Batch API

Copy page

Process jobs asynchronously with Batch API.

Learn how to use OpenAI's Batch API to send asynchronous groups of requests with 50% lower costs, a separate pool of significantly higher rate limits, and a clear 24-hour turnaround time. The service is ideal for processing jobs that don't require immediate responses. You can also explore the API reference directly here.

Overview

While some uses of the OpenAI Platform require you to send synchronous requests, there are many cases where requests do not need an immediate response or rate limits prevent you from executing a large number of queries quickly. Batch processing jobs are often helpful in use cases like:

1. running evaluations
2. classifying large datasets
3. embedding content repositories

The Batch API offers a straightforward set of endpoints that allow you to collect a set of requests into a single file, kick off a batch processing job to execute these requests, query for the status of that batch while the underlying requests execute, and eventually retrieve the collected results when the batch is complete.

Compared to using standard endpoints directly, Batch API has:

1. **Better cost efficiency:** 50% cost discount compared to synchronous APIs
2. **Higher rate limits:** Substantially more headroom compared to the synchronous APIs
3. **Fast completion times:** Each batch completes within 24 hours (and often more quickly)

Getting Started

1. Preparing Your Batch File

Batches start with a `.jsonl` file where each line contains the details of an individual request to the API. For now, the available endpoints are `/v1/chat/completions` (Chat Completions API) and `/v1/embeddings` (Embeddings API). For a given input file, the parameters in each line's `body` field are the same as the parameters for the underlying endpoint. Each request must include a unique `custom_id` value, which you can use to reference results after completion.

Here's an example of an input file with 2 requests. Note that each input file can only include requests to a single model.

```
{"custom_id": "request-1", "method": "POST", "url": "/v1/chat/completions", "body": {"model": "
{"custom_id": "request-2", "method": "POST", "url": "/v1/chat/completions", "body": {"model": "
```

2. Uploading Your Batch Input File

Similar to our Fine-tuning API, you must first upload your input file so that you can reference it correctly when kicking off batches. Upload your `.jsonl` file using the Files API.

Upload files for Batch API javascript

```javascript
import fs from "fs";
import OpenAI from "openai";
const openai = new OpenAI();

const file = await openai.files.create({
  file: fs.createReadStream("batchinput.jsonl"),
  purpose: "batch",
});

console.log(file);
```

3. Creating the Batch

Once you've successfully uploaded your input file, you can use the input File object's ID to create a batch. In this case, let's assume the file ID is `file-abc123`. For now, the completion window can only be set to `24h`. You can also provide custom metadata via an optional `metadata` parameter.

Create the Batch javascript

```javascript
import OpenAI from "openai";
const openai = new OpenAI();

const batch = await openai.batches.create({
  input_file_id: "file-abc123",
  endpoint: "/v1/chat/completions",
  completion_window: "24h"
});

console.log(batch);
```

This request will return a Batch object with metadata about your batch:

```json
{
  "id": "batch_abc123",
  "object": "batch",
  "endpoint": "/v1/chat/completions",
  "errors": null,
  "input_file_id": "file-abc123",
  "completion_window": "24h",
  "status": "validating",
  "output_file_id": null,
  "error_file_id": null,
  "created_at": 1714508499,
  "in_progress_at": null,
  "expires_at": 1714536634,
  "completed_at": null,
  "failed_at": null,
  "expired_at": null,
  "request_counts": {
    "total": 0,
    "completed": 0,
    "failed": 0
  },
  "metadata": null
}
```

4. Checking the Status of a Batch

You can check the status of a batch at any time, which will also return a Batch object.

Check the status of a batch   javascript

```javascript
import OpenAI from "openai";
const openai = new OpenAI();

const batch = await openai.batches.retrieve("batch_abc123");
console.log(batch);
```

The status of a given Batch object can be any of the following:

STATUS	DESCRIPTION
`validating`	the input file is being validated before the batch can begin
`failed`	the input file has failed the validation process
`in_progress`	the input file was successfully validated and the batch is currently being run
`finalizing`	the batch has completed and the results are being prepared
`completed`	the batch has been completed and the results are ready
`expired`	the batch was not able to be completed within the 24-hour time window
`cancelling`	the batch is being cancelled (may take up to 10 minutes)

STATUS	DESCRIPTION
`cancelled`	the batch was cancelled

5. Retrieving the Results

Once the batch is complete, you can download the output by making a request against the Files API via the `output_file_id` field from the Batch object and writing it to a file on your machine, in this case `batch_output.jsonl`

Retrieving the batch results javascript

```javascript
import OpenAI from "openai";
const openai = new OpenAI();

const fileResponse = await openai.files.content("file-xyz123");
const fileContents = await fileResponse.text();

console.log(fileContents);
```

The output `.jsonl` file will have one response line for every successful request line in the input file. Any failed requests in the batch will have their error information written to an error file that can be found via the batch's `error_file_id`.

Note that the output line order **may not match** the input line order. Instead of relying on order to process your results, use the custom_id field which will be present in each line of your output file and allow you to map requests in your input to results in your output.

```
{"id": "batch_req_123", "custom_id": "request-2", "response": {"status_code": 200, "request_id"
{"id": "batch_req_456", "custom_id": "request-1", "response": {"status_code": 200, "request_id"
```

6. Cancelling a Batch

If necessary, you can cancel an ongoing batch. The batch's status will change to `cancelling` until in-flight requests are complete (up to 10 minutes), after which the status will change to `cancelled`.

Cancelling a batch javascript

```javascript
import OpenAI from "openai";
const openai = new OpenAI();

const batch = await openai.batches.cancel("batch_abc123");
console.log(batch);
```

7. Getting a List of All Batches

At any time, you can see all your batches. For users with many batches, you can use the `limit` and `after` parameters to paginate your results.

Getting a list of all batches

javascript

```javascript
import OpenAI from "openai";
const openai = new OpenAI();

const list = await openai.batches.list();

for await (const batch of list) {
  console.log(batch);
}
```

Model Availability

The Batch API can currently be used to execute queries against the following models. The Batch API supports text and vision inputs in the same format as the endpoints for these models:

- `o1`
- `o3-mini`
- `gpt-4o`
- `gpt-4o-mini`
- `gpt-4-turbo`
- `gpt-4`
- `gpt-4-32k`
- `gpt-3.5-turbo`
- `gpt-3.5-turbo-16k`
- `gpt-4-turbo-preview`
- `gpt-4-vision-preview`
- `gpt-4-turbo-2024-04-09`
- `gpt-4-0314`
- `gpt-4-32k-0314`
- `gpt-4-32k-0613`
- `gpt-3.5-turbo-0301`
- `gpt-3.5-turbo-16k-0613`
- `gpt-3.5-turbo-1106`
- `gpt-3.5-turbo-0613`
- `text-embedding-3-large`

`text-embedding-3-small`

`text-embedding-ada-002`

The Batch API also supports fine-tuned models.

Rate Limits

Batch API rate limits are separate from existing per-model rate limits. The Batch API has two new types of rate limits:

1. **Per-batch limits:** A single batch may include up to 50,000 requests, and a batch input file can be up to 200 MB in size. Note that `/v1/embeddings` batches are also restricted to a maximum of 50,000 embedding inputs across all requests in the batch.
2. **Enqueued prompt tokens per model:** Each model has a maximum number of enqueued prompt tokens allowed for batch processing. You can find these limits on the Platform Settings page.

There are no limits for output tokens or number of submitted requests for the Batch API today. Because Batch API rate limits are a new, separate pool, **using the Batch API will not consume tokens from your standard per-model rate limits**, thereby offering you a convenient way to increase the number of requests and processed tokens you can use when querying our API.

Batch Expiration

Batches that do not complete in time eventually move to an `expired` state; unfinished requests within that batch are cancelled, and any responses to completed requests are made available via the batch's output file. You will be charged for tokens consumed from any completed requests.

Expired requests will be written to your error file with the message as shown below. You can use the `custom_id` to retrieve the request data for expired requests.

```
{"id": "batch_req_123", "custom_id": "request-3", "response": null, "error": {"code": "batch_ex
{"id": "batch_req_123", "custom_id": "request-7", "response": null, "error": {"code": "batch_ex
```

Other Resources

For more concrete examples, visit **the OpenAI Cookbook**, which contains sample code for use cases like classification, sentiment analysis, and summary generation.