Batch API

The Gemini Batch API is designed to process large volumes of requests asynchronously at 50% of the standard cost (/gemini-api/docs/pricing). The target turnaround time is 24 hours, but in majority of cases, it is much quicker.

Use Batch API for large-scale, non-urgent tasks such as data pre-processing or running evaluations where an immediate response is not required.

Creating a batch job

You have two ways to submit your requests in Batch API:

- **Inline Requests** (#inline-requests)**:** A list of `GenerateContentRequest` (/api/batch-mode#GenerateContentRequest) objects directly included in your batch creation request. This is suitable for smaller batches that keep the total request size under 20MB. The **output** returned from the model is a list of `inlineResponse` objects.
- **Input File** (#input-file)**:** A JSON Lines (JSONL) (https://jsonlines.org/) file where each line contains a complete `GenerateContentRequest` (/api/batch-mode#GenerateContentRequest) object. This method is recommended for larger requests. The **output** returned from the model is a JSONL file where each line is either a `GenerateContentResponse` or a status object.

Inline requests

For a small number of requests, you can directly embed the `GenerateContentRequest` (/api/batch-mode#GenerateContentRequest) objects within your `BatchGenerateContentRequest` (/api/batch-mode#request-body). The following example calls the `BatchGenerateContent` (/api/batch-mode#google.ai.generativelanguage.v1beta.BatchService.BatchGenerateContent) method with inline requests:

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

```
import {GoogleGenAI} from '@google/genai';
const GEMINI_API_KEY = process.env.GEMINI_API_KEY;

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

const inlinedRequests = [
    {
        contents: [{
            parts: [{text: 'Tell me a one-sentence joke.'}],
            role: 'user'
        }]
    },
    {
        contents: [{
            parts: [{'text': 'Why is the sky blue?'}],
            role: 'user'
        }]
    }
]

const response = await ai.batches.create({
    model: 'gemini-2.5-flash',
    src: inlinedRequests,
    config: {
        displayName: 'inlined-requests-job-1',
    }
});

console.log(response);
```

Input file

For larger sets of requests, prepare a JSON Lines (JSONL) file. Each line in this file must be a JSON object containing a user-defined key and a request object, where the request is a valid **`GenerateContentRequest`** (/api/batch-mode#GenerateContentRequest) object. The user-defined key is used in the response to indicate which output is the result of which request. For example, the request with the key defined as **`request-1`** will have its response annotated with the same key name.

This file is uploaded using the File API (/gemini-api/docs/files). The maximum allowed file size for an input file is 2GB.

The following is an example of a JSONL file. You can save it in a file named **`my-batch-requests.json`**:

```
{"key": "request-1", "request": {"contents": [{"parts": [{"text": "Describe the proces
{"key": "request-2", "request": {"contents": [{"parts": [{"text": "What are the main i
```

Similarly to inline requests, you can specify other parameters like system instructions, tools or other configurations in each request JSON.

You can upload this file using the File API (/gemini-api/docs/files) as shown in the following example. If you are working with multimodal input, you can reference other uploaded files within your JSONL file.

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

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

const GEMINI_API_KEY = process.env.GEMINI_API_KEY;
const ai = new GoogleGenAI({apiKey: GEMINI_API_KEY});
const fileName = "my-batch-requests.jsonl";

// Define the requests
const requests = [
    { "key": "request-1", "request": { "contents": [{ "parts": [{ "text": "Descr
    { "key": "request-2", "request": { "contents": [{ "parts": [{ "text": "What
];

// Construct the full path to file
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);
const filePath = path.join(__dirname, fileName); // __dirname is the directory o
```

```
async function writeBatchRequestsToFile(requests, filePath) {
    try {
        // Use a writable stream for efficiency, especially with larger files.
        const writeStream = fs.createWriteStream(filePath, { flags: 'w' });

        writeStream.on('error', (err) => {
            console.error(`Error writing to file ${filePath}:`, err);
        });

        for (const req of requests) {
            writeStream.write(JSON.stringify(req) + '\n');
        }

        writeStream.end();

        console.log(`Successfully wrote batch requests to ${filePath}`);

    } catch (error) {
        // This catch block is for errors that might occur before stream setup,
        // stream errors are handled by the 'error' event.
        console.error(`An unexpected error occurred:`, error);
    }
}

// Write to a file.
writeBatchRequestsToFile(requests, filePath);

// Upload the file to the File API.
const uploadedFile = await ai.files.upload({file: 'my-batch-requests.jsonl', con
    mimeType: 'jsonl',
}});
console.log(uploadedFile.name);
```

The following example calls the `BatchGenerateContent` (/api/batch-mode#google.ai.generativelanguage.v1beta.BatchService.BatchGenerateContent) method with the input file uploaded using File API:

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

```
// Assumes `uploadedFile` is the file object from the previous step
const fileBatchJob = await ai.batches.create({
    model: 'gemini-2.5-flash',
    src: uploadedFile.name,
    config: {
        displayName: 'file-upload-job-1',
    }
});

console.log(fileBatchJob);
```

When you create a batch job, you will get a job name returned. Use this name for monitoring (#batch-job-status) the job status as well as retrieving the results (#retrieve-batch-results) once the job completes.

The following is an example output that contains a job name:

```
Created batch job from file: batches/123456789
```

Batch embedding support

You can use the Batch API to interact with the Embeddings model (/gemini-api/docs/embeddings) for higher throughput. To create an embeddings batch job with either inline requests (#inline-requests) or input files (#input-file), use the **`batches.create_embeddings`** API and specify the embeddings model.

Python (#python) JavaScript (#javascript)

```
// Creating an embeddings batch job with an input file request:
let fileJob;
fileJob = await client.batches.createEmbeddings({
    model: 'gemini-embedding-001',
    src: {fileName: uploadedBatchRequests.name},
```

```
    config: {displayName: 'Input embeddings batch'},
});
console.log(`Created batch job: ${fileJob.name}`);

// Creating an embeddings batch job with an inline request:
let batchJob;
batchJob = await client.batches.createEmbeddings({
    model: 'gemini-embedding-001',
    // For a predefined a list of requests `inlinedRequests`
    src: {inlinedRequests: inlinedRequests},
    config: {displayName: 'Inlined embeddings batch'},
});
console.log(`Created batch job: ${batchJob.name}`);
```

Read the Embeddings section in the Batch API cookbook (https://github.com/google-gemini/cookbook/blob/main/quickstarts/Batch_mode.ipynb) for more examples.

Request configuration

You can include any request configurations you would use in a standard non-batch request. For example, you could specify the temperature, system instructions or even pass in other modalities. The following example shows an example inline request that contains a system instruction for one of the requests:

Python (#python) JavaScript (#javascript)

```
inlineRequestsList = [
    {contents: [{parts: [{text: 'Write a short poem about a cloud.'}]}]},
    {contents: [{parts: [{text: 'Write a short poem about a cat.'}]}],
     config: {systemInstruction: {parts: [{text: 'You are a cat. Your name is Ne
]
```

Similarly can specify tools to use for a request. The following example shows a request that enables the Google Search tool (/gemini-api/docs/google-search):

Python (#python) JavaScript (#javascript)

```
inlineRequestsList = [
    {contents: [{parts: [{text: 'Who won the euro 1998?'}]}]},
    {contents: [{parts: [{text: 'Who won the euro 2025?'}]}],
     config: {tools: [{googleSearch: {}}]}}
]
```

You can specify structured output (/gemini-api/docs/structured-output) as well. The following example shows how to specify for your batch requests.

Python (#python) JavaScript (#javascript)

```
import {GoogleGenAI, Type} from '@google/genai';
const GEMINI_API_KEY = process.env.GEMINI_API_KEY;

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

const inlinedRequests = [
    {
        contents: [{
            parts: [{text: 'List a few popular cookie recipes, and include the a
            role: 'user'
        }],
        config: {
            responseMimeType: 'application/json',
            responseSchema: {
            type: Type.ARRAY,
            items: {
                type: Type.OBJECT,
                properties: {
                'recipeName': {
                    type: Type.STRING,
                    description: 'Name of the recipe',
                    nullable: false,
                },
```

```
                'ingredients': {
                    type: Type.ARRAY,
                    items: {
                    type: Type.STRING,
                    description: 'Ingredients of the recipe',
                    nullable: false,
                    },
                },
                },
                required: ['recipeName'],
            },
            },
        }
    },
    {
        contents: [{
            parts: [{text: 'List a few popular gluten free cookie recipes, and i
            role: 'user'
        }],
        config: {
            responseMimeType: 'application/json',
            responseSchema: {
            type: Type.ARRAY,
            items: {
                type: Type.OBJECT,
                properties: {
                'recipeName': {
                    type: Type.STRING,
                    description: 'Name of the recipe',
                    nullable: false,
                },
                'ingredients': {
                    type: Type.ARRAY,
                    items: {
                    type: Type.STRING,
                    description: 'Ingredients of the recipe',
                    nullable: false,
                    },
                },
                },
                required: ['recipeName'],
            },
```

```
            },
        }
    }
]

const inlinedBatchJob = await ai.batches.create({
    model: 'gemini-2.5-flash',
    src: inlinedRequests,
    config: {
        displayName: 'inlined-requests-job-1',
    }
});
```

Monitoring job status

Use the operation name obtained when creating the batch job to poll its status. The state field of the batch job will indicate its current status. A batch job can be in one of the following states:

- **JOB_STATE_PENDING**: The job has been created and is waiting to be processed by the service.
- **JOB_STATE_RUNNING**: The job is in progress.
- **JOB_STATE_SUCCEEDED**: The job completed successfully. You can now retrieve the results.
- **JOB_STATE_FAILED**: The job failed. Check the error details for more information.
- **JOB_STATE_CANCELLED**: The job was cancelled by the user.
- **JOB_STATE_EXPIRED**: The job has expired because it was running or pending for more than 48 hours. The job will not have any results to retrieve. You can try submitting the job again or splitting up the requests into smaller batches.

You can poll the job status periodically to check for completion.

Python (#python) JavaScript (#javascript)

```
// Use the name of the job you want to check
```

```
// e.g., inlinedBatchJob.name from the previous step
let batchJob;
const completedStates = new Set([
    'JOB_STATE_SUCCEEDED',
    'JOB_STATE_FAILED',
    'JOB_STATE_CANCELLED',
    'JOB_STATE_EXPIRED',
]);

try {
    batchJob = await ai.batches.get({name: inlinedBatchJob.name});
    while (!completedStates.has(batchJob.state)) {
        console.log(`Current state: ${batchJob.state}`);
        // Wait for 30 seconds before polling again
        await new Promise(resolve => setTimeout(resolve, 30000));
        batchJob = await client.batches.get({ name: batchJob.name });
    }
    console.log(`Job finished with state: ${batchJob.state}`);
    if (batchJob.state === 'JOB_STATE_FAILED') {
        // The exact structure of `error` might vary depending on the SDK
        // This assumes `error` is an object with a `message` property.
        console.error(`Error: ${batchJob.state}`);
    }
} catch (error) {
    console.error(`An error occurred while polling job ${batchJob.name}:`, error
}
```

Retrieving results

Once the job status indicates your batch job has succeeded, the results are available in the **response** field.

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

```
// Use the name of the job you want to check
// e.g., inlinedBatchJob.name from the previous step
```

```
const jobName = "YOUR_BATCH_JOB_NAME";

try {
    const batchJob = await ai.batches.get({ name: jobName });

    if (batchJob.state === 'JOB_STATE_SUCCEEDED') {
        console.log('Found completed batch:', batchJob.displayName);
        console.log(batchJob);

        // If batch job was created with a file destination
        if (batchJob.dest?.fileName) {
            const resultFileName = batchJob.dest.fileName;
            console.log(`Results are in file: ${resultFileName}`);

            console.log("Downloading result file content...");
            const fileContentBuffer = await ai.files.download({ file: resultFile

            // Process fileContentBuffer (Buffer) as needed
            console.log(fileContentBuffer.toString('utf-8'));
        }

        // If batch job was created with inline responses
        else if (batchJob.dest?.inlinedResponses) {
            console.log("Results are inline:");
            for (let i = 0; i < batchJob.dest.inlinedResponses.length; i++) {
                const inlineResponse = batchJob.dest.inlinedResponses[i];
                console.log(`Response ${i + 1}:`);
                if (inlineResponse.response) {
                    // Accessing response, structure may vary.
                    if (inlineResponse.response.text !== undefined) {
                        console.log(inlineResponse.response.text);
                    } else {
                        console.log(inlineResponse.response); // Fallback
                    }
                } else if (inlineResponse.error) {
                    console.error(`Error: ${inlineResponse.error}`);
                }
            }
        }

        // If batch job was an embedding batch with inline responses
        else if (batchJob.dest?.inlinedEmbedContentResponses) {
```

```
            console.log("Embedding results found inline:");
            for (let i = 0; i < batchJob.dest.inlinedEmbedContentResponses.lengt
                const inlineResponse = batchJob.dest.inlinedEmbedContentResponse
                console.log(`Response ${i + 1}:`);
                if (inlineResponse.response) {
                    console.log(inlineResponse.response);
                } else if (inlineResponse.error) {
                    console.error(`Error: ${inlineResponse.error}`);
                }
            }
        } else {
            console.log("No results found (neither file nor inline).");
        }
    } else {
        console.log(`Job did not succeed. Final state: ${batchJob.state}`);
        if (batchJob.error) {
            console.error(`Error: ${typeof batchJob.error === 'string' ? batchJo
        }
    }
} catch (error) {
    console.error(`An error occurred while processing job ${jobName}:`, error);
}
```

Cancelling a batch job

You can cancel an ongoing batch job using its name. When a job is canceled, it stops processing new requests.

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

```
await ai.batches.cancel({name: batchJobToCancel.name});
```

Deleting a batch job

You can delete an existing batch job using its name. When a job is deleted, it stops processing new requests and is removed from the list of batch jobs.

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

```
await ai.batches.delete({name: batchJobToDelete.name});
```

Technical details

- **Supported models:** Batch API supports a range of Gemini models. Refer to the Models page (/gemini-api/docs/models) for each model's support of Batch API. The supported modalities for Batch API are the same as what's supported on the interactive (or non-batch) API.
- **Pricing:** Batch API usage is priced at 50% of the standard interactive API cost for the equivalent model. See the pricing page (/gemini-api/docs/pricing) for details. Refer to the rate limits page (/gemini-api/docs/rate-limits#batch-mode) for details on rate limits for this feature.
- **Service Level Objective (SLO):** Batch jobs are designed to complete within a 24-hour turnaround time. Many jobs may complete much faster depending on their size and current system load.
- **Caching:** Context caching (/gemini-api/docs/caching) is enabled for batch requests. If a request in your batch results in a cache hit, the cached tokens are priced the same as for non-batch API traffic.

Best practices

- **Use input files for large requests:** For a large number of requests, always use the file input method for better manageability and to avoid hitting request size limits for the `BatchGenerateContent` (/api/batch-mode#google.ai.generativelanguage.v1beta.BatchService.BatchGenerateContent) call itself. Note that there's a the 2GB file size limit per input file.
- **Error handling:** Check the `batchStats` for `failedRequestCount` after a job completes. If using

file output, parse each line to check if it's a `GenerateContentResponse` or a status object indicating an error for that specific request. See the troubleshooting guide (/gemini-api/docs/troubleshooting#error-codes) for a complete set of error codes.

- **Submit jobs once:** The creation of a batch job is not idempotent. If you send the same creation request twice, two separate batch jobs will be created.
- **Break up very large batches:** While the target turnaround time is 24 hours, actual processing time can vary based on system load and job size. For large jobs, consider breaking them into smaller batches if intermediate results are needed sooner.

What's next

- Check out the Batch API notebook (https://colab.research.google.com/github/google-gemini/cookbook/blob/main/quickstarts/Batch_mode.ipynb) for more examples.
- The OpenAI compatibility layer supports Batch API. Read the examples on the OpenAI Compatibility (/gemini-api/docs/openai#batch) page.

Last updated 2025-10-09 UTC.