Building with extended thinking

Copy page

Extended thinking gives Claude enhanced reasoning capabilities for complex tasks, while providing varying levels of transparency into its step-by-step thought process before it delivers its final answer.

Supported models

Extended thinking is supported in the following models:

- Claude Sonnet 4.5 (`claude-sonnet-4-5-20250929`)
- Claude Sonnet 4 (`claude-sonnet-4-20250514`)
- Claude Sonnet 3.7 (`claude-3-7-sonnet-20250219`) (deprecated)
- Claude Haiku 4.5 (`claude-haiku-4-5-20251001`)
- 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`)

> API behavior differs across Claude Sonnet 3.7 and Claude 4 models, but the API shapes remain exactly the same.
>
> For more information, see Differences in thinking across model versions.

How extended thinking works

When extended thinking is turned on, Claude creates `thinking` content blocks where it outputs its internal reasoning. Claude incorporates insights from this reasoning before crafting a final response.

The API response will include `thinking` content blocks, followed by `te` blocks.

Ask Docs

```
{
  "content": [
    {
      "type": "thinking",
      "thinking": "Let me analyze this step by step...",
      "signature": "WaUjzkypQ2mUEVM36O2TxuC06KN8xyfbJwyem2dw3URve/op91XWHOEBLL
    },
    {
      "type": "text",
      "text": "Based on my analysis..."
    }
  ]
}
```


For more information about the response format of extended thinking, see the Messages API Reference.

How to use extended thinking

Here is an example of using extended thinking in the Messages API:

Shell

Ask Docs

```
     --header "anthropic-version: 2023-06-01" \
     --header "content-type: application/json" \
     --data \
'{
    "model": "claude-sonnet-4-5",
    "max_tokens": 16000,
    "thinking": {
        "type": "enabled",
        "budget_tokens": 10000
    },
    "messages": [
        {
            "role": "user",
            "content": "Are there an infinite number of prime numbers such tha
        }
    ]
}'
```

To turn on extended thinking, add a `thinking` object, with the `type` parameter set to `enabled` and the `budget_tokens` to a specified token budget for extended thinking.

The `budget_tokens` parameter determines the maximum number of tokens Claude is allowed to use for its internal reasoning process. In Claude 4 models, this limit applies to full thinking tokens, and not to the summarized output. Larger budgets can improve response quality by enabling more thorough analysis for complex problems, although Claude may not use the entire budget allocated, especially at ranges above 32k.

`budget_tokens` must be set to a value less than `max_tokens`. However, when using interleaved thinking with tools, you can exceed this limit as the token limit becomes your entire context window (200k tokens).

Summarized thinking

With extended thinking enabled, the Messages API for Claude 4 models returns a summary of Claude's full thinking process. Summarized thinking provides the full intelligence benefits of extended thinking, while preventing misuse.

Here are some important considerations for summarized thinking:

- You're charged for the full thinking tokens generated by the original request, not the summary tokens.
- The billed output token count will **not match** the count of tokens you see in the response.

- As Anthropic seeks to improve the extended thinking feature, summarization behavior is subject to change.
- Summarization preserves the key ideas of Claude's thinking process with minimal added latency, enabling a streamable user experience and easy migration from Claude Sonnet 3.7 to Claude 4 models.
- Summarization is processed by a different model than the one you target in your requests. The thinking model does not see the summarized output.

> Claude Sonnet 3.7 continues to return full thinking output.
>
> In rare cases where you need access to full thinking output for Claude 4 models, contact our sales team.

Streaming thinking

You can stream extended thinking responses using server-sent events (SSE).

When streaming is enabled for extended thinking, you receive thinking content via `thinking_delta` events.

For more documention on streaming via the Messages API, see Streaming Messages.

Here's how to handle streaming with thinking:

Shell

```
     --header "anthropic-version: 2023-06-01" \
     --header "content-type: application/json" \
     --data \
'{
    "model": "claude-sonnet-4-5",
    "max_tokens": 16000,
    "stream": true,
    "thinking": {
        "type": "enabled",
        "budget_tokens": 10000
    },
    "messages": [
        {
            "role": "user",
            "content": "What is 27 * 453?"
        }
    ]
}'
```

Try in Console

Example streaming output:

```
event: content_block_start
data: {"type": "content_block_start", "index": 0, "content_block": {"type": "t

event: content_block_delta
data: {"type": "content_block_delta", "index": 0, "delta": {"type": "thinking_

event: content_block_delta
data: {"type": "content_block_delta", "index": 0, "delta": {"type": "thinking_

// Additional thinking deltas...

event: content_block_delta
data: {"type": "content_block_delta", "index": 0, "delta": {"type": "signature

event: content_block_stop
data: {"type": "content_block_stop", "index": 0}

event: content_block_start
data: {"type": "content_block_start", "index": 1, "content_block": {"type": "t

event: content_block_delta
data: {"type": "content_block_delta", "index": 1, "delta": {"type": "text_delt

// Additional text deltas...

event: content_block_stop
data: {"type": "content_block_stop", "index": 1}

event: message_delta
data: {"type": "message_delta", "delta": {"stop_reason": "end_turn", "stop_seq

event: message_stop
data: {"type": "message_stop"}
```

When using streaming with thinking enabled, you might notice that text sometimes arrives in larger chunks alternating with smaller, token-by-token delivery. This is expected behavior, especially for thinking content.

The streaming system needs to process content in batches for optimal performance, which can result in this "chunky" delivery pattern, with possible delays between streaming events. We're continuously working to improve this experience, with future updates focused on making thinking content stream more smoothly.

Extended thinking can be used alongside tool use, allowing Claude to reason through tool selection and results processing.

When using extended thinking with tool use, be aware of the following limitations:

1. **Tool choice limitation**: Tool use with thinking only supports `tool_choice: {"type": "auto"}` (the default) or `tool_choice: {"type": "none"}`. Using `tool_choice: {"type": "any"}` or `tool_choice: {"type": "tool", "name": "..."}` will result in an error because these options force tool use, which is incompatible with extended thinking.
2. **Preserving thinking blocks**: During tool use, you must pass `thinking` blocks back to the API for the last assistant message. Include the complete unmodified block back to the API to maintain reasoning continuity.

Toggling thinking modes in conversations

You cannot toggle thinking in the middle of an assistant turn, including during tool use loops. The entire assistant turn must operate in a single thinking mode:

- **If thinking is enabled**, the final assistant turn must start with a thinking block.
- **If thinking is disabled**, the final assistant turn must not contain any thinking blocks

From the model's perspective, **tool use loops are part of the assistant turn**. An assistant turn doesn't complete until Claude finishes its full response, which may include multiple tool calls and results.

For example, this sequence is all part of a **single assistant turn**:

```
User: "What's the weather in Paris?"
Assistant: [thinking] + [tool_use: get_weather]
User: [tool_result: "20°C, sunny"]
Assistant: [text: "The weather in Paris is 20°C and sunny"]
```

Even though there are multiple API messages, the tool use loop is conceptually part of one continuous assistant response.

Common error scenarios

You might encounter this error:

```
with a thinking block (preceding the lastmost set of `tool_use` and
`tool_result` blocks).
```

This typically occurs when:

1. You had thinking **disabled** during a tool use sequence
2. You want to enable thinking again
3. Your last assistant message contains tool use blocks but no thinking block

Practical guidance

✗ Invalid: Toggling thinking immediately after tool use

```
User: "What's the weather?"
Assistant: [tool_use] (thinking disabled)
User: [tool_result]
// Cannot enable thinking here - still in the same assistant turn
```

✓ Valid: Complete the assistant turn first

```
User: "What's the weather?"
Assistant: [tool_use] (thinking disabled)
User: [tool_result]
Assistant: [text: "It's sunny"]
User: "What about tomorrow?" (thinking disabled)
Assistant: [thinking] + [text: "..."] (thinking enabled - new turn)
```

Best practice: Plan your thinking strategy at the start of each turn rather than trying to toggle mid-turn.

ⓘ Toggling thinking modes also invalidates prompt caching for message history. For more details, see the Extended thinking with prompt caching section.

Example: Passing thinking blocks with tool results

During tool use, you must pass `thinking` blocks back to the API, and you must include the complete unmodified block back to the API. This is critical for maintaining the model's reasoning flow and conversation integrity.

> While you can omit `thinking` blocks from prior `assistant` role turns, we suggest always passing back all thinking blocks to the API for any multi-turn conversation. The API will:
>
> - Automatically filter the provided thinking blocks
> - Use the relevant thinking blocks necessary to preserve the model's reasoning
> - Only bill for the input tokens for the blocks shown to Claude

> When toggling thinking modes during a conversation, remember that the entire assistant turn (including tool use loops) must operate in a single thinking mode. For more details, see Toggling thinking modes in conversations.

When Claude invokes tools, it is pausing its construction of a response to await external information. When tool results are returned, Claude will continue building that existing response. This necessitates preserving thinking blocks during tool use, for a couple of reasons:

1. **Reasoning continuity**: The thinking blocks capture Claude's step-by-step reasoning that led to tool requests. When you post tool results, including the original thinking ensures Claude can continue its reasoning from where it left off.
2. **Context maintenance**: While tool results appear as user messages in the API structure, they're part of a continuous reasoning flow. Preserving thinking blocks maintains this conceptual flow across multiple API calls. For more information on context management, see our guide on context windows.

Important: When providing `thinking` blocks, the entire sequence of consecutive `thinking` blocks must match the outputs generated by the model during the original request; you cannot rearrange or modify the sequence of these blocks.

Interleaved thinking

Extended thinking with tool use in Claude 4 models supports interleaved thinking, which enables Claude to think between tool calls and make more sophisticated reasoning after receiving tool results. Ask Docs

- Chain multiple tool calls with reasoning steps in between
- Make more nuanced decisions based on intermediate results

To enable interleaved thinking, add the beta header `interleaved-thinking-2025-05-14` to your API request.

Here are some important considerations for interleaved thinking:

- With interleaved thinking, the `budget_tokens` can exceed the `max_tokens` parameter, as it represents the total budget across all thinking blocks within one assistant turn.
- Interleaved thinking is only supported for tools used via the Messages API.
- Interleaved thinking is supported for Claude 4 models only, with the beta header `interleaved-thinking-2025-05-14 .`
- Direct calls to the Claude API allow you to pass `interleaved-thinking-2025-05-14` in requests to any model, with no effect.
- On 3rd-party platforms (e.g., Amazon Bedrock and Vertex AI), if you pass `interleaved-thinking-2025-05-14` to any model aside from Claude Opus 4.5, Claude Opus 4.1, Opus 4, or Sonnet 4, your request will fail.

Tool use without interleaved thinking

Tool use with interleaved thinking

Extended thinking with prompt caching

Prompt caching with thinking has several important considerations:

> Extended thinking tasks often take longer than 5 minutes to complete. Consider using the 1-hour cache duration to maintain cache hits across longer thinking sessions and multi-step workflows.

Thinking block context removal

- Thinking blocks from previous turns are removed from context, which can affect cache breakpoints

- This creates a tradeoff: while thinking blocks don't consume context window space visually, they still count toward your input token usage when cached
- If thinking becomes disabled, requests will fail if you pass thinking content in the current tool use turn. In other contexts, thinking content passed to the API is simply ignored

Cache invalidation patterns

- Changes to thinking parameters (enabled/disabled or budget allocation) invalidate message cache breakpoints
- Interleaved thinking amplifies cache invalidation, as thinking blocks can occur between multiple tool calls
- System prompts and tools remain cached despite thinking parameter changes or block removal

> While thinking blocks are removed for caching and context calculations, they must be preserved when continuing conversations with tool use, especially with interleaved thinking.

Understanding thinking block caching behavior

When using extended thinking with tool use, thinking blocks exhibit specific caching behavior that affects token counting:

How it works:

1. Caching only occurs when you make a subsequent request that includes tool results
2. When the subsequent request is made, the previous conversation history (including thinking blocks) can be cached
3. These cached thinking blocks count as input tokens in your usage metrics when read from the cache
4. When a non-tool-result user block is included, all previous thinking blocks are ignored and stripped from context

Detailed example flow:

Request 1:

```
User: "What's the weather in Paris?"
```

Ask Docs

```
[thinking_block_1] + [tool_use block 1]
```

Request 2:

```
User: ["What's the weather in Paris?"],
Assistant: [thinking_block_1] + [tool_use block 1],
User: [tool_result_1, cache=True]
```

Response 2:

```
[thinking_block_2] + [text block 2]
```

Request 2 writes a cache of the request content (not the response). The cache includes the original user message, the first thinking block, tool use block, and the tool result.

Request 3:

```
User: ["What's the weather in Paris?"],
Assistant: [thinking_block_1] + [tool_use block 1],
User: [tool_result_1, cache=True],
Assistant: [thinking_block_2] + [text block 2],
User: [Text response, cache=True]
```

For Claude Opus 4.5 and later, all previous thinking blocks are kept by default. For older models, because a non-tool-result user block was included, all previous thinking blocks are ignored. This request will be processed the same as:

```
User: ["What's the weather in Paris?"],
Assistant: [tool_use block 1],
User: [tool_result_1, cache=True],
Assistant: [text block 2],
User: [Text response, cache=True]
```

Key points:

- This caching behavior happens automatically, even without explicit `cache_control` markers

- **System prompt caching (preserved when thinking changes)**
- **Messages caching (invalidated when thinking changes)**

Max tokens and context window size with extended thinking

In older Claude models (prior to Claude Sonnet 3.7), if the sum of prompt tokens and `max_tokens` exceeded the model's context window, the system would automatically adjust `max_tokens` to fit within the context limit. This meant you could set a large `max_tokens` value and the system would silently reduce it as needed.

With Claude 3.7 and 4 models, `max_tokens` (which includes your thinking budget when thinking is enabled) is enforced as a strict limit. The system will now return a validation error if prompt tokens + `max_tokens` exceeds the context window size.

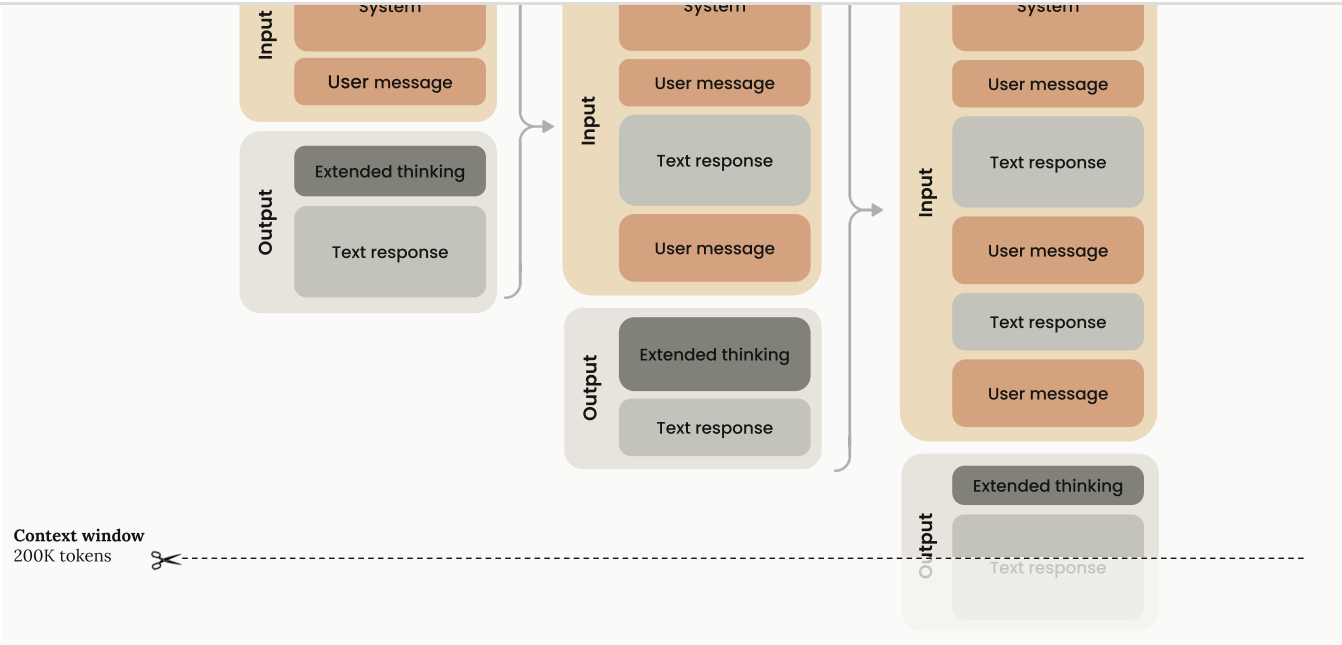
> You can read through our guide on context windows for a more thorough deep dive.

The context window with extended thinking

When calculating context window usage with thinking enabled, there are some considerations to be aware of:

- Thinking blocks from previous turns are stripped and not counted towards your context window
- Current turn thinking counts towards your `max_tokens` limit for that turn

The diagram below demonstrates the specialized token management when extended thinking is enabled:



The effective context window is calculated as:

```
context window =
  (current input tokens - previous thinking tokens) +
  (thinking tokens + encrypted thinking tokens + text output tokens)
```

We recommend using the token counting API to get accurate token counts for your specific use case, especially when working with multi-turn conversations that include thinking.

The context window with extended thinking and tool use

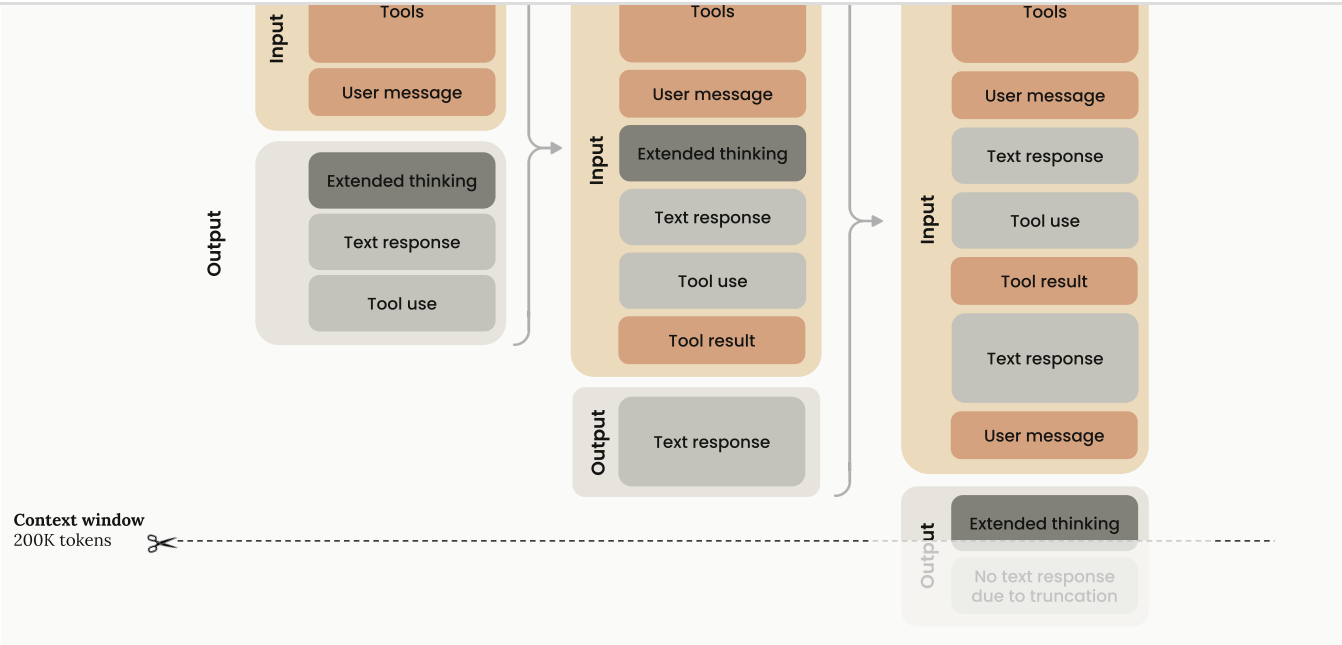
When using extended thinking with tool use, thinking blocks must be explicitly preserved and returned with the tool results.

The effective context window calculation for extended thinking with tool use becomes:

```
context window =
  (current input tokens + previous thinking tokens + tool use tokens) +
  (thinking tokens + encrypted thinking tokens + text output tokens)
```

The diagram below illustrates token management for extended thinking with tool use:

Ask Docs



Managing tokens with extended thinking

Given the context window and `max_tokens` behavior with extended thinking Claude 3.7 and 4 models, you may need to:

- More actively monitor and manage your token usage
- Adjust `max_tokens` values as your prompt length changes
- Potentially use the token counting endpoints more frequently
- Be aware that previous thinking blocks don't accumulate in your context window

This change has been made to provide more predictable and transparent behavior, especially as maximum token limits have increased significantly.

Thinking encryption

Full thinking content is encrypted and returned in the `signature` field. This field is used to verify that thinking blocks were generated by Claude when passed back to the API.

> It is only strictly necessary to send back thinking blocks when using tools with extended thinking. Otherwise you can omit thinking blocks from previous turns, or let the API strip them for you if you pass them back.
>
> If sending back thinking blocks, we recommend passing everything back as you received it for consistency and to avoid potential issues.

When [streaming responses](), the signature is added via a `signature_delta` inside a `content_block_delta` event just before the `content_block_stop` event.

- `signature` values are significantly longer in Claude 4 models than in previous models.
- The `signature` field is an opaque field and should not be interpreted or parsed - it exists solely for verification purposes.
- `signature` values are compatible across platforms (Claude APIs, Amazon Bedrock, and Vertex AI). Values generated on one platform will be compatible with another.

Thinking redaction

Occasionally Claude's internal reasoning will be flagged by our safety systems. When this occurs, we encrypt some or all of the `thinking` block and return it to you as a `redacted_thinking` block. `redacted_thinking` blocks are decrypted when passed back to the API, allowing Claude to continue its response without losing context.

When building customer-facing applications that use extended thinking:

- Be aware that redacted thinking blocks contain encrypted content that isn't human-readable
- Consider providing a simple explanation like: "Some of Claude's internal reasoning has been automatically encrypted for safety reasons. This doesn't affect the quality of responses."
- If showing thinking blocks to users, you can filter out redacted blocks while preserving normal thinking blocks
- Be transparent that using extended thinking features may occasionally result in some reasoning being encrypted
- Implement appropriate error handling to gracefully manage redacted thinking without breaking your UI

Here's an example showing both normal and redacted thinking blocks:

```
    {
      "type": "thinking",
      "thinking": "Let me analyze this step by step...",
      "signature": "WaUjzkypQ2mUEVM36O2TxuC06KN8xyfbJwyem2dw3URve/op91XWHOEBLL
    },
    {
      "type": "redacted_thinking",
      "data": "EmwKAhgBEgy3va3pzix/LafPsn4aDFIT2Xlxh0L5L8rLVyIwxtE3rAFBa8cr3qp
    },
    {
      "type": "text",
      "text": "Based on my analysis..."
    }
  ]
}
```

Seeing redacted thinking blocks in your output is expected behavior. The model can still use this redacted reasoning to inform its responses while maintaining safety guardrails.

If you need to test redacted thinking handling in your application, you can use this special test string as your prompt:

```
ANTHROPIC_MAGIC_STRING_TRIGGER_REDACTED_THINKING_46C9A13E193C177646C739
8A98432ECCCE4C1253D5E2D82641AC0E52CC2876CB
```

When passing `thinking` and `redacted_thinking` blocks back to the API in a multi-turn conversation, you must include the complete unmodified block back to the API for the last assistant turn. This is critical for maintaining the model's reasoning flow. We suggest always passing back all thinking blocks to the API. For more details, see the Preserving thinking blocks section above.

Example: Working with redacted thinking blocks

Differences in thinking across model versions

The Messages API handles thinking differently across Claude Sonnet 3.7 and Claude 4 models, primarily in redaction and summarization behavior.

See the table below for a condensed comparison:

Thinking Output	Returns full thinking output	Returns summarized thinking	Returns summarized thinking
Interleaved Thinking	Not supported	Supported with `interleaved-thinking-2025-05-14` beta header	Supported with `interleaved-thinking-2025-05-14` beta header
Thinking Block Preservation	Not preserved across turns	Not preserved across turns	**Preserved by default** (enables cache optimization, token savings)

Thinking block preservation in Claude Opus 4.5

Claude Opus 4.5 introduces a new default behavior: **thinking blocks from previous assistant turns are preserved in model context by default**. This differs from earlier models, which remove thinking blocks from prior turns.

Benefits of thinking block preservation:

- **Cache optimization**: When using tool use, preserved thinking blocks enable cache hits as they are passed back with tool results and cached incrementally across the assistant turn, resulting in token savings in multi-step workflows
- **No intelligence impact**: Preserving thinking blocks has no negative effect on model performance

Important considerations:

- **Context usage**: Long conversations will consume more context space since thinking blocks are retained in context
- **Automatic behavior**: This is the default behavior for Claude Opus 4.5—no code changes or beta headers required
- **Backward compatibility**: To leverage this feature, continue passing complete, unmodified thinking blocks back to the API as you would for tool use

> For earlier models (Claude Sonnet 4.5, Opus 4.1, etc.), thinking blocks from previous turns continue to be removed from context. The existing behavior described in the Extended thinking with prompt caching section applies to those models.

For complete pricing information including base rates, cache writes, cache hits, and output tokens, see the pricing page.

The thinking process incurs charges for:

- Tokens used during thinking (output tokens)
- Thinking blocks from the last assistant turn included in subsequent requests (input tokens)
- Standard text output tokens

ⓘ When extended thinking is enabled, a specialized system prompt is automatically included to support this feature.

When using summarized thinking:

- **Input tokens**: Tokens in your original request (excludes thinking tokens from previous turns)
- **Output tokens (billed)**: The original thinking tokens that Claude generated internally
- **Output tokens (visible)**: The summarized thinking tokens you see in the response
- **No charge**: Tokens used to generate the summary

⚠ The billed output token count will **not** match the visible token count in the response. You are billed for the full thinking process, not the summary you see.

Best practices and considerations for extended thinking

Working with thinking budgets

- **Budget optimization:** The minimum budget is 1,024 tokens. We suggest starting at the minimum and increasing the thinking budget incrementally to find the optimal range for your use case. Higher token counts enable more comprehensive reasoning but with diminishing returns depending on the task. Increasing the budget can improve response quality at the tradeoff of increased latency. For critical tasks, test different settings to find the optimal balance. Note that the thinking budget is a target rather than a strict limit—actual token usage may vary based on the task.
- **Starting points:** Start with larger thinking budgets (16k+ tokens) for complex tasks and adjust based on your needs.

32k tokens causes long running requests that might run up against system timeouts and open connection limits.

- **Token usage tracking:** Monitor thinking token usage to optimize costs and performance.

Performance considerations

- **Response times:** Be prepared for potentially longer response times due to the additional processing required for the reasoning process. Factor in that generating thinking blocks may increase overall response time.
- **Streaming requirements:** Streaming is required when `max_tokens` is greater than 21,333. When streaming, be prepared to handle both thinking and text content blocks as they arrive.

Feature compatibility

- Thinking isn't compatible with `temperature` or `top_k` modifications as well as forced tool use.
- When thinking is enabled, you can set `top_p` to values between 1 and 0.95.
- You cannot pre-fill responses when thinking is enabled.
- Changes to the thinking budget invalidate cached prompt prefixes that include messages. However, cached system prompts and tool definitions will continue to work when thinking parameters change.

Usage guidelines

- **Task selection:** Use extended thinking for particularly complex tasks that benefit from step-by-step reasoning like math, coding, and analysis.
- **Context handling:** You do not need to remove previous thinking blocks yourself. The Claude API automatically ignores thinking blocks from previous turns and they are not included when calculating context usage.
- **Prompt engineering:** Review our extended thinking prompting tips if you want to maximize Claude's thinking capabilities.

Next steps

Explore practical examples of thinking in our cookbook.

</>

Extended thinking prompting tips

Learn prompt engineering best practices for extended thinking.

Claude Docs

Help and security

Availability

Status

Support

Discord

Terms and policies

Privacy policy

Responsible disclosure policy

Terms of service: Commercial

Terms of service: Consumer

Usage policy

Ask Docs