understanding

Gemini models can process documents in PDF format, using native vision to understand entire document contexts. This goes beyond simple text extraction, allowing Gemini to:

- Analyze and interpret content, including text, images, diagrams, charts, and tables, even in long documents up to 1000 pages.
- Extract information into structured output (/gemini-api/docs/structured-output) formats.
- Summarize and answer questions based on both the visual and textual elements in a document.
- Transcribe document content (e.g. to HTML), preserving layouts and formatting, for use in downstream applications.

Passing inline PDF data

You can pass inline PDF data in the request to `generateContent`. For PDF payloads under 20MB, you can choose between uploading base64 encoded documents or directly uploading locally stored files.

The following example shows you how to fetch a PDF from a URL and convert it to bytes for processing:

Python (#python) JavaScript (#javascript) Go (#go) REST (#rest)

```
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/epr
        .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-2.5-flash",
        contents: contents
    });
    console.log(response.text);
}

main();
```

You can also read a PDF from a local file for processing:

Python (#python) JavaScript (#javascript) Go (#go)

```
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
            }
        }
```

```
    ];

    const response = await ai.models.generateContent({
        model: "gemini-2.5-flash",
        contents: contents
    });
    console.log(response.text);
}

main();
```

Uploading PDFs using the File API

You can use the File API (/gemini-api/docs/files) 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 20MB.

Note: The File API (/gemini-api/docs/files) lets you store up to 50MB of PDF files. Files are stored for 48 hours. You can access them in that period with your API key, but you can't download them 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 to simplify uploading and processing large PDF files from URLs:

Python (#python) JavaScript (#javascript) Go (#go) REST (#rest)

```
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/
        .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 second

        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.mimeTy
        content.push(fileContent);
    }

    const response = await ai.models.generateContent({
        model: 'gemini-2.5-flash',
        contents: content,
    });

    console.log(response.text);

}
```

```
main();
```

Large PDFs stored locally

Python (#python) JavaScript (#javascript) Go (#go) REST (#rest)

```
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 second

        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.mimeTy
```

```
        content.push(fileContent);
    }

    const response = await ai.models.generateContent({
        model: 'gemini-2.5-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.

PythonREST (#rest)
(#python)

```
from google import genai
import pathlib

client = genai.Client()

fpath = pathlib.Path('example.txt')
fpath.write_text('hello')

file = client.files.upload(file='example.txt')

file_info = client.files.get(name=file.name)
print(file_info.model_dump_json(indent=4))
```

Passing multiple PDFs

The Gemini API is capable of processing multiple PDF documents (up to 1000 pages) in a single request, as long as the combined size of the documents and the text prompt stays within the model's context window.

Python (#python) JavaScript (#javascript) Go (#go) REST (#rest)

```
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 second

        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
    ];

    let file1 = await uploadRemotePDF("https://arxiv.org/pdf/2312.1
    if (file1.uri && file1.mimeType) {
```

```
        const fileContent = createPartFromUri(file1.uri, file1.mime
        content.push(fileContent);
    }
    let file2 = await uploadRemotePDF("https://arxiv.org/pdf/2403.0
    if (file2.uri && file2.mimeType) {
        const fileContent = createPartFromUri(file2.uri, file2.mime
        content.push(fileContent);
    }

    const response = await ai.models.generateContent({
        model: 'gemini-2.5-flash',
        contents: content,
    });

    console.log(response.text);
}

main();
```

Technical details

Gemini supports a maximum of 1,000 document pages. 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 (/gemini-api/docs/long-context), 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.

Document types

Technically, you can pass other MIME types for document understanding, like TXT, Markdown, HTML, XML, etc. However, document vision ***only meaningfully understands PDFs***. Other types will be extracted as pure text, and the model won't be able to interpret what we see in the rendering of those files. Any file-type specifics like charts, diagrams, HTML tags, Markdown formatting, etc., will be lost.

Best practices

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.

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.

Last updated 2025-09-22 UTC.