Context editing

Automatically manage conversation context as it grows with context editing.

Copy page

Overview

Context editing allows you to automatically manage conversation context as it grows, helping you optimize costs and stay within context window limits. You can use server-side API strategies, client-side SDK features, or both together.

Approach	Where it runs	Strategies	How it works
Server-side	API	Tool result clearing (`clear_tool_uses_20250919`) Thinking block clearing (`clear_thinking_20251015`)	Applied before the prompt reaches Claude. Clears specific content from conversation history. Each strategy can be configured independently.
Client-side	SDK	Compaction	Available in Python and TypeScript SDKs when using `tool_runner` . Generates a summary and replaces full conversation history. See Compaction below.

Server-side strategies

Context editing is currently in beta with support for tool result clearing and thinking block clearing. To enable it, use the beta header `context-management-2025-06-27` in your API requests.

Please reach out through our feedback form to share your feedback on this feature.

Tool result clearing

the oldest tool results in chronological order, replacing them with placeholder text to let Claude know the tool result was removed. By default, only tool results are cleared. You can optionally clear both tool results and tool calls (the tool use parameters) by setting `clear_tool_inputs` to true.

Thinking block clearing

The `clear_thinking_20251015` strategy manages `thinking` blocks in conversations when extended thinking is enabled. This strategy automatically clears older thinking blocks from previous turns.

Default behavior: When extended thinking is enabled without configuring the `clear_thinking_20251015` strategy, the API automatically keeps only the thinking blocks from the last assistant turn (equivalent to `keep: {type: "thinking_turns", value: 1}`).

To maximize cache hits, preserve all thinking blocks by setting `keep: "all"`.

An assistant conversation turn may include multiple content blocks (e.g. when using tools) and multiple thinking blocks (e.g. with interleaved thinking).

Context editing happens server-side

Context editing is applied **server-side** before the prompt reaches Claude. Your client application maintains the full, unmodified conversation history—you do not need to sync your client state with the edited version. Continue managing your full conversation history locally as you normally would.

Context editing and prompt caching

Context editing's interaction with prompt caching varies by strategy:

- **Tool result clearing**: Invalidates cached prompt prefixes when content is cleared. To account for this, we recommend clearing enough tokens to make the cache invalidation worthwhile. Use the `clear_at_least` parameter to ensure a minimum number of tokens is cleared each time. You'll incur cache write costs each time content is cleared, but subsequent requests can reuse the newly cached prefix.
- **Thinking block clearing**: When thinking blocks are **kept** in context (not cleared), the prompt cache is preserved, enabling cache hits and reducing input token costs. When

performance of context window availability.

Supported models

Context editing is available on:

- Claude Opus 4.5 (`claude-opus-4-5-20251101`)
- Claude Opus 4.1 (`claude-opus-4-1-20250805`)
- Claude Opus 4 (`claude-opus-4-20250514`)
- Claude Sonnet 4.5 (`claude-sonnet-4-5-20250929`)
- Claude Sonnet 4 (`claude-sonnet-4-20250514`)
- Claude Haiku 4.5 (`claude-haiku-4-5-20251001`)

Tool result clearing usage

The simplest way to enable tool result clearing is to specify only the strategy type, as all other configuration options will use their default values:

cURL

```
    --header "anthropic-version: 2023-06-01" \
    --header "content-type: application/json" \
    --header "anthropic-beta: context-management-2025-06-27" \
    --data '{
        "model": "claude-sonnet-4-5",
        "max_tokens": 4096,
        "messages": [
            {
                "role": "user",
                "content": "Search for recent developments in AI"
            }
        ],
        "tools": [
            {
                "type": "web_search_20250305",
                "name": "web_search"
            }
        ],
        "context_management": {
            "edits": [
                {"type": "clear_tool_uses_20250919"}
            ]
        }
    }'
```

Advanced configuration

You can customize the tool result clearing behavior with additional parameters:

cURL

Claude Docs

```
    --header "anthropic-version: 2023-06-01" \
    --header "content-type: application/json" \
    --header "anthropic-beta: context-management-2025-06-27" \
    --data '{
        "model": "claude-sonnet-4-5",
        "max_tokens": 4096,
        "messages": [
            {
                "role": "user",
                "content": "Create a simple command line calculator app using
            }
        ],
        "tools": [
            {
                "type": "text_editor_20250728",
                "name": "str_replace_based_edit_tool",
                "max_characters": 10000
            },
            {
                "type": "web_search_20250305",
                "name": "web_search",
                "max_uses": 3
            }
        ],
        "context_management": {
            "edits": [
                {
                    "type": "clear_tool_uses_20250919",
                    "trigger": {
                        "type": "input_tokens",
                        "value": 30000
                    },
                    "keep": {
                        "type": "tool_uses",
                        "value": 3
                    },
                    "clear_at_least": {
                        "type": "input_tokens",
                        "value": 5000
                    },
                    "exclude_tools": ["web_search"]
                }
            ]
        }
    }'
```

Claude Docs

Enable thinking block clearing to manage context and prompt caching effectively when extended thinking is enabled:

cURL

```
curl https://api.anthropic.com/v1/messages \
    --header "x-api-key: $ANTHROPIC_API_KEY" \
    --header "anthropic-version: 2023-06-01" \
    --header "content-type: application/json" \
    --header "anthropic-beta: context-management-2025-06-27" \
    --data '{
        "model": "claude-sonnet-4-5-20250929",
        "max_tokens": 1024,
        "messages": [...],
        "thinking": {
            "type": "enabled",
            "budget_tokens": 10000
        },
        "context_management": {
            "edits": [
                {
                    "type": "clear_thinking_20251015",
                    "keep": {
                        "type": "thinking_turns",
                        "value": 2
                    }
                }
            ]
        }
    }'
```

Configuration options for thinking block clearing

The `clear_thinking_20251015` strategy supports the following configuration:

Configuration option	Default	Description
`keep`	`{type: "thinking_turns", value: 1}`	Defines how many recent assistant turns with thinking blocks to preserve. Use `{type: "thinking_turns", value: N}` where N must be > 0 to keep the last N turns, or `"all"` to keep all thinking blocks.

```
// Keep thinking blocks from the last 3 assistant turns
{
  "type": "clear_thinking_20251015",
  "keep": {
    "type": "thinking_turns",
    "value": 3
  }
}

// Keep all thinking blocks (maximizes cache hits)
{
  "type": "clear_thinking_20251015",
  "keep": "all"
}
```

Combining strategies

You can use both thinking block clearing and tool result clearing together:

> When using multiple strategies, the `clear_thinking_20251015` strategy must be listed first in the `edits` array.

Python

```
    max_tokens=1024,
    messages=[...],
    thinking={
        "type": "enabled",
        "budget_tokens": 10000
    },
    tools=[...],
    betas=["context-management-2025-06-27"],
    context_management={
        "edits": [
            {
                "type": "clear_thinking_20251015",
                "keep": {
                    "type": "thinking_turns",
                    "value": 2
                }
            },
            {
                "type": "clear_tool_uses_20250919",
                "trigger": {
                    "type": "input_tokens",
                    "value": 50000
                },
                "keep": {
                    "type": "tool_uses",
                    "value": 5
                }
            }
        ]
    }
)
```

Configuration options for tool result clearing

Configuration option	Default	Description
`trigger`	100,000 input tokens	Defines when the context editing strategy activates. Once the prompt exceeds this threshold, clearing will begin. You can specify this value in either `input_tokens` or `tool_uses`.
`keep`	3 tool uses	Defines how many recent tool use/result pairs to keep after clearing occurs. The API removes the oldest tool interactions first, preserving the most recent ones.

`clear_at_least`	None	strategy activates. If the API can't clear at least the specified amount, the strategy will not be applied. This helps determine if context clearing is worth breaking your prompt cache.
`exclude_tools`	None	List of tool names whose tool uses and results should never be cleared. Useful for preserving important context.
`clear_tool_inputs`	`false`	Controls whether the tool call parameters are cleared along with the tool results. By default, only the tool results are cleared while keeping Claude's original tool calls visible.

Context editing response

You can see which context edits were applied to your request using the `context_management` response field, along with helpful statistics about the content and input tokens cleared.

Response

```
{
    "id": "msg_013Zva2CMHLNnXjNJJKqJ2EF",
    "type": "message",
    "role": "assistant",
    "content": [...],
    "usage": {...},
    "context_management": {
        "applied_edits": [
            // When using `clear_thinking_20251015`
            {
                "type": "clear_thinking_20251015",
                "cleared_thinking_turns": 3,
                "cleared_input_tokens": 15000
            },
            // When using `clear_tool_uses_20250919`
            {
                "type": "clear_tool_uses_20250919",
                "cleared_tool_uses": 8,
                "cleared_input_tokens": 50000
            }
        ]
    }
}
```

Claude Docs

Streaming Response

```
{
    "type": "message_delta",
    "delta": {
        "stop_reason": "end_turn",
        "stop_sequence": null
    },
    "usage": {
        "output_tokens": 1024
    },
    "context_management": {
        "applied_edits": [...]
    }
}
```

Token counting

The token counting endpoint supports context management, allowing you to preview how many tokens your prompt will use after context editing is applied.

cURL

```
    --header "anthropic-version: 2023-06-01" \
    --header "content-type: application/json" \
    --header "anthropic-beta: context-management-2025-06-27" \
    --data '{
        "model": "claude-sonnet-4-5",
        "messages": [
            {
                "role": "user",
                "content": "Continue our conversation..."
            }
        ],
        "tools": [...],
        "context_management": {
            "edits": [
                {
                    "type": "clear_tool_uses_20250919",
                    "trigger": {
                        "type": "input_tokens",
                        "value": 30000
                    },
                    "keep": {
                        "type": "tool_uses",
                        "value": 5
                    }
                }
            ]
        }
    }'
```

Response

```
{
    "input_tokens": 25000,
    "context_management": {
        "original_input_tokens": 70000
    }
}
```

The response shows both the final token count after context management is applied (`input_tokens`) and the original token count before any clearing occurred (`original_input_tokens`).

Claude Docs

Context editing can be combined with the memory tool. When your conversation context approaches the configured clearing threshold, Claude receives an automatic warning to preserve important information. This enables Claude to save tool results or context to its memory files before they're cleared from the conversation history.

This combination allows you to:

- **Preserve important context**: Claude can write essential information from tool results to memory files before those results are cleared
- **Maintain long-running workflows**: Enable agentic workflows that would otherwise exceed context limits by offloading information to persistent storage
- **Access information on demand**: Claude can look up previously cleared information from memory files when needed, rather than keeping everything in the active context window

For example, in a file editing workflow where Claude performs many operations, Claude can summarize completed changes to memory files as the context grows. When tool results are cleared, Claude retains access to that information through its memory system and can continue working effectively.

To use both features together, enable them in your API request:

Python

```python
response = client.beta.messages.create(
    model="claude-sonnet-4-5",
    max_tokens=4096,
    messages=[...],
    tools=[
        {
            "type": "memory_20250818",
            "name": "memory"
        },
        # Your other tools
    ],
    betas=["context-management-2025-06-27"],
    context_management={
        "edits": [
            {"type": "clear_tool_uses_20250919"}
        ]
    }
)
```

Claude Docs

> Compaction is available in the Python and TypeScript SDKs when using the `tool_runner` method.

Compaction is an SDK feature that automatically manages conversation context by generating summaries when token usage grows too large. Unlike server-side context editing strategies that clear content, compaction instructs Claude to summarize the conversation history, then replaces the full history with that summary. This allows Claude to continue working on long-running tasks that would otherwise exceed the context window.

How compaction works

When compaction is enabled, the SDK monitors token usage after each model response:

1. **Threshold check**: The SDK calculates total tokens as `input_tokens + cache_creation_input_tokens + cache_read_input_tokens + output_tokens`
2. **Summary generation**: When the threshold is exceeded, a summary prompt is injected as a user turn, and Claude generates a structured summary wrapped in `<summary> </summary>` tags
3. **Context replacement**: The SDK extracts the summary and replaces the entire message history with it
4. **Continuation**: The conversation resumes from the summary, with Claude picking up where it left off

Using compaction

Add `compaction_control` to your `tool_runner` call:

Python

Claude Docs

```python
client = anthropic.Anthropic()

runner = client.beta.messages.tool_runner(
    model="claude-sonnet-4-5",
    max_tokens=4096,
    tools=[...],
    messages=[
        {
            "role": "user",
            "content": "Analyze all the files in this directory and write a su
        }
    ],
    compaction_control={
        "enabled": True,
        "context_token_threshold": 100000
    }
)

for message in runner:
    print(f"Tokens used: {message.usage.input_tokens}")

final = runner.until_done()
```

What happens during compaction

As the conversation grows, the message history accumulates:

Before compaction (approaching 100k tokens):

```
[
  { "role": "user", "content": "Analyze all files and write a report..." },
  { "role": "assistant", "content": "I'll help. Let me start by reading..." },
  { "role": "user", "content": [{ "type": "tool_result", "tool_use_id": "...",
  { "role": "assistant", "content": "Based on file1.txt, I see..." },
  { "role": "user", "content": [{ "type": "tool_result", "tool_use_id": "...",
  { "role": "assistant", "content": "After analyzing file2.txt..." },
  // ... 50 more exchanges like this ...
]
```

When tokens exceed the threshold, the SDK injects a summary request and Claude generates a summary. The entire history is then replaced:

```
[
  {
    "role": "assistant",
    "content": "# Task Overview\nThe user requested analysis of directory file
  }
]
```

Claude continues working from this summary as if it were the original conversation history.

Configuration options

Parameter	Type	Required	Default	Description
`enabled`	boolean	Yes	-	Whether to enable automatic compaction
`context_token_threshold`	number	No	100,000	Token count at which compaction triggers
`model`	string	No	Same as main model	Model to use for generating summaries
`summary_prompt`	string	No	See below	Custom prompt for summary generation

Choosing a token threshold

The threshold determines when compaction occurs. A lower threshold means more frequent compactions with smaller context windows. A higher threshold allows more context but risks hitting limits.

Python

```
    "enabled": True,
    "context_token_threshold": 50000
}

# Less frequent compaction when you need more context
compaction_control={
    "enabled": True,
    "context_token_threshold": 150000
}
```

Using a different model for summaries

You can use a faster or cheaper model for generating summaries:

Python

```
compaction_control={
    "enabled": True,
    "context_token_threshold": 100000,
    "model": "claude-haiku-4-5"
}
```

Custom summary prompts

You can provide a custom prompt for domain-specific needs. Your prompt should instruct Claude to wrap its summary in `<summary></summary>` tags.

Python

```
compaction_control={
    "enabled": True,
    "context_token_threshold": 100000,
    "summary_prompt": """Summarize the research conducted so far, including:
- Sources consulted and key findings
- Questions answered and remaining unknowns
- Recommended next steps

Wrap your summary in <summary></summary> tags."""
}
```

The built-in summary prompt instructs Claude to create a structured continuation summary including:

1. **Task Overview**: The user's core request, success criteria, and constraints
2. **Current State**: What has been completed, files modified, and artifacts produced
3. **Important Discoveries**: Technical constraints, decisions made, errors resolved, and failed approaches
4. **Next Steps**: Specific actions needed, blockers, and priority order
5. **Context to Preserve**: User preferences, domain-specific details, and commitments made

This structure enables Claude to resume work efficiently without losing important context or repeating mistakes.

> **View full default prompt**

Limitations

Server-side tools

⚠ Compaction requires special consideration when using server-side tools such as web search or web fetch.

When using server-side tools, the SDK may incorrectly calculate token usage, causing compaction to trigger at the wrong time.

For example, after a web search operation, the API response might show:

```
{
  "usage": {
    "input_tokens": 63000,
    "cache_read_input_tokens": 270000,
    "output_tokens": 1400
  }
}
```

calls made by the server-side tool, not your actual conversation context. Your real context length might only be the 63,000 `input_tokens`, but the SDK sees 333k and triggers compaction prematurely.

Workarounds:

- Use the token counting endpoint to get accurate context length
- Avoid compaction when using server-side tools extensively

Tool use edge cases

When compaction is triggered while a tool use response is pending, the SDK removes the tool use block from the message history before generating the summary. Claude will re-issue the tool call after resuming from the summary if still needed.

Monitoring compaction

Enable logging to track when compaction occurs:

Python

```python
import logging

logging.basicConfig(level=logging.INFO)
logging.getLogger("anthropic.lib.tools").setLevel(logging.INFO)

# Logs will show:
# INFO: Token usage 105000 has exceeded the threshold of 100000. Performing co
# INFO: Compaction complete. New token usage: 2500
```

When to use compaction

Good use cases:

- Long-running agent tasks that process many files or data sources
- Research workflows that accumulate large amounts of information
- Multi-step tasks with clear, measurable progress
- Tasks that produce artifacts (files, reports) that persist outside the conversation

Less ideal use cases:

Claude Docs

- Tasks that need to maintain exact state across many variables

Claude Docs

Claude Docs

Terms and policies

Privacy policy

Responsible disclosure policy

Terms of service: Commercial

Terms of service: Consumer

Usage policy