Tool search tool

Copy page

The tool search tool enables Claude to work with hundreds or thousands of tools by dynamically discovering and loading them on-demand. Instead of loading all tool definitions into the context window upfront, Claude searches your tool catalog—including tool names, descriptions, argument names, and argument descriptions—and loads only the tools it needs.

This approach solves two critical challenges as tool libraries scale:

- **Context efficiency**: Tool definitions can consume massive portions of your context window (50 tools ≈ 10-20K tokens), leaving less room for actual work
- **Tool selection accuracy**: Claude's ability to correctly select tools degrades significantly with more than 30-50 conventionally-available tools

Although this is provided as a server-side tool, you can also implement your own client-side tool search functionality. See Custom tool search implementation for details.

The tool search tool is currently in public beta. Include the appropriate beta header for your provider:

Provider	Beta header	Supported models
Claude API Microsoft Foundry	`advanced-tool-use-2025-11-20`	Claude Opus 4.5 Claude Sonnet 4.5
Google Cloud's Vertex AI	`tool-search-tool-2025-10-19`	Claude Opus 4.5 Claude Sonnet 4.5
Amazon Bedrock	`tool-search-tool-2025-10-19`	Claude Opus 4.5

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

On Amazon Bedrock, server-side tool search is available only via the invok
converse API.

Claude Docs

How tool search works

There are two tool search variants:

- **Regex** (`tool_search_tool_regex_20251119`): Claude constructs regex patterns to search for tools
- **BM25** (`tool_search_tool_bm25_20251119`): Claude uses natural language queries to search for tools

When you enable the tool search tool:

1. You include a tool search tool (e.g., `tool_search_tool_regex_20251119` or `tool_search_tool_bm25_20251119`) in your tools list
2. You provide all tool definitions with `defer_loading: true` for tools that shouldn't be loaded immediately
3. Claude sees only the tool search tool and any non-deferred tools initially
4. When Claude needs additional tools, it searches using a tool search tool
5. The API returns 3-5 most relevant `tool_reference` blocks
6. These references are automatically expanded into full tool definitions
7. Claude selects from the discovered tools and invokes them

This keeps your context window efficient while maintaining high tool selection accuracy.

Quick start

Here's a simple example with deferred tools:

Shell

```
    --header "anthropic-beta: advanced-tool-use-2025-11-20" \
    --header "content-type: application/json" \
    --data '{
        "model": "claude-sonnet-4-5-20250929",
        "max_tokens": 2048,
        "messages": [
            {
                "role": "user",
                "content": "What is the weather in San Francisco?"
            }
        ],
        "tools": [
            {
                "type": "tool_search_tool_regex_20251119",
                "name": "tool_search_tool_regex"
            },
            {
                "name": "get_weather",
                "description": "Get the weather at a specific location",
                "input_schema": {
                    "type": "object",
                    "properties": {
                        "location": {"type": "string"},
                        "unit": {
                            "type": "string",
                            "enum": ["celsius", "fahrenheit"]
                        }
                    },
                    "required": ["location"]
                },
                "defer_loading": true
            },
            {
                "name": "search_files",
                "description": "Search through files in the workspace",
                "input_schema": {
                    "type": "object",
                    "properties": {
                        "query": {"type": "string"},
                        "file_types": {
                            "type": "array",
                            "items": {"type": "string"}
                        }
                    },
                    "required": ["query"]
                },
                "defer_loading": true
    }
```

Tool definition

The tool search tool has two variants:

JSON

```json
{
  "type": "tool_search_tool_regex_20251119",
  "name": "tool_search_tool_regex"
}
```

JSON

```json
{
  "type": "tool_search_tool_bm25_20251119",
  "name": "tool_search_tool_bm25"
}
```

⚠ **Regex variant query format: Python regex, NOT natural language**

When using `tool_search_tool_regex_20251119`, Claude constructs regex patterns using Python's `re.search()` syntax, not natural language queries. Common patterns:

- `"weather"` - matches tool names/descriptions containing "weather"
- `"get_.*_data"` - matches tools like `get_user_data`, `get_weather_data`
- `"database.*query|query.*database"` - OR patterns for flexibility
- `"(?i)slack"` - case-insensitive search

Maximum query length: 200 characters

ⓘ **BM25 variant query format: Natural language**

When using `tool_search_tool_bm25_20251119`, Claude uses natural language queries to search for tools.

Mark tools for on-demand loading by adding `defer_loading: true` :

JSON

```json
{
  "name": "get_weather",
  "description": "Get current weather for a location",
  "input_schema": {
    "type": "object",
    "properties": {
      "location": { "type": "string" },
      "unit": { "type": "string", "enum": ["celsius", "fahrenheit"] }
    },
    "required": ["location"]
  },
  "defer_loading": true
}
```

Key points:

- Tools without `defer_loading` are loaded into context immediately
- Tools with `defer_loading: true` are only loaded when Claude discovers them via search
- The tool search tool itself should **never** have `defer_loading: true`
- Keep your 3-5 most frequently used tools as non-deferred for optimal performance

Both tool search variants (`regex` and `bm25`) search tool names, descriptions, argument names, and argument descriptions.

Response format

When Claude uses the tool search tool, the response includes new block types:

JSON

Claude Docs

```
    "content": [
      {
        "type": "text",
        "text": "I'll search for tools to help with the weather information."
      },
      {
        "type": "server_tool_use",
        "id": "srvtoolu_01ABC123",
        "name": "tool_search_tool_regex",
        "input": {
          "query": "weather"
        }
      },
      {
        "type": "tool_search_tool_result",
        "tool_use_id": "srvtoolu_01ABC123",
        "content": {
          "type": "tool_search_tool_search_result",
          "tool_references": [{ "type": "tool_reference", "tool_name": "get_weat
        }
      },
      {
        "type": "text",
        "text": "I found a weather tool. Let me get the weather for San Francisc
      },
      {
        "type": "tool_use",
        "id": "toolu_01XYZ789",
        "name": "get_weather",
        "input": { "location": "San Francisco", "unit": "fahrenheit" }
      }
    ],
    "stop_reason": "tool_use"
  }
```

Understanding the response

- **`server_tool_use`** : Indicates Claude is invoking the tool search tool
- **`tool_search_tool_result`** : Contains the search results with a nested `tool_search_tool_search_result` object
- **`tool_references`** : Array of `tool_reference` objects pointing to discovered tools
- **`tool_use`** : Claude invoking the discovered tool

automatically in the API as long as you provide all matching tool definitions in the `tools` parameter.

MCP integration

The tool search tool works with MCP servers. Add the `"mcp-client-2025-11-20"` beta header to your API request, and then use `mcp_toolset` with `default_config` to defer loading MCP tools:

Shell

```
    --header "anthropic-version: 2023-06-01" \
    --header "anthropic-beta: advanced-tool-use-2025-11-20,mcp-client-2025-11-20
    --header "content-type: application/json" \
    --data '{
      "model": "claude-sonnet-4-5-20250929",
      "max_tokens": 2048,
      "mcp_servers": [
        {
          "type": "url",
          "name": "database-server",
          "url": "https://mcp-db.example.com"
        }
      ],
      "tools": [
        {
          "type": "tool_search_tool_regex_20251119",
          "name": "tool_search_tool_regex"
        },
        {
          "type": "mcp_toolset",
          "mcp_server_name": "database-server",
          "default_config": {
            "defer_loading": true
          },
          "configs": {
            "search_events": {
              "defer_loading": false
            }
          }
        }
      ],
      "messages": [
        {
          "role": "user",
          "content": "What events are in my database?"
        }
      ]
    }'
```

MCP configuration options:

- `default_config.defer_loading` : Set default for all tools from the MCP server
- `configs` : Override defaults for specific tools by name
- Combine multiple MCP servers with tool search for massive tool libraries

You can implement your own tool search logic (e.g., using embeddings or semantic search) by returning `tool_reference` blocks from a custom tool. When Claude calls your custom search tool, return a standard `tool_result` with `tool_reference` blocks in the content array:

JSON

```json
{
  "type": "tool_result",
  "tool_use_id": "toolu_your_tool_id",
  "content": [
    { "type": "tool_reference", "tool_name": "discovered_tool_name" }
  ]
}
```

Every tool referenced must have a corresponding tool definition in the top-level `tools` parameter with `defer_loading: true`. This approach lets you use more sophisticated search algorithms while maintaining compatibility with the tool search system.

> The `tool_search_tool_result` format shown in the Response format section is the server-side format used internally by Anthropic's built-in tool search. For custom client-side implementations, always use the standard `tool_result` format with `tool_reference` content blocks as shown above.

For a complete example using embeddings, see our tool search with embeddings cookbook.

Error handling

> The tool search tool is not compatible with tool use examples. If you need to provide examples of tool usage, use standard tool calling without tool search.

HTTP errors (400 status)

These errors prevent the request from being processed:

All tools deferred:

Claude Docs

```
  "error": {
    "type": "invalid_request_error",
    "message": "All tools have defer_loading set. At least one tool must be no
  }
}
```

Missing tool definition:

```
{
  "type": "error",
  "error": {
    "type": "invalid_request_error",
    "message": "Tool reference 'unknown_tool' has no corresponding tool defini
  }
}
```

Tool result errors (200 status)

Errors during tool execution return a 200 response with error information in the body:

JSON

```json
{
  "type": "tool_result",
  "tool_use_id": "srvtoolu_01ABC123",
  "content": {
    "type": "tool_search_tool_result_error",
    "error_code": "invalid_pattern"
  }
}
```

Error codes:

- `too_many_requests` : Rate limit exceeded for tool search operations
- `invalid_pattern` : Malformed regex pattern
- `pattern_too_long` : Pattern exceeds 200 character limit
- `unavailable` : Tool search service temporarily unavailable

Claude Docs

400 Error: All tools are deferred

400 Error: Missing tool definition

Claude doesn’t find expected tools

Prompt caching

Tool search works with prompt caching. Add `cache_control` breakpoints to optimize multi-turn conversations:

Python

```python
# First request with tool search
messages = [
    {
        "role": "user",
        "content": "What's the weather in Seattle?"
    }
]

response1 = client.beta.messages.create(
    model="claude-sonnet-4-5-20250929",
    betas=["advanced-tool-use-2025-11-20"],
    max_tokens=2048,
    messages=messages,
    tools=[
        {
            "type": "tool_search_tool_regex_20251119",
            "name": "tool_search_tool_regex"
        },
        {
            "name": "get_weather",
            "description": "Get weather for a location",
            "input_schema": {
                "type": "object",
                "properties": {
                    "location": {"type": "string"}
                },
                "required": ["location"]
            },
            "defer_loading": True
        }
    ]
)

# Add Claude's response to conversation
messages.append({
    "role": "assistant",
    "content": response1.content
})

# Second request with cache breakpoint
messages.append({
    "role": "user",
    "content": "What about New York?",
    "cache_control": {"type": "ephemeral"}
})
```

Claude Docs

```
    betas=["advanced-tool-use-2025-11-20"],
    max_tokens=2048,
    messages=messages,
    tools=[
        {
            "type": "tool_search_tool_regex_20251119",
            "name": "tool_search_tool_regex"
        },
        {
            "name": "get_weather",
            "description": "Get weather for a location",
            "input_schema": {
                "type": "object",
                "properties": {
                    "location": {"type": "string"}
                },
                "required": ["location"]
```

Streaming

```
event: content_block_start
data: {"type": "content_block_start", "index": 1, "content_block": {"type": "s

// Search query streamed
event: content_block_delta
data: {"type": "content_block_delta", "index": 1, "delta": {"type": "input_jsc

// Pause while search executes

// Search results streamed
event: content_block_start
data: {"type": "content_block_start", "index": 2, "content_block": {"type": "t

// Claude continues with discovered tools
```

Batch requests

You can include the tool search tool in the Messages Batches API. Tool search operations through the Messages Batches API are priced the same as those in regular Messages API requests.

Limits and best practices

Limits

- **Maximum tools:** 10,000 tools in your catalog

- **Model support**: Sonnet 4.0+, Opus 4.0+ only (no Haiku)

When to use tool search

Good use cases:

- 10+ tools available in your system
- Tool definitions consuming >10K tokens
- Experiencing tool selection accuracy issues with large tool sets
- Building MCP-powered systems with multiple servers (200+ tools)
- Tool library growing over time

When traditional tool calling might be better:

- Less than 10 tools total
- All tools are frequently used in every request
- Very small tool definitions (<100 tokens total)

Optimization tips

- Keep 3-5 most frequently used tools as non-deferred
- Write clear, descriptive tool names and descriptions
- Use semantic keywords in descriptions that match how users describe tasks
- Add a system prompt section describing available tool categories: "You can search for tools to interact with Slack, GitHub, and Jira"
- Monitor which tools Claude discovers to refine descriptions

Usage

Tool search tool usage is tracked in the response usage object:

JSON

```
    "input_tokens": 1024,
    "output_tokens": 256,
    "server_tool_use": {
      "tool_search_requests": 2
    }
  }
}
```

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