Copy page

Remote MCP

Allow models to use remote MCP servers to perform tasks.

Model Context Protocol (MCP) is an open protocol that standardizes how applications provide tools and context to LLMs. The MCP tool in the Responses API allows developers to give the model access to tools hosted on **Remote MCP servers**. These are MCP servers maintained by developers and organizations across the internet that expose these tools to MCP clients, like the Responses API.

Calling a remote MCP server with the Responses API is straightforward. For example, here's how you can use the DeepWiki MCP server to ask questions about nearly any public GitHub repository.

A Responses API request with MCP tools enabled

python

```python
from openai import OpenAI

client = OpenAI()

resp = client.responses.create(
    model="gpt-4.1",
    tools=[
        {
            "type": "mcp",
            "server_label": "deepwiki",
            "server_url": "https://mcp.deepwiki.com/mcp",
            "require_approval": "never",
        },
    ],
    input="What transport protocols are supported in the 2025-03-26 version of the MCP
)

print(resp.output_text)
```

It is very important that developers trust any remote MCP server they use with the Responses API. A malicious server can exfiltrate sensitive data from anything that enters the model's context. Carefully review the Risks and Safety section below before using this tool.

The MCP ecosystem

We are still in the early days of the MCP ecosystem. Some popular remote MCP servers today include Cloudflare, Hubspot, Intercom, Paypal, Pipedream, Plaid, Shopify, Stripe, Square, Twilio and Zapier. We expect many more servers—and registries making it easy to discover these servers—to launch in the coming months. The MCP protocol itself is also early, and we expect to add many more updates to our MCP tool as the protocol evolves.

How it works

The MCP tool works only in the Responses API, and is available across all our new models (gpt-4o, gpt-4.1, and our reasoning models). When you're using the MCP tool, you only pay for tokens used when importing tool definitions or making tool calls—there are no additional fees involved.

Step 1: Getting the list of tools from the MCP server

The first thing the Responses API does when you attach a remote MCP server to the `tools` array, is attempt to get a list of tools from the server. The Responses API supports remote MCP servers that support either the Streamable HTTP or the HTTP/SSE transport protocol.

If successful in retrieving the list of tools, a new `mcp_list_tools` output item will be visible in the Response object that is created for each MCP server. The `tools` property of this object will show the tools that were successfully imported.

```json
{
  "id": "mcpl_682d4379df088191886b70f4ec39f90403937d5f622d7a90",
  "type": "mcp_list_tools",
  "server_label": "deepwiki",
  "tools": [
    {
      "name": "read_wiki_structure",
      "input_schema": {
        "type": "object",
        "properties": {
          "repoName": {
            "type": "string",
            "description": "GitHub repository: owner/repo (e.g. \"facebook/react\")"
          }
        },
        "required": [
          "repoName"
        ],
        "additionalProperties": false,
        "annotations": null,
        "description": "",
        "$schema": "http://json-schema.org/draft-07/schema#"
      }
    },
    // ... other tools
  ]
}
```

As long as the `mcp_list_tools` item is present in the context of the model, we will not attempt to pull a refreshed list of tools from an MCP server. We recommend you keep this item in the model's context as part of every conversation or workflow execution to optimize for latency.

Filtering tools

Some MCP servers can have dozens of tools, and exposing many tools to the model can result in high cost and latency. If you're only interested in a subset of tools an MCP server exposes, you can use the `allowed_tools` parameter to only import those tools.

Constrain allowed tools

python

```python
from openai import OpenAI

client = OpenAI()

resp = client.responses.create(
    model="gpt-4.1",
    tools=[{
        "type": "mcp",
        "server_label": "deepwiki",
        "server_url": "https://mcp.deepwiki.com/mcp",
        "require_approval": "never",
        "allowed_tools": ["ask_question"],
    }],
    input="What transport protocols does the 2025-03-26 version of the MCP spec (mode
)

print(resp.output_text)
```

Step 2: Calling tools

Once the model has access to these tool definitions, it may choose to call them depending on what's in the model's context. When the model decides to call an MCP tool, we make an request to the remote MCP server to call the tool, take it's output and put that into the model's context. This creates an `mcp_call` item which looks like this:

```json
{
  "id": "mcp_682d437d90a88191bf88cd03aae0c3e503937d5f622d7a90",
  "type": "mcp_call",
  "approval_request_id": null,
  "arguments": "{\"repoName\":\"modelcontextprotocol/modelcontextprotocol\",\"questior
  "error": null,
  "name": "ask_question",
  "output": "The 2025-03-26 version of the Model Context Protocol (MCP) specification
  "server_label": "deepwiki"
}
```

As you can see, this includes both the arguments the model decided to use for this tool call, and the `output` that the remote MCP server returned. All models can choose to make multiple (MCP) tool calls in the Responses API, and so, you may see several of these items generated in a single Response API request.

Failed tool calls will populate the error field of this item with MCP protocol errors, MCP tool execution errors, or general connectivity errors. The MCP errors are documented in the MCP spec here.

Approvals

By default, OpenAI will request your approval before any data is shared with a remote MCP server. Approvals help you maintain control and visibility over what data is being sent to an MCP server. We highly recommend that you carefully review (and optionally, log) all data being shared with a remote MCP server. A request for an approval to make an MCP tool call creates a `mcp_approval_request` item in the Response's output that looks like this:

```
{
  "id": "mcpr_682d498e3bd4819196a0ce1664f8e77b04ad1e533afccbfa",
  "type": "mcp_approval_request",
  "arguments": "{\"repoName\":\"modelcontextprotocol/modelcontextprotocol\",\"question\
  "name": "ask_question",
  "server_label": "deepwiki"
}
```

You can then respond to this by creating a new Response object and appending an `mcp_approval_response` item to it.

Approving the use of tools in an API request python

```python
from openai import OpenAI

client = OpenAI()

resp = client.responses.create(
    model="gpt-4.1",
    tools=[{
        "type": "mcp",
        "server_label": "deepwiki",
        "server_url": "https://mcp.deepwiki.com/mcp",
    }],
    previous_response_id="resp_682d498bdefc81918b4a6aa477bfafd904ad1e533afccbfa",
    input=[{
        "type": "mcp_approval_response",
        "approve": True,
        "approval_request_id": "mcpr_682d498e3bd4819196a0ce1664f8e77b04ad1e533afccbfa'
    }],
)

print(resp.output_text)
```

Here we're using the `previous_response_id` parameter to chain this new Response, with the previous Response that generated the approval request. But you can also pass back the outputs from one response, as inputs into another for maximum control over what enter's the model's context.

If and when you feel comfortable trusting a remote MCP server, you can choose to skip the approvals for reduced latency. To do this, you can set the `require_approval` parameter of the MCP tool to an object listing just the tools you'd like to skip approvals for like shown below, or set it to the value `'never'` to skip approvals for all tools in that remote MCP server.

Never require approval for some tools

python

```python
from openai import OpenAI

client = OpenAI()

resp = client.responses.create(
    model="gpt-4.1",
    tools=[
        {
            "type": "mcp",
            "server_label": "deepwiki",
            "server_url": "https://mcp.deepwiki.com/mcp",
            "require_approval": {
                "never": {
                    "tool_names": ["ask_question", "read_wiki_structure"]
                }
            }
        },
    ],
    input="What transport protocols does the 2025-03-26 version of the MCP spec (model
)

print(resp.output_text)
```

Authentication

Unlike the DeepWiki MCP server, most other MCP servers require authentication. The MCP tool in the Responses API gives you the ability to flexibly specify headers that should be included in any request made to a remote MCP server. These headers can be used to share API keys, oAuth access tokens, or any other authentication scheme the remote MCP server implements.

The most common header used by remote MCP servers is the `Authorization` header. This is what passing this header looks like:

Use Stripe MCP tool

python

```python
from openai import OpenAI

client = OpenAI()

resp = client.responses.create(
    model="gpt-4.1",
    input="Create a payment link for $20",
    tools=[
        {
            "type": "mcp",
            "server_label": "stripe",
            "server_url": "https://mcp.stripe.com",
            "headers": {
                "Authorization": "Bearer $STRIPE_API_KEY"
            }
        }
    ]
)

print(resp.output_text)
```

To prevent the leakage of sensitive keys, the Responses API does not store the values of **any** string you provide in the `headers` object. These values will also not be visible in the Response object created. Additionally, because some remote MCP servers generate authenticated URLs, we also discard the *path* portion of the `server_url` in our responses (i.e. `example.com/mcp` becomes `example.com`). Because of this, you must send the full path of the MCP `server_url` and any relevant `headers` in every Responses API creation request you make.

Risks and safety

The MCP tool permits you to connect OpenAI to services that have not been verified by OpenAI and allows OpenAI to access, send and receive data, and take action in these services. All MCP servers are third-party services that are subject to their own terms and conditions.

If you come across a malicious MCP server, please report it to `security@openai.com`.

Connecting to trusted servers

Pick official servers hosted by the service providers themselves (e.g. we recommend connecting to the Stripe server hosted by Stripe themselves on mcp.stripe.com, instead of a Stripe MCP server hosted by a third party). Because there aren't too many official remote MCP servers today, you may be tempted to use a MCP server hosted by an organization that doesn't operate that server and simply proxies request to that service via your API. If you must do this, be extra careful in doing your due diligence on these "aggregators", and carefully review how they use your data.

Log and review data being shared with third party MCP servers.

Because MCP servers define their own tool definitions, they may request for data that you may not always be comfortable sharing with the host of that MCP server. Because of this, the MCP tool in the Responses API defaults to requiring approvals of each MCP tool call being made. When developing your application, review the type of data being shared with these MCP servers carefully and robustly. Once you gain confidence in your trust of this MCP server, you can skip these approvals for more performant execution.

We also recommend logging any data sent to MCP servers. If you're using the Responses API with `store=true`, these data are already logged via the API for 30 days unless Zero Data Retention is enabled for your organization. You may also want to log these data in your own systems and perform periodic reviews on this to ensure data is being shared per your expectations.

Malicious MCP servers may include hidden instructions (prompt injections) designed to make OpenAI models behave unexpectedly. While OpenAI has implemented built-in safeguards to help detect and block these threats, it's essential to carefully review inputs and outputs, and ensure connections are established only with trusted servers.

MCP servers may update tool behavior unexpectedly, potentially leading to unintended or malicious behavior.

Implications on Zero Data Retention and Data Residency

The MCP tool is compatible with Zero Data Retention and Data Residency, but it's important to note that MCP servers are third-party services, and data sent to an MCP server is subject to their data retention and data residency policies.

In other words, if you're an organization with Data Residency in Europe, OpenAI will limit inference and storage of Customer Content to take place in Europe up until the point communication or data is sent to the MCP server. It is your responsibility to ensure that the MCP server also adheres to any

Zero Data Retention or Data Residency requirements you may have. Learn more about Zero Data Retention and Data Residency here.

Usage notes

API AVAILABILITY	RATE LIMITS	NOTES
✓ Responses ✗ Chat Completions ✗ Assistants	**Tier 1** 200 RPM **Tier 2 and 3** 1000 RPM **Tier 4 and 5** 2000 RPM	Pricing ZDR and data residency