Computer use tool

Copy page

Claude can interact with computer environments through the computer use tool, which provides screenshot capabilities and mouse/keyboard control for autonomous desktop interaction.

Computer use is currently in beta and requires a beta header:

- `"computer-use-2025-11-24"` for Claude Opus 4.5
- `"computer-use-2025-01-24"` for Claude Sonnet 4.5, Haiku 4.5, Opus 4.1, Sonnet 4, Opus 4, and Sonnet 3.7 (deprecated)

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

Overview

Computer use is a beta feature that enables Claude to interact with desktop environments. This tool provides:

- **Screenshot capture**: See what's currently displayed on screen
- **Mouse control**: Click, drag, and move the cursor
- **Keyboard input**: Type text and use keyboard shortcuts
- **Desktop automation**: Interact with any application or interface

While computer use can be augmented with other tools like bash and text editor for more comprehensive automation workflows, computer use specifically refers to the computer use tool's capability to see and control desktop environments.

Model compatibility

Computer use is available for the following Claude models:

Ask Docs

All other supported models	`computer_20250124`	`computer-use-2025-01-24`

ⓘ Claude Opus 4.5 introduces the `computer_20251124` tool version with new capabilities including the zoom action for detailed screen region inspection. All other models (Sonnet 4.5, Haiku 4.5, Sonnet 4, Opus 4, Opus 4.1, and Sonnet 3.7) use the `computer_20250124` tool version.

⚠ Older tool versions are not guaranteed to be backwards-compatible with newer models. Always use the tool version that corresponds to your model version.

Security considerations

Computer use is a beta feature with unique risks distinct from standard API features. These risks are heightened when interacting with the internet.

⚠ To minimize risks, consider taking precautions such as:

1. Using a dedicated virtual machine or container with minimal privileges to prevent direct system attacks or accidents.
2. Avoiding giving the model access to sensitive data, such as account login information, to prevent information theft.
3. Limiting internet access to an allowlist of domains to reduce exposure to malicious content.
4. Asking a human to confirm decisions that may result in meaningful real-world consequences as well as any tasks requiring affirmative consent, such as accepting cookies, executing financial transactions, or agreeing to terms of service.

In some circumstances, Claude will follow commands found in content even if it conflicts with the user's instructions. For example, Claude instructions on webpages or contained in images may override instructions or cause Claude to make mistakes. We suggest taking precautions to isolate Claude from sensitive data and actions to avoid risks related to prompt injection.

We've trained the model to resist these prompt injections and have added an extra layer of defense. If you use our computer use tools, we'll automatically run classifiers on your prompts to flag potential instances of prompt injections. When these classifiers identify

extra protection won't be ideal for every use case (for example, use cases without a human in the loop), so if you'd like to opt out and turn it off, please contact us.

We still suggest taking precautions to isolate Claude from sensitive data and actions to avoid risks related to prompt injection.

Finally, please inform end users of relevant risks and obtain their consent prior to enabling computer use in your own products.

Computer use reference implementation

Get started quickly with our computer use reference implementation that includes a web interface, Docker container, example tool implementations, and an agent loop.

Note: The implementation has been updated to include new tools for both Claude 4 models and Claude Sonnet 3.7. Be sure to pull the latest version of the repo to access these new features.

Please use this form to provide feedback on the quality of the model responses, the API itself, or the quality of the documentation - we cannot wait to hear from you!

Quick start

Here's how to get started with computer use:

Python

Ask Docs

```python
client = anthropic.Anthropic()

response = client.beta.messages.create(
    model="claude-sonnet-4-5",  # or another compatible model
    max_tokens=1024,
    tools=[
        {
          "type": "computer_20250124",
          "name": "computer",
          "display_width_px": 1024,
          "display_height_px": 768,
          "display_number": 1,
        },
        {
          "type": "text_editor_20250728",
          "name": "str_replace_based_edit_tool"
        },
        {
          "type": "bash_20250124",
          "name": "bash"
        }
    ],
    messages=[{"role": "user", "content": "Save a picture of a cat to my deskt
    betas=["computer-use-2025-01-24"]
)
print(response)
```

A beta header is only required for the computer use tool.

The example above shows all three tools being used together, which requires the beta header because it includes the computer use tool.

How computer use works

1 **Provide Claude with the computer use tool and a user prompt**

- Add the computer use tool (and optionally other tools) to your API request

2 **Claude decides to use the computer use tool**

- Claude assesses if the computer use tool can help with the user's query.
- If yes, Claude constructs a properly formatted tool use request.
- The API response has a `stop_reason` of `tool_use`, signaling Claude's intent.

3 **Extract tool input, evaluate the tool on a computer, and return results**

- On your end, extract the tool name and input from Claude's request.
- Use the tool on a container or Virtual Machine.
- Continue the conversation with a new `user` message containing a `tool_result` content block.

4 **Claude continues calling computer use tools until it's completed the task**

- Claude analyzes the tool results to determine if more tool use is needed or the task has been completed.
- If Claude decides it needs another tool, it responds with another `tool_use` `stop_reason` and you should return to step 3.
- Otherwise, it crafts a text response to the user.

We refer to the repetition of steps 3 and 4 without user input as the "agent loop" - i.e., Claude responding with a tool use request and your application responding to Claude with the results of evaluating that request.

The computing environment

Computer use requires a sandboxed computing environment where Claude can safely interact with applications and the web. This environment includes:

1. **Virtual display**: A virtual X11 display server (using Xvfb) that renders the desktop interface Claude will see through screenshots and control with mouse/keyboard actions.
2. **Desktop environment**: A lightweight UI with window manager (Mutter) and panel (Tint2) running on Linux, which provides a consistent graphical interface for Claude to interact with.
3. **Applications**: Pre-installed Linux applications like Firefox, LibreOffice, text editors, and file managers that Claude can use to complete tasks.

virtual environment.

5. **Agent loop**: A program that handles communication between Claude and the environment, sending Claude's actions to the environment and returning the results (screenshots, command outputs) back to Claude.

When you use computer use, Claude doesn't directly connect to this environment. Instead, your application:

1. Receives Claude's tool use requests
2. Translates them into actions in your computing environment
3. Captures the results (screenshots, command outputs, etc.)
4. Returns these results to Claude

For security and isolation, the reference implementation runs all of this inside a Docker container with appropriate port mappings for viewing and interacting with the environment.

How to implement computer use

Start with our reference implementation

We have built a reference implementation that includes everything you need to get started quickly with computer use:

- A containerized environment suitable for computer use with Claude
- Implementations of the computer use tools
- An agent loop that interacts with the Claude API and executes the computer use tools
- A web interface to interact with the container, agent loop, and tools.

Understanding the multi-agent loop

The core of computer use is the "agent loop" - a cycle where Claude requests tool actions, your application executes them, and returns results to Claude. Here's a simplified example:

Claude Docs

Ask Docs

```
    messages: list[dict],
    api_key: str,
    max_tokens: int = 4096,
    tool_version: str,
    thinking_budget: int | None = None,
    max_iterations: int = 10,  # Add iteration limit to prevent infinite loops
):
    """
    A simple agent loop for Claude computer use interactions.

    This function handles the back-and-forth between:
    1. Sending user messages to Claude
    2. Claude requesting to use tools
    3. Your app executing those tools
    4. Sending tool results back to Claude
    """
    # Set up tools and API parameters
    client = Anthropic(api_key=api_key)
    beta_flag = "computer-use-2025-01-24" if "20250124" in tool_version else "

    # Configure tools - you should already have these initialized elsewhere
    tools = [
        {"type": f"computer_{tool_version}", "name": "computer", "display_widt
        {"type": f"text_editor_{tool_version}", "name": "str_replace_editor"},
        {"type": f"bash_{tool_version}", "name": "bash"}
    ]

    # Main agent loop (with iteration limit to prevent runaway API costs)
    iterations = 0
    while True and iterations < max_iterations:
        iterations += 1
        # Set up optional thinking parameter (for Claude Sonnet 3.7)
        thinking = None
        if thinking_budget:
            thinking = {"type": "enabled", "budget_tokens": thinking_budget}

        # Call the Claude API
        response = client.beta.messages.create(
            model=model,
            max_tokens=max_tokens,
            messages=messages,
            tools=tools,
            betas=[beta_flag],
            thinking=thinking
        )

        # Add Claude's response to the conversation history
```

```
        # Check if Claude used any tools
        tool_results = []
        for block in response_content:
            if block.type == "tool_use":
                # In a real app, you would execute the tool here
                # For example: result = run_tool(block.name, block.input)
                result = {"result": "Tool executed successfully"}

                # Format the result for Claude
                tool_results.append({
                    "type": "tool_result",
                    "tool_use_id": block.id,
                    "content": result
                })

        # If no tools were used, Claude is done - return the final messages
        if not tool_results:
            return messages

        # Add tool results to messages for the next iteration with Claude
        messages.append({"role": "user", "content": tool_results})
```

The loop continues until either Claude responds without requesting any tools (task completion) or the maximum iteration limit is reached. This safeguard prevents potential infinite loops that could result in unexpected API costs.

We recommend trying the reference implementation out before reading the rest of this documentation.

Optimize model performance with prompting

Here are some tips on how to get the best quality outputs:

1. Specify simple, well-defined tasks and provide explicit instructions for each step.
2. Claude sometimes assumes outcomes of its actions without explicitly checking their results. To prevent this you can prompt Claude with `After each step, take a screenshot and carefully evaluate if you have achieved the right outcome. Explicitly show your thinking: "I have evaluated step X..." If not correct, try again. Only when you confirm a step was executed correctly should you move on to the next one.`
3. Some UI elements (like dropdowns and scrollbars) might be tricky for Claude to manipulate using mouse movements. If you experience this, try prompting the model

successful outcomes in your prompt.

5. If you need the model to log in, provide it with the username and password in your prompt inside xml tags like `<robot_credentials>`. Using computer use within applications that require login increases the risk of bad outcomes as a result of prompt injection. Please review our guide on mitigating prompt injections before providing the model with login credentials.

> If you repeatedly encounter a clear set of issues or know in advance the tasks Claude will need to complete, use the system prompt to provide Claude with explicit tips or instructions on how to do the tasks successfully.

System prompts

When one of the Anthropic-defined tools is requested via the Claude API, a computer use-specific system prompt is generated. It's similar to the tool use system prompt but starts with:

> *You have access to a set of functions you can use to answer the user's question. This includes access to a sandboxed computing environment. You do NOT currently have the ability to inspect files or interact with external resources, except by invoking the below functions.*

As with regular tool use, the user-provided `system_prompt` field is still respected and used in the construction of the combined system prompt.

Available actions

The computer use tool supports these actions:

Basic actions (all versions)

- **screenshot** - Capture the current display
- **left_click** - Click at coordinates `[x, y]`
- **type** - Type text string
- **key** - Press key or key combination (e.g., "ctrl+s")
- **mouse_move** - Move cursor to coordinates

Enhanced actions (`computer_20250124`) Available in Claude 4 models and Claude Sonnet 3.7:

- **right_click**, **middle_click** - Additional mouse buttons
- **double_click**, **triple_click** - Multiple clicks
- **left_mouse_down**, **left_mouse_up** - Fine-grained click control
- **hold_key** - Hold a key while performing other actions
- **wait** - Pause between actions

Enhanced actions (`computer_20251124`) Available in Claude Opus 4.5:

- All actions from `computer_20250124`
- **zoom** - View a specific region of the screen at full resolution. Requires `enable_zoom: true` in tool definition. Takes a `region` parameter with coordinates `[x1, y1, x2, y2]` defining top-left and bottom-right corners of the area to inspect.

> **Example actions**

Tool parameters

Parameter	Required	Description
`type`	Yes	Tool version (`computer_20251124` , `computer_20250124` , or `computer_20241022`)
`name`	Yes	Must be "computer"
`display_width_px`	Yes	Display width in pixels
`display_height_px`	Yes	Display height in pixels
`display_number`	No	Display number for X11 environments
`enable_zoom`	No	Enable zoom action (`computer_20251124` only). Set to `true` to allow Claude to zoom into specific screen regions. Default: `false`

Important: The computer use tool must be explicitly executed by your application - Claude cannot execute it directly. You are responsible for implementing the screenshot capture, mouse movements, keyboard inputs, and other actions based on Claude's requests.

Claude Sonnet 3.7 introduced a new "thinking" capability that allows you to see the model's reasoning process as it works through complex tasks. This feature helps you understand how Claude is approaching a problem and can be particularly valuable for debugging or educational purposes.

To enable thinking, add a `thinking` parameter to your API request:

```
"thinking": {
  "type": "enabled",
  "budget_tokens": 1024
}
```

The `budget_tokens` parameter specifies how many tokens Claude can use for thinking. This is subtracted from your overall `max_tokens` budget.

When thinking is enabled, Claude will return its reasoning process as part of the response, which can help you:

1. Understand the model's decision-making process
2. Identify potential issues or misconceptions
3. Learn from Claude's approach to problem-solving
4. Get more visibility into complex multi-step operations

Here's an example of what thinking output might look like:

```
[Thinking]
I need to save a picture of a cat to the desktop. Let me break this down into

1. First, I'll take a screenshot to see what's on the desktop
2. Then I'll look for a web browser to search for cat images
3. After finding a suitable image, I'll need to save it to the desktop

Let me start by taking a screenshot to see what's available...
```

Augmenting computer use with other tools

The computer use tool can be combined with other tools to create more powerful automation workflows. This is particularly useful when you need to:

- Integrate with custom APIs or services (custom tools)

Shell

Ask Docs

Claude Docs

Ask Docs

```
-H "anthropic-version: 2023-06-01" \
-H "anthropic-beta: computer-use-2025-01-24" \
-d '{
  "model": "claude-sonnet-4-5",
  "max_tokens": 2000,
  "tools": [
    {
      "type": "computer_20250124",
      "name": "computer",
      "display_width_px": 1024,
      "display_height_px": 768,
      "display_number": 1
    },
    {
      "type": "text_editor_20250728",
      "name": "str_replace_based_edit_tool"
    },
    {
      "type": "bash_20250124",
      "name": "bash"
    },
    {
      "name": "get_weather",
      "description": "Get the current weather in a given location",
      "input_schema": {
        "type": "object",
        "properties": {
          "location": {
            "type": "string",
            "description": "The city and state, e.g. San Francisco, CA"
          },
          "unit": {
            "type": "string",
            "enum": ["celsius", "fahrenheit"],
            "description": "The unit of temperature, either 'celsius' or 'fa
          }
        },
        "required": ["location"]
      }
    }
  ],
  "messages": [
    {
      "role": "user",
      "content": "Find flights from San Francisco to a place with warmer wea
    }
  ],
```

```
        budget_tokens": 1024
    }
}'
```

Build a custom computer use environment

The reference implementation is meant to help you get started with computer use. It includes all of the components needed have Claude use a computer. However, you can build your own environment for computer use to suit your needs. You'll need:

- A virtualized or containerized environment suitable for computer use with Claude
- An implementation of at least one of the Anthropic-defined computer use tools
- An agent loop that interacts with the Claude API and executes the `tool_use` results using your tool implementations
- An API or UI that allows user input to start the agent loop

Implement the computer use tool

The computer use tool is implemented as a schema-less tool. When using this tool, you don't need to provide an input schema as with other tools; the schema is built into Claude's model and can't be modified.

1. **Set up your computing environment**

   Create a virtual display or connect to an existing display that Claude will interact with. This typically involves setting up Xvfb (X Virtual Framebuffer) or similar technology.

2. **Implement action handlers**

   Create functions to handle each action type that Claude might request:

```
        return capture_screenshot()
    elif action_type == "left_click":
        x, y = params["coordinate"]
        return click_at(x, y)
    elif action_type == "type":
        return type_text(params["text"])
    # ... handle other actions
```

3 Process Claude's tool calls

Extract and execute tool calls from Claude's responses:

```
for content in response.content:
    if content.type == "tool_use":
        action = content.input["action"]
        result = handle_computer_action(action, content.input)

        # Return result to Claude
        tool_result = {
            "type": "tool_result",
            "tool_use_id": content.id,
            "content": result
        }
```

4 Implement the agent loop

Create a loop that continues until Claude completes the task:

```
while True:
    response = client.beta.messages.create(...)

    # Check if Claude used any tools
    tool_results = process_tool_calls(response)

    if not tool_results:
        # No more tool use, task complete
        break

    # Continue conversation with tool results
    messages.append({"role": "user", "content": tool_results})
```

When implementing the computer use tool, various errors may occur. Here's how to handle them:

- **Screenshot capture failure**
- **Invalid coordinates**
- **Action execution failure**

Handle coordinate scaling for higher resolutions

The API constrains images to a maximum of 1568 pixels on the longest edge and approximately 1.15 megapixels total (see image resizing for details). For example, a 1512x982 screen gets downsampled to approximately 1330x864. Claude analyzes this smaller image and returns coordinates in that space, but your tool executes clicks in the original screen space.

This can cause Claude's click coordinates to miss their targets unless you handle the coordinate transformation.

To fix this, resize screenshots yourself and scale Claude's coordinates back up:

Python

```python
def get_scale_factor(width, height):
    """Calculate scale factor to meet API constraints."""
    long_edge = max(width, height)
    total_pixels = width * height

    long_edge_scale = 1568 / long_edge
    total_pixels_scale = math.sqrt(1_150_000 / total_pixels)

    return min(1.0, long_edge_scale, total_pixels_scale)

# When capturing screenshot
scale = get_scale_factor(screen_width, screen_height)
scaled_width = int(screen_width * scale)
scaled_height = int(screen_height * scale)

# Resize image to scaled dimensions before sending to Claude
screenshot = capture_and_resize(scaled_width, scaled_height)

# When handling Claude's coordinates, scale them back up
def execute_click(x, y):
    screen_x = x / scale
    screen_y = y / scale
    perform_click(screen_x, screen_y)
```

Follow implementation best practices

- Use appropriate display resolution
- Implement proper screenshot handling
- Add action delays
- Validate actions before execution
- Log actions for debugging

The computer use functionality is in beta. While Claude's capabilities are cutting edge, developers should be aware of its limitations:

1. **Latency**: the current computer use latency for human-AI interactions may be too slow compared to regular human-directed computer actions. We recommend focusing on use cases where speed isn't critical (e.g., background information gathering, automated software testing) in trusted environments.
2. **Computer vision accuracy and reliability**: Claude may make mistakes or hallucinate when outputting specific coordinates while generating actions. Claude Sonnet 3.7 introduces the thinking capability that can help you understand the model's reasoning and identify potential issues.
3. **Tool selection accuracy and reliability**: Claude may make mistakes or hallucinate when selecting tools while generating actions or take unexpected actions to solve problems. Additionally, reliability may be lower when interacting with niche applications or multiple applications at once. We recommend that users prompt the model carefully when requesting complex tasks.
4. **Scrolling reliability**: Claude Sonnet 3.7 introduced dedicated scroll actions with direction control that improves reliability. The model can now explicitly scroll in any direction (up/down/left/right) by a specified amount.
5. **Spreadsheet interaction**: Mouse clicks for spreadsheet interaction have improved in Claude Sonnet 3.7 with the addition of more precise mouse control actions like `left_mouse_down` , `left_mouse_up` , and new modifier key support. Cell selection can be more reliable by using these fine-grained controls and combining modifier keys with clicks.
6. **Account creation and content generation on social and communications platforms**: While Claude will visit websites, we are limiting its ability to create accounts or generate and share content or otherwise engage in human impersonation across social media websites and platforms. We may update this capability in the future.
7. **Vulnerabilities**: Vulnerabilities like jailbreaking or prompt injection may persist across frontier AI systems, including the beta computer use API. In some circumstances, Claude will follow commands found in content, sometimes even in conflict with the user's instructions. For example, Claude instructions on webpages or contained in images may override instructions or cause Claude to make mistakes. We recommend: a. Limiting computer use to trusted environments such as virtual machines or containers with minimal privileges b. Avoiding giving computer use access to sensitive accounts or data without strict oversight c. Informing end users of relevant risks and obtaining their consent before enabling or requesting permissions necessary for computer use features in your applications

Always carefully review and verify Claude's computer use actions and logs. Do not use Claude for tasks requiring perfect precision or sensitive user information without human oversight.

Pricing

Computer use follows the standard tool use pricing. When using the computer use tool:

System prompt overhead: The computer use beta adds 466-499 tokens to the system prompt

Computer use tool token usage:

Model	Input tokens per tool definition
Claude 4.x models	735 tokens
Claude Sonnet 3.7 (deprecated)	735 tokens

Additional token consumption:

- Screenshot images (see Vision pricing)
- Tool execution results returned to Claude

If you're also using bash or text editor tools alongside computer use, those tools have their own token costs as documented in their respective pages.

Next steps

Reference implementation ↗

Get started quickly with our complete Docker-based implementation

Claude Docs

Learn more about tool use and creating custom tools

Claude Docs

Claude Docs

Discord

Terms and policies

Privacy policy

Responsible disclosure policy

Terms of service: Commercial

Terms of service: Consumer

Usage policy

Ask Docs