Batch processing

Copy page

Batch processing is a powerful approach for handling large volumes of requests efficiently. Instead of processing requests one at a time with immediate responses, batch processing allows you to submit multiple requests together for asynchronous processing. This pattern is particularly useful when:

- You need to process large volumes of data
- Immediate responses are not required
- You want to optimize for cost efficiency
- You're running large-scale evaluations or analyses

The Message Batches API is our first implementation of this pattern.

Message Batches API

The Message Batches API is a powerful, cost-effective way to asynchronously process large volumes of Messages requests. This approach is well-suited to tasks that do not require immediate responses, with most batches finishing in less than 1 hour while reducing costs by 50% and increasing throughput.

You can explore the API reference directly, in addition to this guide.

How the Message Batches API works

When you send a request to the Message Batches API:

1. The system creates a new Message Batch with the provided Messages requests.
2. The batch is then processed asynchronously, with each request handled independently.

This is especially useful for bulk operations that don't require immediate results, such as:

- Large-scale evaluations: Process thousands of test cases efficiently.
- Content moderation: Analyze large volumes of user-generated content asynchronously.
- Data analysis: Generate insights or summaries for large datasets.
- Bulk content generation: Create large amounts of text for various purposes (e.g., product descriptions, article summaries).

Batch limitations

- A Message Batch is limited to either 100,000 Message requests or 256 MB in size, whichever is reached first.
- We process each batch as fast as possible, with most batches completing within 1 hour. You will be able to access batch results when all messages have completed or after 24 hours, whichever comes first. Batches will expire if processing does not complete within 24 hours.
- Batch results are available for 29 days after creation. After that, you may still view the Batch, but its results will no longer be available for download.
- Batches are scoped to a Workspace. You may view all batches—and their results—that were created within the Workspace that your API key belongs to.
- Rate limits apply to both Batches API HTTP requests and the number of requests within a batch waiting to be processed. See Message Batches API rate limits. Additionally, we may slow down processing based on current demand and your request volume. In that case, you may see more requests expiring after 24 hours.
- Due to high throughput and concurrent processing, batches may go slightly over your Workspace's configured spend limit.

Supported models

All active models support the Message Batches API.

What can be batched

Any request that you can make to the Messages API can be included in a batch. This includes:

- Vision

- Multi-turn conversations
- Any beta features

Since each request in the batch is processed independently, you can mix different types of requests within a single batch.

> Since batches can take longer than 5 minutes to process, consider using the 1-hour cache duration with prompt caching for better cache hit rates when processing batches with shared context.

Pricing

The Batches API offers significant cost savings. All usage is charged at 50% of the standard API prices.

Model	Batch input	Batch output
Claude Opus 4.5	$2.50 / MTok	$12.50 / MTok
Claude Opus 4.1	$7.50 / MTok	$37.50 / MTok
Claude Opus 4	$7.50 / MTok	$37.50 / MTok
Claude Sonnet 4.5	$1.50 / MTok	$7.50 / MTok
Claude Sonnet 4	$1.50 / MTok	$7.50 / MTok
Claude Sonnet 3.7 (deprecated)	$1.50 / MTok	$7.50 / MTok
Claude Haiku 4.5	$0.50 / MTok	$2.50 / MTok
Claude Haiku 3.5	$0.40 / MTok	$2 / MTok
Claude Opus 3 (deprecated)	$7.50 / MTok	$37.50 / MTok
Claude Haiku 3	$0.125 / MTok	$0.625 / MTok

Claude Docs

Prepare and create your batch

A Message Batch is composed of a list of requests to create a Message. The shape of an individual request is comprised of:

- A unique `custom_id` for identifying the Messages request
- A `params` object with the standard Messages API parameters

You can create a batch by passing this list into the `requests` parameter:

Shell

```
curl https://api.anthropic.com/v1/messages/batches \
     --header "x-api-key: $ANTHROPIC_API_KEY" \
     --header "anthropic-version: 2023-06-01" \
     --header "content-type: application/json" \
     --data \
'{
    "requests": [
        {
            "custom_id": "my-first-request",
            "params": {
                "model": "claude-sonnet-4-5",
                "max_tokens": 1024,
                "messages": [
                    {"role": "user", "content": "Hello, world"}
                ]
            }
        },
        {
            "custom_id": "my-second-request",
            "params": {
                "model": "claude-sonnet-4-5",
                "max_tokens": 1024,
                "messages": [
                    {"role": "user", "content": "Hi again, friend"}
                ]
            }
        }
    ]
}'
```

you'd use for a Messages API call.

Test your batch requests with the Messages API

Validation of the `params` object for each message request is performed asynchronously, and validation errors are returned when processing of the entire batch has ended. You can ensure that you are building your input correctly by verifying your request shape with the Messages API first.

When a batch is first created, the response will have a processing status of `in_progress`.

JSON

```json
{
  "id": "msgbatch_01HkcTjaV5uDC8jWR4ZsDV8d",
  "type": "message_batch",
  "processing_status": "in_progress",
  "request_counts": {
    "processing": 2,
    "succeeded": 0,
    "errored": 0,
    "canceled": 0,
    "expired": 0
  },
  "ended_at": null,
  "created_at": "2024-09-24T18:37:24.100435Z",
  "expires_at": "2024-09-25T18:37:24.100435Z",
  "cancel_initiated_at": null,
  "results_url": null
}
```

Tracking your batch

The Message Batch's `processing_status` field indicates the stage of processing the batch is in. It starts as `in_progress`, then updates to `ended` once all the requests in the batch have finished processing, and results are ready. You can monitor the state of your batch by visiting the Console, or using the retrieval endpoint.

Polling for Message Batch completion

batch status periodically until processing has ended:

Python

```python
import anthropic
import time

client = anthropic.Anthropic()

message_batch = None
while True:
    message_batch = client.messages.batches.retrieve(
        MESSAGE_BATCH_ID
    )
    if message_batch.processing_status == "ended":
        break

    print(f"Batch {MESSAGE_BATCH_ID} is still processing...")
    time.sleep(60)
print(message_batch)
```

Listing all Message Batches

You can list all Message Batches in your Workspace using the list endpoint. The API supports pagination, automatically fetching additional pages as needed:

Python

```python
import anthropic

client = anthropic.Anthropic()

# Automatically fetches more pages as needed.
for message_batch in client.messages.batches.list(
    limit=20
):
    print(message_batch)
```

Retrieving batch results

Claude Docs

Result Type	Description
`succeeded`	Request was successful. Includes the message result.
`errored`	Request encountered an error and a message was not created. Possible errors include invalid requests and internal server errors. You will not be billed for these requests.
`canceled`	User canceled the batch before this request could be sent to the model. You will not be billed for these requests.
`expired`	Batch reached its 24 hour expiration before this request could be sent to the model. You will not be billed for these requests.

You will see an overview of your results with the batch's `request_counts`, which shows how many requests reached each of these four states.

Results of the batch are available for download at the `results_url` property on the Message Batch, and if the organization permission allows, in the Console. Because of the potentially large size of the results, it's recommended to stream results back rather than download them all at once.

Shell

```
    --header "anthropic-version: 2023-06-01" \
    --header "x-api-key: $ANTHROPIC_API_KEY" \
    | grep -o '"results_url":[[:space:]]*"[^"]*"' \
    | cut -d'"' -f4 \
    | while read -r url; do
      curl -s "$url" \
        --header "anthropic-version: 2023-06-01" \
        --header "x-api-key: $ANTHROPIC_API_KEY" \
        | sed 's/}{/}\n{/g' \
        | while IFS= read -r line
      do
        result_type=$(echo "$line" | sed -n 's/.*"result":[[:space:]]*{[[:space:
        custom_id=$(echo "$line" | sed -n 's/.*"custom_id":[[:space:]]*"\([^"]*\
        error_type=$(echo "$line" | sed -n 's/.*"error":[[:space:]]*{[[:space:]]

        case "$result_type" in
          "succeeded")
            echo "Success! $custom_id"
            ;;
          "errored")
            if [ "$error_type" = "invalid_request" ]; then
              # Request body must be fixed before re-sending request
              echo "Validation error: $custom_id"
            else
              # Request can be retried directly
              echo "Server error: $custom_id"
            fi
            ;;
          "expired")
            echo "Expired: $line"
            ;;
        esac
      done
    done
```

The results will be in `.jsonl` format, where each line is a valid JSON object representing the result of a single request in the Message Batch. For each streamed result, you can do something different depending on its `custom_id` and result type. Here is an example set of results:

.jsonl file

If your result has an error, its `result.error` will be set to our standard error shape.

Batch results may not match input order

Batch results can be returned in any order, and may not match the ordering of requests when the batch was created. In the above example, the result for the second batch request is returned before the first. To correctly match results with their corresponding requests, always use the `custom_id` field.

Canceling a Message Batch

You can cancel a Message Batch that is currently processing using the cancel endpoint. Immediately after cancellation, a batch's `processing_status` will be `canceling`. You can use the same polling technique described above to wait until cancellation is finalized. Canceled batches end up with a status of `ended` and may contain partial results for requests that were processed before cancellation.

Python

```python
import anthropic

client = anthropic.Anthropic()

message_batch = client.messages.batches.cancel(
    MESSAGE_BATCH_ID,
)
print(message_batch)
```

The response will show the batch in a `canceling` state:

JSON

```
    "type": "message_batch",
    "processing_status": "canceling",
    "request_counts": {
      "processing": 2,
      "succeeded": 0,
      "errored": 0,
      "canceled": 0,
      "expired": 0
    },
    "ended_at": null,
    "created_at": "2024-09-24T18:37:24.100435Z",
    "expires_at": "2024-09-25T18:37:24.100435Z",
    "cancel_initiated_at": "2024-09-24T18:39:03.114875Z",
    "results_url": null
  }
```

Using prompt caching with Message Batches

The Message Batches API supports prompt caching, allowing you to potentially reduce costs and processing time for batch requests. The pricing discounts from prompt caching and Message Batches can stack, providing even greater cost savings when both features are used together. However, since batch requests are processed asynchronously and concurrently, cache hits are provided on a best-effort basis. Users typically experience cache hit rates ranging from 30% to 98%, depending on their traffic patterns.

To maximize the likelihood of cache hits in your batch requests:

1. Include identical `cache_control` blocks in every Message request within your batch
2. Maintain a steady stream of requests to prevent cache entries from expiring after their 5-minute lifetime
3. Structure your requests to share as much cached content as possible

Example of implementing prompt caching in a batch:

Shell

Claude Docs

Claude Docs

```
     --header "content-type: application/json" \
     --data \
'{
    "requests": [
        {
            "custom_id": "my-first-request",
            "params": {
                "model": "claude-sonnet-4-5",
                "max_tokens": 1024,
                "system": [
                    {
                        "type": "text",
                        "text": "You are an AI assistant tasked with analyzing
                    },
                    {
                        "type": "text",
                        "text": "<the entire contents of Pride and Prejudice>"
                        "cache_control": {"type": "ephemeral"}
                    }
                ],
                "messages": [
                    {"role": "user", "content": "Analyze the major themes in P
                ]
            }
        },
        {
            "custom_id": "my-second-request",
            "params": {
                "model": "claude-sonnet-4-5",
                "max_tokens": 1024,
                "system": [
                    {
                        "type": "text",
                        "text": "You are an AI assistant tasked with analyzing
                    },
                    {
                        "type": "text",
                        "text": "<the entire contents of Pride and Prejudice>"
                        "cache_control": {"type": "ephemeral"}
                    }
                ],
                "messages": [
                    {"role": "user", "content": "Write a summary of Pride and
                ]
            }
        }
    ]
```

In this example, both requests in the batch include identical system messages and the full text of Pride and Prejudice marked with `cache_control` to increase the likelihood of cache hits.

Best practices for effective batching

To get the most out of the Batches API:

- Monitor batch processing status regularly and implement appropriate retry logic for failed requests.
- Use meaningful `custom_id` values to easily match results with requests, since order is not guaranteed.
- Consider breaking very large datasets into multiple batches for better manageability.
- Dry run a single request shape with the Messages API to avoid validation errors.

Troubleshooting common issues

If experiencing unexpected behavior:

- Verify that the total batch request size doesn't exceed 256 MB. If the request size is too large, you may get a 413 `request_too_large` error.
- Check that you're using supported models for all requests in the batch.
- Ensure each request in the batch has a unique `custom_id` .
- Ensure that it has been less than 29 days since batch `created_at` (not processing `ended_at`) time. If over 29 days have passed, results will no longer be viewable.
- Confirm that the batch has not been canceled.

Note that the failure of one request in a batch does not affect the processing of other requests.

Batch storage and privacy

- **Workspace isolation**: Batches are isolated within the Workspace they are created in. They can only be accessed by API keys associated with that Workspace, or users with permission to view Workspace batches in the Console.

Claude Docs

FAQ

- How long does it take for a batch to process?
- Is the Batches API available for all models?
- Can I use the Message Batches API with other API features?
- How does the Message Batches API affect pricing?
- Can I update a batch after it's been submitted?
- Are there Message Batches API rate limits and do they interact with the Messages API rate limits?
- How do I handle errors in my batch requests?
- How does the Message Batches API handle privacy and data separation?
- Can I use prompt caching in the Message Batches API?

Claude Docs

Solutions	Learn
AI agents	Blog

Claude Docs