Documentation

Batch Processing

Groq Batch API

Process large-scale workloads asynchronously with our Batch API.

What is Batch Processing?

Batch processing lets you run thousands of API requests at scale by submitting your workload as a batch to Groq and letting us process it with a 24-hour turnaround.

While synchronous API calls are perfect for our fast inference speed for realtime applications, asynchronous batch processing is perfect for use cases where volume of data matters more than synchronous responses, such as processing large datasets, generating content in bulk, and running evaluations. Compared to using our synchronous endpoints, Batch API has:

- **Higher rate limits:** Substantially increased limits compared to on-demand APIs
- **24-hour (or less) turnaround:** Each batch completes within 24 hours (or often more quickly)

Model Availability

The Batch API can currently be used to execute queries for text inputs with llama-3.3-70b-versatile and llama-3.1-8b-instant.

Model Pricing

MODEL	INPUT TOKEN PRICE (PER MILLION TOKENS)	OUTPUT TOKEN PRICE (PER MILLION TOKENS)
llama-3.1-8b-instant	$0.0375	$0.0600
llama-3.3-70b-versatile	$0.442	$0.592

Getting Started

1. Prepare Your Batch File

A batch is composed of a list of API requests and every batch job starts with a JSON Lines (JSONL) file that contains the requests you want processed. Each line in this file represents a single API call.

The Groq Batch API currently supports chat completion requests through `/v1/chat/completions`.
The structure for each line must include:

- `custom_id`: Your unique identifier for tracking the batch request
- `method`: The HTTP method (currently `POST` only)
- `url`: The API endpoint to call (`/v1/chat/completions`)
- `body`: The parameters of your request matching our synchronous API format. See our API Reference here.

The following is an example of a JSONL batch file:

```
{"custom_id": "request-1", "method": "POST", "url": "/v1/chat/completions", "body": {"model": "llama-3.1-8b-instan
{"custom_id": "request-2", "method": "POST", "url": "/v1/chat/completions", "body": {"model": "llama-3.1-8b-instan
{"custom_id": "request-3", "method": "POST", "url": "/v1/chat/completions", "body": {"model": "llama-3.1-8b-instan
```

Converting Sync Calls to Batch Format

If you're familiar with making synchronous API calls, converting them to batch format is straightforward. Here's how a regular API call transforms into a batch request:

```
# Your typical synchronous API call:
response = client.chat.completions.create(
    model="llama-3.1-8b-instant",
    messages=[
        {"role": "user", "content": "What is quantum computing?"}
    ]
```

```
)

# The same call in batch format (must be on a single line as JSONL):
{"custom_id": "quantum-1", "method": "POST", "url": "/v1/chat/completions", "body": {"model": "llama-3.1-8b-instan
```

2. Upload Your Batch File

Upload your `.jsonl` batch file using the Files API endpoint for when kicking off your batch job:

Note: The Files API currently only supports `.jsonl` files up ot 200MB in size.

Python JavaScript curl

```python
import requests # pip install requests first!

def upload_file_to_groq(api_key, file_path):
    url = "https://api.groq.com/openai/v1/files"

    headers = {
        "Authorization": f"Bearer {api_key}"
    }

    # Prepare the file and form data
    files = {
        "file": ("batch_file.jsonl", open(file_path, "rb"))
    }

    data = {
        "purpose": "batch"
    }

    # Make the POST request
    response = requests.post(url, headers=headers, files=files, data=data)

    return response.json()

# Usage example
api_key = "YOUR_GROQ_API_KEY"  # Replace with your actual API key
file_path = "batch_file.jsonl"  # Path to your JSONL file

try:
    result = upload_file_to_groq(api_key, file_path)
    print(result)
except Exception as e:
    print(f"Error: {e}")
```

You will receive a JSON response that contains the ID (`id`) for your file object that you will then use to create your batch job:

```
{
"id":"file_01jh6x76wtemjr74t1fh0faj5t",
"object":"file",
"bytes":966,
"created_at":1736472501,
"filename":"input_file.jsonl",
"purpose":"batch"
}
```

3. Create Your Batch Job

Once you've uploaded your `.jsonl` file, you can use the file object ID (in this case, `file_01jh6x76wtemjr74t1fh0faj5t` as shown in Step 2) to create a batch:

Note: For now, the completion window for batch jobs can only be set to 24 hours (`24h`).

Python JavaScript curl

```python
import requests # pip install requests first!

def create_batch(api_key, input_file_id):
    url = "https://api.groq.com/openai/v1/batches"

    headers = {
        "Authorization": f"Bearer {api_key}",
        "Content-Type": "application/json"
    }

    data = {
        "input_file_id": input_file_id,
        "endpoint": "/v1/chat/completions",
        "completion_window": "24h"
    }

    response = requests.post(url, headers=headers, json=data)
    return response.json()

# Usage example
api_key = "YOUR_GROQ_API_KEY"
file_id = "file_01jh6x76wtemjr74t1fh0faj5t" # replace with your `id` from file upload API response object

try:
    result = create_batch(api_key, file_id)
    print(result)
except Exception as e:
    print(f"Error: {e}")
```

This request will return a Batch object with metadata about your batch, including the batch `id` that you can use to check the status of your batch:

```json
{
"id":"batch_01jh6xa7reempvjyh6n3yst2zw",
"object":"batch",
"endpoint":"/v1/chat/completions",
"errors":null,
"input_file_id":"file_01jh6x76wtemjr74t1fh0faj5t",
"completion_window":"24h",
"status":"validating",
"output_file_id":null,
"error_file_id":null,
"finalizing_at":null,
"failed_at":null,
"expired_at":null,
"cancelled_at":null,
"request_counts":{
    "total":0,
    "completed":0,
    "failed":0
},
"metadata":null,
"created_at":1736472600,
"expires_at":1736559000,
"cancelling_at":null,
"completed_at":null,
"in_progress_at":null
}
```

4. Check Batch Status

You can check the status of a batch any time your heart desires with the batch `id` (in this case, `batch_01jh6xa7reempvjyh6n3yst2zw` from the above Batch response object), which will also return a Batch object:

Python JavaScript curl

```python
import requests # pip install requests first!

def get_batch_status(api_key, batch_id):
    url = f"https://api.groq.com/openai/v1/batches/{batch_id}"

    headers = {
        "Authorization": f"Bearer {api_key}",
        "Content-Type": "application/json"
    }

    response = requests.get(url, headers=headers)
    return response.json()

# Usage example
api_key = "YOUR_GROQ_API_KEY"
batch_id = "batch_01jh6xa7reempvjyh6n3yst2zw"

try:
    result = get_batch_status(api_key, batch_id)
    print(result)
except Exception as e:
    print(f"Error: {e}")
```

The status of a given batch job can return any of the following status codes:

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

When your batch job is complete, the Batch object will return an `output_file_id` and/or an `error_file_id` that you can then use to retrieve your results (as shown below in Step 5). Here's an example:

```
{
    "id":"batch_01jh6xa7reempvjyh6n3yst2zw",
    "object":"batch",
    "endpoint":"/v1/chat/completions",
    "errors":[
        {
            "code":"invalid_method",
            "message":"Invalid value: 'GET'. Supported values are: 'POST'","param":"method",
            "line":4
        }
    ],
    "input_file_id":"file_01jh6x76wtemjr74t1fh0faj5t",
    "completion_window":"24h",
    "status":"completed",
    "output_file_id":"file_01jh6xa97be52b7pg88czwrrwb",
    "error_file_id":"file_01jh6xa9cte52a5xjnmnt5y0je",
    "finalizing_at":null,
    "failed_at":null,
    "expired_at":null,
    "cancelled_at":null,
    "request_counts":{
        "total":3,
        "completed":2,
        "failed":1
    },
    "metadata":null,
    "created_at":1736472600,
    "expires_at":1736559000,
    "cancelling_at":null,
    "completed_at":1736472607,
    "in_progress_at":1736472601
}
```

5. Retrieve Batch Results

Now for the fun. Once the batch is complete, you can retrieve the results using the `output_file_id` from your Batch object (in this case, `file_01jh6xa97be52b7pg88czwrrwb` from the above Batch response object) and write it to a file on your machine (`batch_output.jsonl` in this case) to view them:

Python JavaScript curl

```python
import requests # pip install requests first!

def download_file_content(api_key, output_file_id, output_file):
    url = f"https://api.groq.com/openai/v1/files/{output_file_id}/content"

    headers = {
        "Authorization": f"Bearer {api_key}"
    }

    response = requests.get(url, headers=headers)

    # Write the content to a file
    with open(output_file, 'wb') as f:
        f.write(response.content)

    return f"File downloaded successfully to {output_file}"

# Usage example
api_key = "YOUR_GROQ_API_KEY"
output_file_id = "file_01jh6xa97be52b7pg88czwrrwb" # replace with your own completed batch job's `output_file_
output_file = "batch_output.jsonl" # replace with your own file of choice to download batch job contents to

try:
    result = download_file_content(api_key, file_id, output_file)
    print(result)
except Exception as e:
    print(f"Error: {e}")
```

The `output.jsonl` file will have one response line per successful request line of your batch file. Each line includes the original `custom_id` for mapping results, a unique batch request ID, and the response:

```
{"id": "batch_req_123", "custom_id": "my-request-1", "response": {"status_code": 200, "request_id": "req_abc", "bo
```

Any failed requests in the batch will have their error information written to an error file that can be accessed via the batch's `error_file_id`.

Note: Results may not appears in the same order as your batch request submissions. Always use the `custom_id` field to match results with your original request.

List Batches

You can view all your batch jobs by making a call to `https://api.groq.com/v1/batches`:

```
curl https://api.groq.com/v1/batches \
  -H "Authorization: Bearer $GROQ_API_KEY" \
  -H "Content-Type: application/json"
```

Batch Expiration

Input, intermediate files, and results from processed batches will be stored securely for up to 30 days in Groq's systems. You may also immediately delete once a processed batch is retrieved.

Rate limits

The Batch API rate limits are separate than existing per-model rate limits for synchronous requests. Using the Batch API will not consume tokens from your standard per-model limits, which means you can conveniently leverage batch processing to increase the number of tokens you process with us.

See your limits here.