

 AGNTCON + MCPCON NA 2026

THE FRONTEND STRIKES BACK

WebMCP and the agent-ready browser



INTRODUCED BY

Andrew Nolan

Co-creator of WebMCP



SPEAKER

Ryan Roemer

SVP, Nearform

🔗 OUR QUESTION

Where does the browser fit in the AI ecosystem?

As an agent interface and runtime.

Why do we care?



If you build frontends

Agents are increasingly your web app users.



If you build AI systems

You can offload AI work to your user's browser

01 · INTERFACE

WebMCP

01

What is WebMCP?

① MCP tools, served by your web page

A proposed W3C standard.

② Two APIs

Imperative JavaScript now, declarative HTML forms in the future.

③ In browsers now

Origin trial in Chrome and Edge, plus a growing list of agent clients. We'll try two shortly!

Why WebMCP?

① Agents already use your site

In a browser, just poorly: screen-scraping and clicking through UIs built for humans.

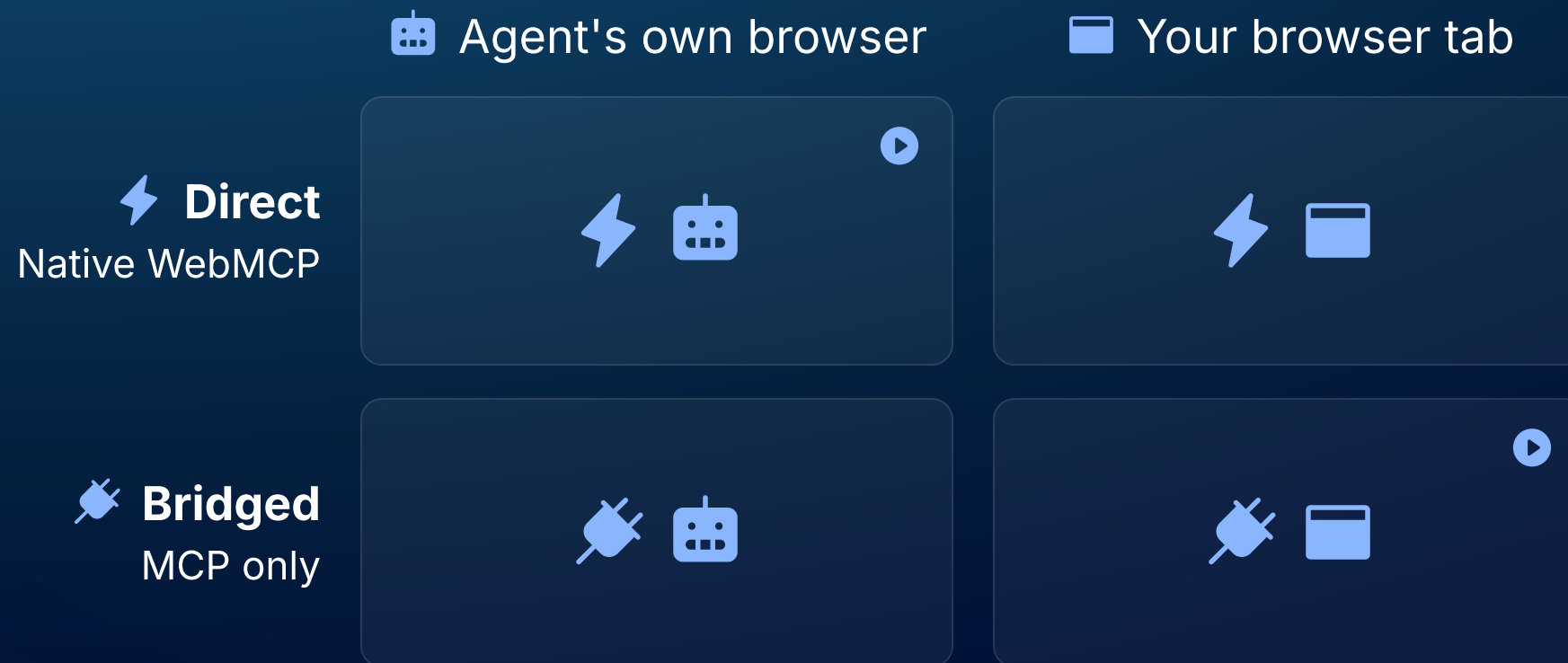
② Your users are already logged in

Tools run in their tab, with the **session** they already have. Browser takes care of the hard parts: **sandboxing**, **isolation**, **permissions**, etc.

③ A good bet to start on now

Tools are the functions your UI already calls. No new server.

Who calls, and from where?

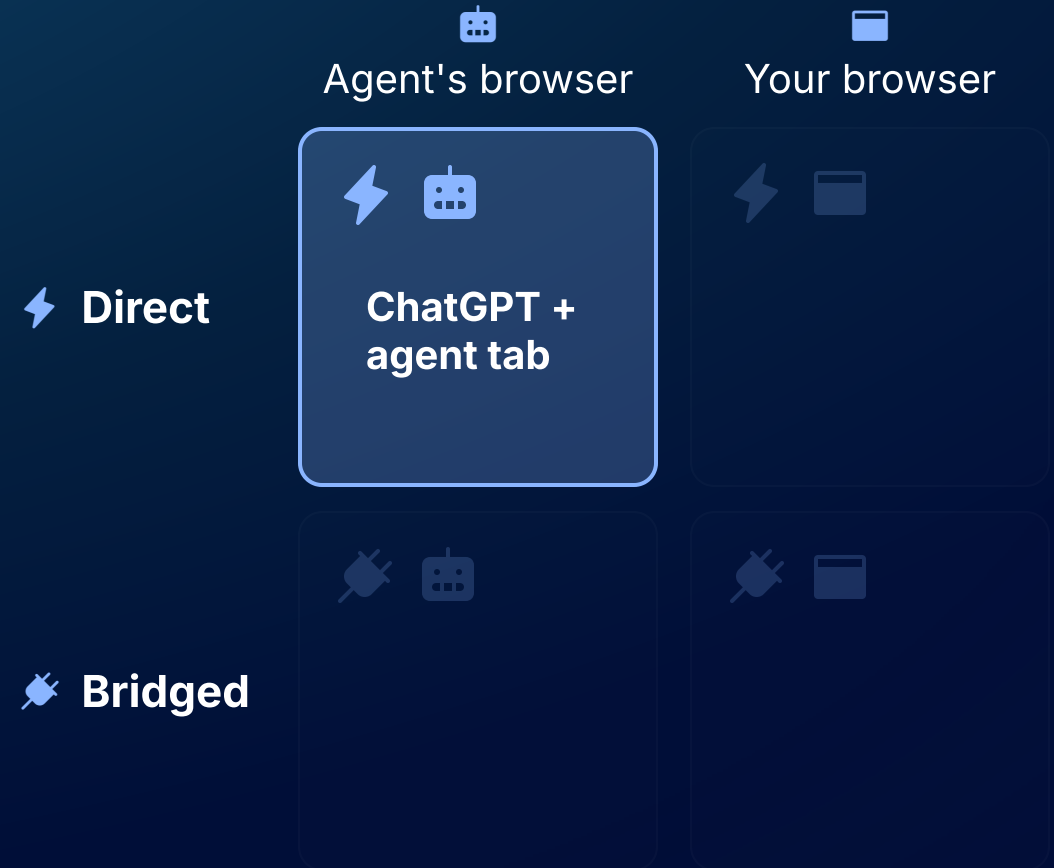


 In a demo today

ChatGPT Desktop shops Reebok

reebok.com 

- ▶ **ChatGPT desktop**'s built-in browser calls WebMCP tools in pages
- ▶ **Shopify** ships WebMCP tools on every Liquid storefront.



The WebMCP API

• model-context.js



```
1 const { modelContext } = document;
2
3 // The page: register tools.
4 await modelContext.registerTool(/* ... */);
5
6 // An agent: find the tools, and call one.
7 const [tool] = await modelContext.getTools();
8 console.log(await modelContext.executeTool(tool, { query: "WebMCP" }));
9
10 // Either side: hear when the list changes.
11 modelContext.addEventListener("toolchange" /*, ... */);
```

One tool, up close

• register-tool.js



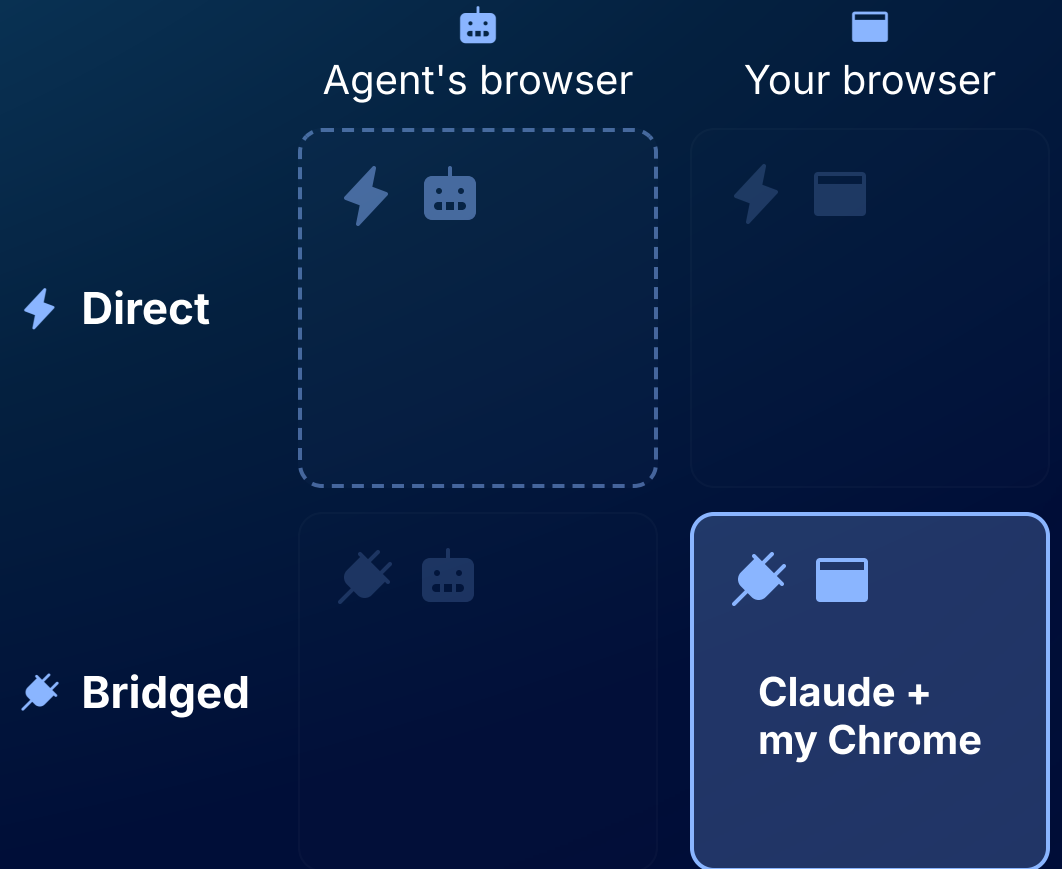
```
1 document.modelContext.registerTool({
2   name: "search_documents",
3   description:
4     "Search the company knowledge base. Use for any question about our docs or expertise.",
5   inputSchema: {
6     type: "object",
7     properties: {
8       query: { type: "string", description: "Natural language query." },
9     },
10    required: ["query"],
11  },
12  // Use any existing app code, context, auth, prefs. Can update UI too.
13  execute: ({ query }) => window.app.searchDocuments(query),
14 });
```

▶ DEMO · ANY MCP CLIENT + REAL USER BROWSER

Claude Desktop calls our page

nearform.github.io/vector-search-web 

- ▶ [@mcp-b/webmcp-local-relay](#)
- ▶ [My_page](#): WebMCP tools + a **relay embed**.
- ▶ [Claude](#): MCP connection to **relay server**.



Tips for WebMCP

① Consider the UI when a tool runs

Tool actions can optionally update or ignore the UI.

② Read up on the safety features

Hint the risk: read-only, untrusted, consequential. Share tools only with origins you trust.

③ Keep up: this space moves wildly fast

See Chrome's [best practices](#). Join the [early preview program](#) and watch the [spec](#)!

02 · RUNTIME

AI in the browser

Why web AI?

Privacy: data never leaves the device.

Offline: works with no network, after the download.

Latency: no round trip to a server.

Cost: the user's GPU, not your inference bill.

It's the future: WebGPU and smaller, better models

The available pieces



Embeddings



Vector database



Classifiers



Deciders



Tools



**Language
models**



**Fine-tuned
models**



Here now



Emerging



In a demo today



Similarity search in a knowledge store

nearform.github.io/vector-search-web 

- ▶ Precomputed **embeddings** (2-5 MB) and **content** (3 MB), one download.
- ▶ Data download in 200-800 ms, then ~25 ms to embed and ~5 ms to search.
- ▶ **Vector search here.** Lexical or hybrid is a config change. ([Orama](#))

The simplest way to try web AI inference

● prompt-api.js 

```
1 // Chrome 148+ (desktop): on by default.  
2 // Edge Canary/Dev 138+: edge://flags → "Prompt API for on-device language model".  
3 const session = await LanguageModel.create();  
4  
5 console.log(await session.prompt("In one sentence, what is bikeshedding?"));  
6  
7 session.destroy();
```

// console output 

Five ways to run web AI inference

RUNTIME	TOOLS	IPHONE 15 PRO	CONTEXT
 Prompt API	 Behind a flag	 Unavailable	~9K 
 WebLLM	 Only select models	 Runs	1-4K  16K
 wllama	 You run it	 Runs	128-256K  32K
 Transformers.js	 Parse it yourself	 Answers, then dies	~32K  4K
 LiteRT-LM.js	 Runs them for you	 Tab dies on load	128-256K  32K

[Read the details](#) 

[Try the demo](#) 

Tips for web AI inference

① Model size

Smallest model that works: faster to download and run.

② Context size

More context costs memory, speed, and often quality.

③ Experiment a lot

Different runtimes, models, and configurations.

④ Don't forget evals

Run app AI workflows in a real browser over Playwright or CDP. Test comprehensively.

03 · WEB AGENTS

Agents in the browser



A team of agents tackles a research task

nearform.github.io/web-agents 

- ▶ A coordinator, a researcher, and a writer.
- ▶ Searching the vector search app over WebMCP.
- ▶ Taking and reading notes as they go.

Tips for web agents

① Tools

Give your in-tab model the same WebMCP tools outside agents use.

② Trade time for context

Small, focused calls. Each agent summarizes before handing off.

③ Specific models

A small model per task, fine-tuned, or post-trained for the behavior you need.

So, what now?



If you build frontends

Agents are increasingly your web app users.

Register a tool on an existing app and consume it.



If you build AI systems

You can offload AI work to your user's browser

Move one AI piece into the tab: a classifier, decider, vector search, or a small model.

Thanks!

 [ryan-roemer](https://github.com/ryan-roemer)

 [in/ryanroemer](https://www.linkedin.com/in/ryanroemer)

 [@ryan-roemer.bsky.social](https://bsky.app/profile/ryan-roemer.bsky.social)

 ryan.roemer@nearform.com

SLIDES & SOURCE

 ryan-roemer.github.io/the-frontend-strikes-back

 github.com/ryan-roemer/the-frontend-strikes-back