For logs

```
COMMAND   PID        USER   FD   TYPE             DEVICE SIZE/OFF NODE NAME
node    94289 manojpadala   14u  IPv4 0x977070b38b1dc48e      0t0  TCP localhost:6274 (L
ISTEN)
(mcp_venv) (base) manojpadala@MJs-MacBook-Air tavily % kill -9 94289
(mcp_venv) (base) manojpadala@MJs-MacBook-Air tavily % mcp dev server.py
Starting MCP inspector...
⚙️ Proxy server listening on localhost:6277
🔑 Session token: a5c94c7c8e09fc1e57b4ad65c89b4b20e4ab01edd291fae7a8f344cbb1696635
   Use this token to authenticate requests or set DANGEROUSLY_OMIT_AUTH=true to disable
auth

🚀 MCP Inspector is up and running at:
   http://localhost:6274/?MCP_PROXY_AUTH_TOKEN=a5c94c7c8e09fc1e57b4ad65c89b4b20e4ab01edd
291fae7a8f344cbb1696635

🌐 Opening browser...
New STDIO connection request
Query parameters: {"command":"uv","args":"run --with mcp mcp run server.py","env":"{\"HO
ME\":\"/Users/manojpadala\",\"LOGNAME\":\"manojpadala\",\"PATH\":\"/Users/manojpadala/.n
pm/_npx/5a9d879542beca3a/node_modules/.bin:/Users/manojpadala/Downloads/Klavis_Ai/tavily
/node_modules/.bin:/Users/manojpadala/Downloads/Klavis_Ai/node_modules/.bin:/Users/manoj
padala/Downloads/node_modules/.bin:/Users/manojpadala/node_modules/.bin:/Users/node_modu
les/.bin:/node_modules/.bin:/opt/homebrew/lib/node_modules/npm/node_modules/@npmcli/run-
script/lib/node-gyp-bin:/Users/manojpadala/Downloads/Klavis_Ai/tavily/mcp_venv/bin:/Appl
```

This screenshot shows the process of starting the Tavily MCP server in development mode using the command:

```
mcp dev server.py
```

What's happening here:

1. The existing MCP process is terminated (`kill -9 94289`).
2. The MCP server (`server.py`) is started in development mode using the MCP CLI.
3. The MCP Inspector launches automatically:
   - A **proxy server** is set up on `localhost:6274`.
   - A **session token** is generated for authentication.
   - The **MCP Inspector UI** becomes accessible at a local URL (shown in the log).
4. The MCP CLI opens a browser tab with the MCP Inspector interface.

How the tool should be used:

- **During development**, run `mcp dev server.py` to launch your MCP server with live inspection.
- The **MCP Inspector** lets you:
    - View available tools registered by the server.
    - Test tool calls with sample input.
    - Inspect responses and logs in real-time.
- Use the generated session token if you need to authenticate external requests to your MCP server.
- The MCP Inspector's local UI is the quickest way to debug and confirm that tools (like `tavily.search` or `tavily.crawl`) are correctly loaded and responding.

```
m=256color\",\"USER\":\"manojpadala\"},"transportType":"stdio"}
STDIO transport: command=/opt/homebrew/bin/uv, args=run,--with,mcp,mcp,run,server.py
Created server transport
Created client transport
Received POST message for sessionId 30ba5ccd-3452-4c92-89d4-004ddff8782d
Received POST message for sessionId 30ba5ccd-3452-4c92-89d4-004ddff8782d
Received POST message for sessionId 30ba5ccd-3452-4c92-89d4-004ddff8782d
Received POST message for sessionId 30ba5ccd-3452-4c92-89d4-004ddff8782d
Received POST message for sessionId 30ba5ccd-3452-4c92-89d4-004ddff8782d
Received POST message for sessionId 30ba5ccd-3452-4c92-89d4-004ddff8782d
Received POST message for sessionId 30ba5ccd-3452-4c92-89d4-004ddff8782d
```

What you're seeing:

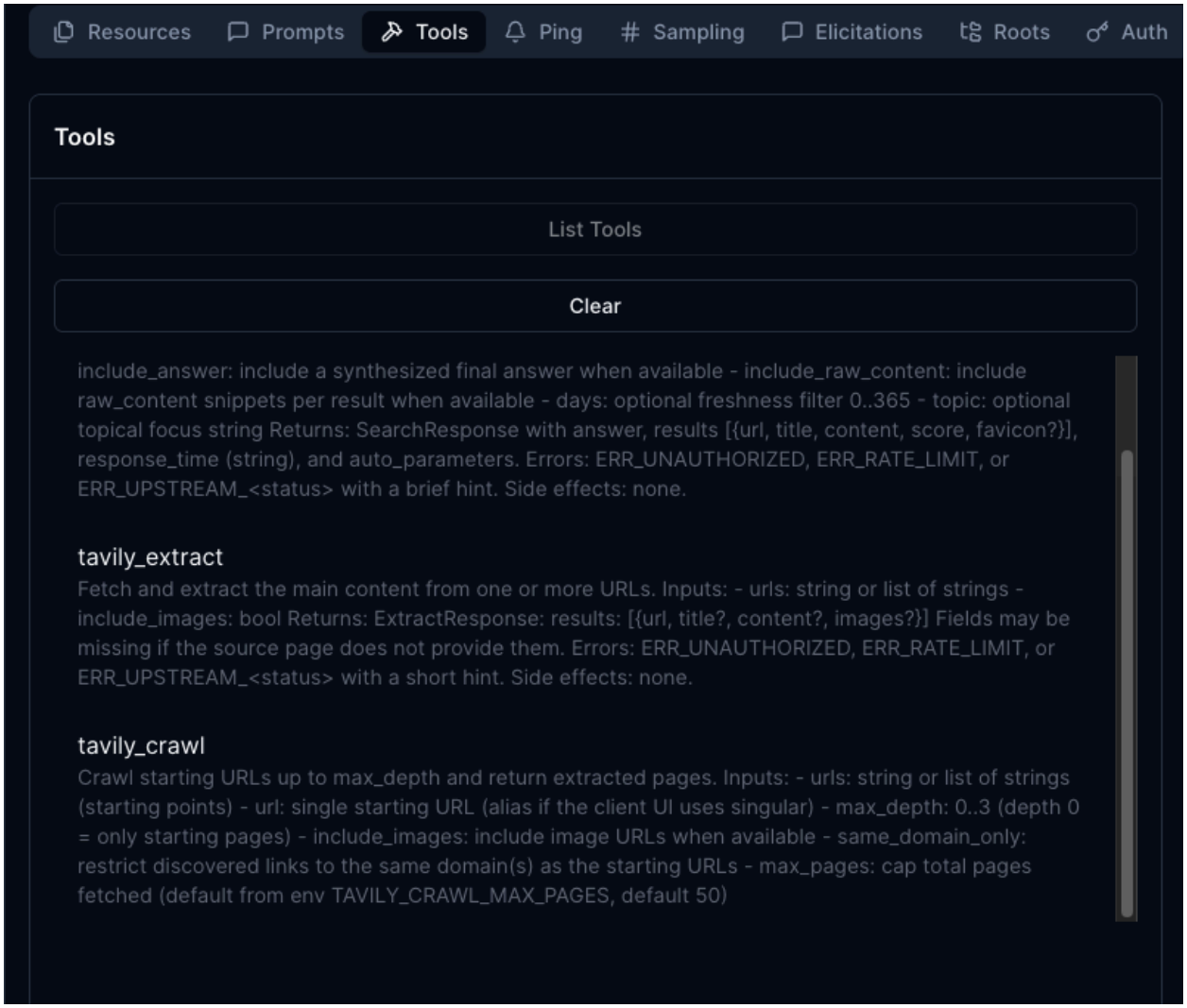
- The MCP CLI has launched your server via **STDIO transport** (`mcp run server.py` under `uv`).
- **"Created server transport / Created client transport"** → handshake is complete; the Inspector/client is connected to the server.
- The repeated lines **"Received POST message for sessionId 30ba5ccd-…"** show HTTP-style RPCs flowing through the proxy to your server. These are requests like `list_tools`, `call_tool`, health checks, or UI polling — all tied to the same session.

How to use this in practice:

- In the MCP Inspector, pick a tool (e.g., `tavily_search`) and click **Run**.
  You should see one or more **POST** lines here, followed by your tool's own logs (e.g., `tavily_search ok … results=5`).

- If you only see POST lines but **no tool logs or results**, the request likely failed validation or the tool raised an error — check the Inspector response for a normalized `ERR_*` message.

- Multiple POSTs in a burst can be normal (the Inspector may send a sequence: list tools → validate → run tool → fetch result).



tavily_search

Execute a Tavily web search and return structured results. Inputs: - query: non-empty search text - search_depth: 'basic' for faster shallow search, 'advanced' for deeper search - max_results: 1..10 - include_answer: include a synthesized final answer when available - include_raw_content: include raw_content snippets per result when available - days: optional freshness filter 0..365 - topic: optional topical focus string Returns: SearchResponse with answer, results [{url, title, content, score, favicon?}], response_time (string), and auto_parameters. Errors: ERR_UNAUTHORIZED, ERR_RATE_LIMIT, or ERR_UPSTREAM_<status> with a brief hint. Side effects: none.

query *

tell me about model context protocol

search_depth

basic

max_results

5

include_answer

☑ Toggle this option

include_raw_content

☐ Toggle this option

days

Copy JSON | Format JSON

days

Copy JSON | Format JSON

topic

Copy JSON | Format JSON

```
{
  query: "tell me about model context protocol"
  answer: "The Model Context Protocol is an open standard for connecting AI sys
          tems to data sources, enabling secure two-way communication. It was i
          ntroduced by Anthropic to standardize AI tool integration. It helps A
          I agents operate autonomously and streamline complex workflows."
  results: [
    0: {
      url: "https://en.wikipedia.org/wiki/Model_Context_Protocol"
      title: "Model Context Protocol - Wikipedia"
      raw_content: null
      score: 0.90330297
      favicon: null
      content: "The **Model Context Protocol** (**MCP**) is an open standard,
               open-sourceframework introduced by Ant..."
    }
    1: {
      url: "https://www.anthropic.com/news/model-context-protocol"
      title: "Introducing the Model Context Protocol - Anthropic"
      raw_content: null
      score: 0.86981434
      favicon: null
      content: "Today, we're open-sourcing the Model Context Protocol (MCP),
               a new standard for connecting AI assist..."
    }
    2: {
      url: "https://www.forbes.com/sites/davidbirch/2025/04/26/why-you-need-t
           o-know-about-the-model-context-prot..."
      title: "Why You Need To Know About The Model Context Protocol - Forbes"
      raw_content: null
      score: 0.8622012
      favicon: null
      content: "The Model Context Protocol (MCP) is an open standard that ena
               bles developers to build secure, two-wa..."
    }
    3: {
      url: "https://www.ibm.com/think/topics/model-context-protocol"
```

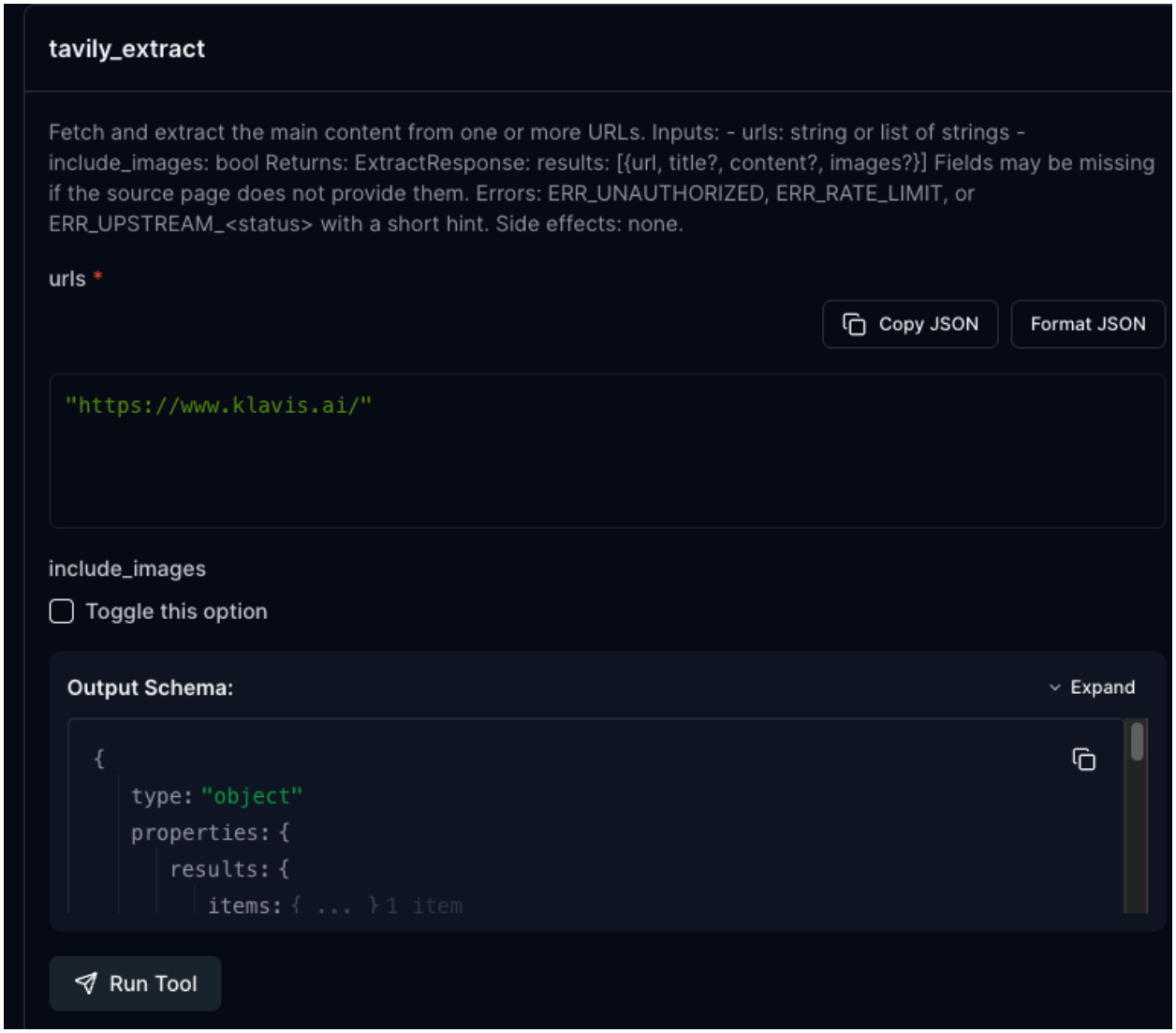
Use `tavily.search` in MCP Inspector (crisp)

1. Open the **Tools** tab → pick **`tavily_search`** (alias of `tavily.search`).
2. Fill the form:
    - **query**: e.g., `tell me about model context protocol`
    - **search_depth**: `basic` (faster) or `advanced` (deeper)
    - **max_results**: `5` is a good default
    - **include_answer**: ✅ on (get a synthesized summary when available)
    - **include_raw_content**: off unless you want longer snippets
    - **days** (optional): freshness filter
    - **topic** (optional): nudge toward a vertical (e.g., “security”, “research”)
3. Click **Run**.

Read the result

- Top-level fields:
    - **query**: echoes what you asked
    - **answer**: a short synthesized summary (present when sources allow)
    - **results[]**: list of sources (see below)
    - **response_time**: how long it took (string)
- Each **results[i]** item:
    - **url**, **title**

- **content** (short cleaned snippet; `raw_content` may be null if not requested)
- **score** (relevance), **favicon** (if available)



Tool Result: Success

Structured Content:


```
{
  results: [
    0: {
      url: "https://www.klavis.ai/"
      title: null
      content: "![KlavisAI](/_next/image?url=%2Fimages%2Ffavicon%2Ffavicon.ic
               o&w=48&q=75)
               ![KlavisAI](/_next/image?url=%2Fimages%2Ffavicon%2Ffavicon.ico
               &w=48&q=75)
               ![KlavisAI](/_next/image?url=%2Fimages%2Ffavicon%2Ffavicon.ico
               &w=48&q=75)

               [X (Twitter)](https://x.com/Klavis_AI)[LinkedIn](https://www.l
               inkedin.com/company/klavis-ai/)[GitHub](https://github.com/Kla
               vis-AI/klavis)[Discord](https://discord.gg/p7TuTEcssn)[YouTub
               e](https://www.youtube.com/@KlavisAI)

               © Copyright 2025 KlavisAI. All Rights Reserved.

               ![Discord](/images/companies/Discord_logo.svg)

               # MCP Integrationsfor AI Applications

               ### Easiest way to connect to high-quality, secure MCP Servers
               at scale.

               Easiest way to connect to high-quality, secure MCP Servers at
               scale.

               ## Integrate in a minute, Scale to millions

               `from klavis import Klavis
               klavis_client = Klavis(api_key="KLAVIS_API_KEY")
               mcp_server = klavis_client.mcp_server.create_server_instance(
                server_name=<MCP_SERVER_NAME>,
                user_id=<USER_ID>,
                platform_name=<PLATFORM_NAME>
```

```
## MCP Evaluation & Analytics

### Accurate Tool Design

Advanced tool design with internal Gemini-level evaluation sys
tem for superior agent performance

### Real-Time Observability

Comprehensive logs, metrics, and alerts purpose-built for seam
less integrations

### Evolving Flywheel

Continuously learning system that adapts to make your agents i
ncreasingly reliable

![Enterprise security dashboard showing encryption, access con
trol, and compliance features](/_next/image?url=%2Fimages%2Fla
ngding_page%2Fsecurity.png&w=3840&q=75)

## Enterprise-Grade Security

### Klavis Guardrails Architecture

Multi-layered security system detecting tool poisoning, prompt
injection, privilege escalation, and command injection in MCP
interactions

### Granular Scopes & Permissions

Precise control over scopes and permissions requested from ext
ernal APIs for enhanced security

### SOC 2 Compliance

Dedicated to maintaining the highest security standards for da
ta protection and privacy

## What our customers say

Discover what our community has to say about their Klavis expe
rience.
```

Using `tavily_extract`

The `tavily_extract` tool is used to fetch and extract the main textual content from one or more specified URLs.

Steps to Use:

1. In the **Tools** tab, select **`tavily_extract`**.

2. Enter the URL or a list of URLs in the `urls` field.
   Example:
   `"https://www.klavis.ai/"`
   or
   `["https://site1.com", "https://site2.com"]`

3. (Optional) Enable **include_images** if you want image URLs to be returned.

4. Click **Run Tool**.

Output:

- Returns a JSON object with:
  - **url** – the source URL.
  - **title** – page title (if available).
  - **content** – extracted main content in plain text or markdown.
  - **images** – list of image URLs (only if `include_images` is enabled).

Tavily_crawl

tavily_crawl

Crawl starting URLs up to max_depth and return extracted pages. Inputs: - urls: string or list of strings (starting points) - url: single starting URL (alias if the client UI uses singular) - max_depth: 0..3 (depth 0 = only starting pages) - include_images: include image URLs when available - same_domain_only: restrict discovered links to the same domain(s) as the starting URLs - max_pages: cap total pages fetched (default from env TAVILY_CRAWL_MAX_PAGES, default 50)

urls

Copy JSON | Format JSON

url

Copy JSON | Format JSON

```
"https://www.klavis.ai/"
```

max_depth

5

include_images

☐ Toggle this option

same_domain_only

☑ Toggle this option

max_pages

5

Structured Content:


```
{
  pages: [
    0: {
      url: "https://www.klavis.ai/"
      title: null
      content: "Open Source MCP Integrations for AI Applications | Klavis AI

               - [Home](/)
               - [MCP Servers](/mcp-serve..."
      images: null
      depth: null
      parent: null
      favicon: null
    }
    1: {
      url: "https://www.klavis.ai/faq"
      title: null
      content: "![KlavisAI](/_next/image?url=%2Fimages%2Ffavicon%2Ffavicon.ic
               o&w=48&q=75)
               ![KlavisAI](/_next/image?u..."
      images: null
      depth: null
      parent: null
      favicon: null
    }
    2: {
      url: "https://www.klavis.ai/privacy-policy"
      title: null
      content: "![KlavisAI](/_next/image?url=%2Fimages%2Ffavicon%2Ffavicon.ic
               o&w=48&q=75)
               ![KlavisAI](/_next/image?u..."
      images: null
      depth: null
      parent: null
      favicon: null
    }
```


Using `tavily_crawl`

Goal: discover and extract multiple pages starting from one URL.

Steps

1. In **Tools**, choose **`tavily_crawl`** (alias of `tavily.crawl`).
2. Fill the fields:
    - **url**: `"https://www.klavis.ai/"`
    - **max_depth**: `5` (how many link-levels to follow)
    - **same_domain_only**: enable to stay on `klavis.ai` (optional)
    - **max_pages**: `5` (cap total pages)
    - **include_images**: toggle if you want image URLs (optional)
3. Click **Run Tool**.

What you get

- A JSON object with **`pages`** — one entry per discovered page:
    - **url** and **title** (if available)
    - **content** (clean text/markdown from the page)
    - **images** (if requested)
    - Optional **depth**, **parent**, **favicon** (may be `null` depending on source)