ALPIC

Skills vs. MCP vs. Code Mode: cutting through the hype (and the rage)

Nikolay Rodionov, Cofounder @Alpic

MCP Dev Summit - April 2

Eleanor Berger
@intellectronica

Afficher la traduction

I am no longer using _any_ MCP servers in my local setup [@code / @GitHubCopilot, @opencode, @claudeai code].

- Context7 → SKILL + curl
- Tavily → SKILL + curl
- Playwright → SKILL + agent-browser

SKILLs are all you need!

3:53 PM · 20 janv. 2026 · **113 k** vues

Aiden Bai
@aidenybai

Souscrire

Afficher la traduction

im probably being an midwit but here goes

why do we need MCP if skills exist now?

7:07 PM · 20 janv. 2026 · **133,2 k** vues

147 16 411 183

Pertinence

Voir les citations

@levelsio
@levelsio

Souscrire

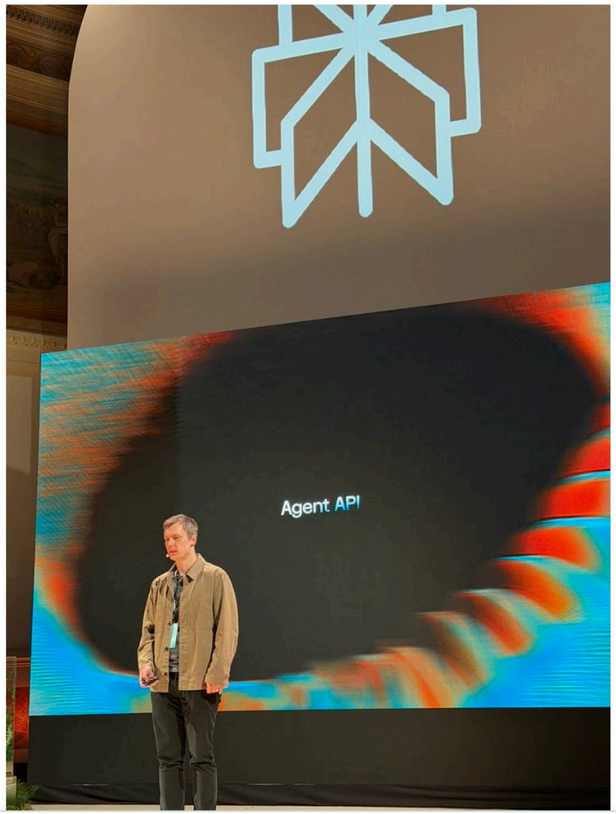
Afficher la traduction

Thank god MCP is dead

Just as useless of an idea as LLMs.txt was

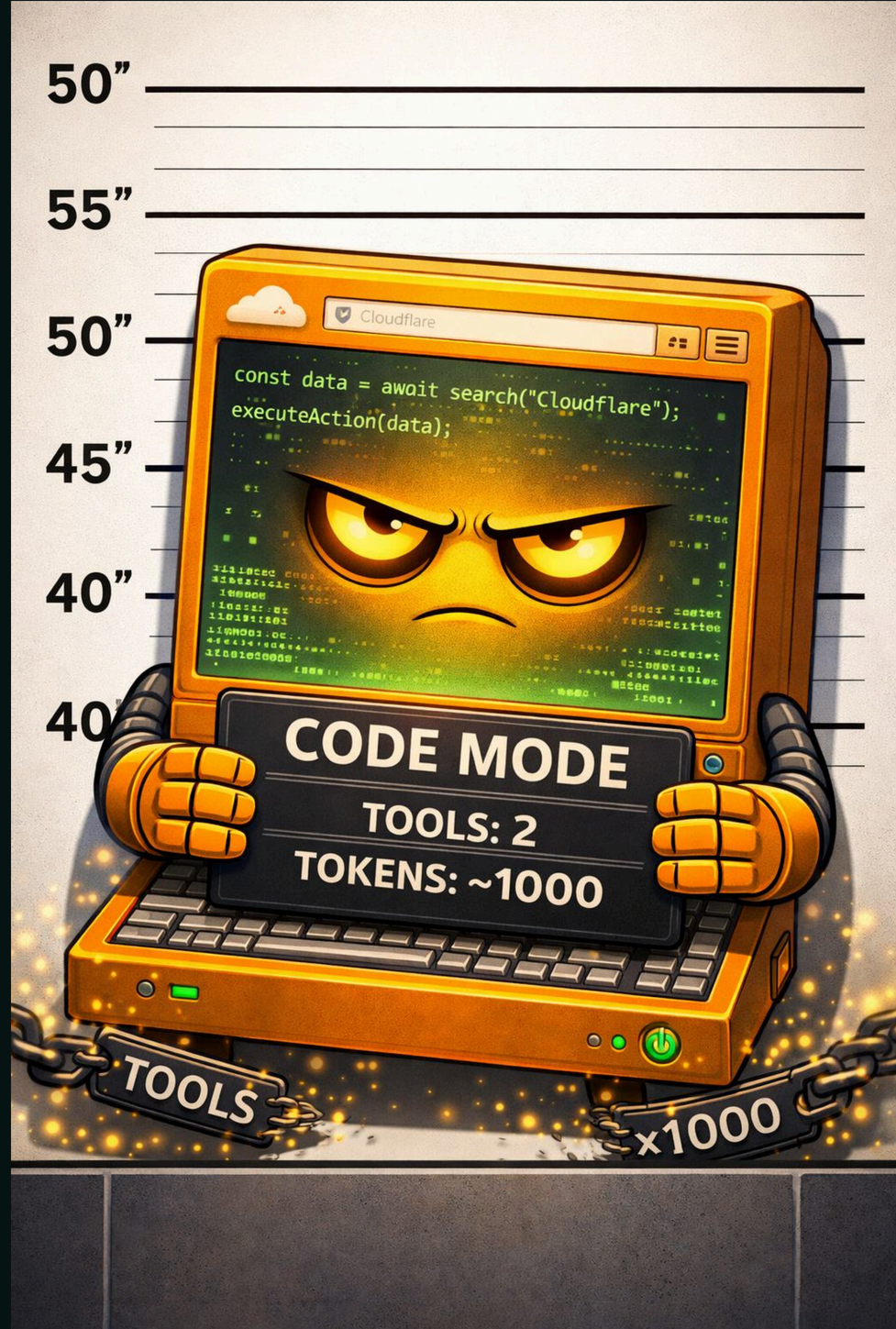
It's all dumb abstractions that AI doesn't need because AI's are as smart as humans so they can just use what was already there which is APIs

> **Morgan** @morganlinton · 11 mars
> The cofounder and CTO of Perplexity, @denisyarats just said internally at Perplexity they're moving away from MCPs and instead using APIs and CLIs 👀



Is MCP Dead ?

Suspect #1:
Code mode



Problem: MCP uses too much context!

MCP Clients load:

- server name & description
- tool names and description
- tool schemas

Average tool footprint:
~500 tokens

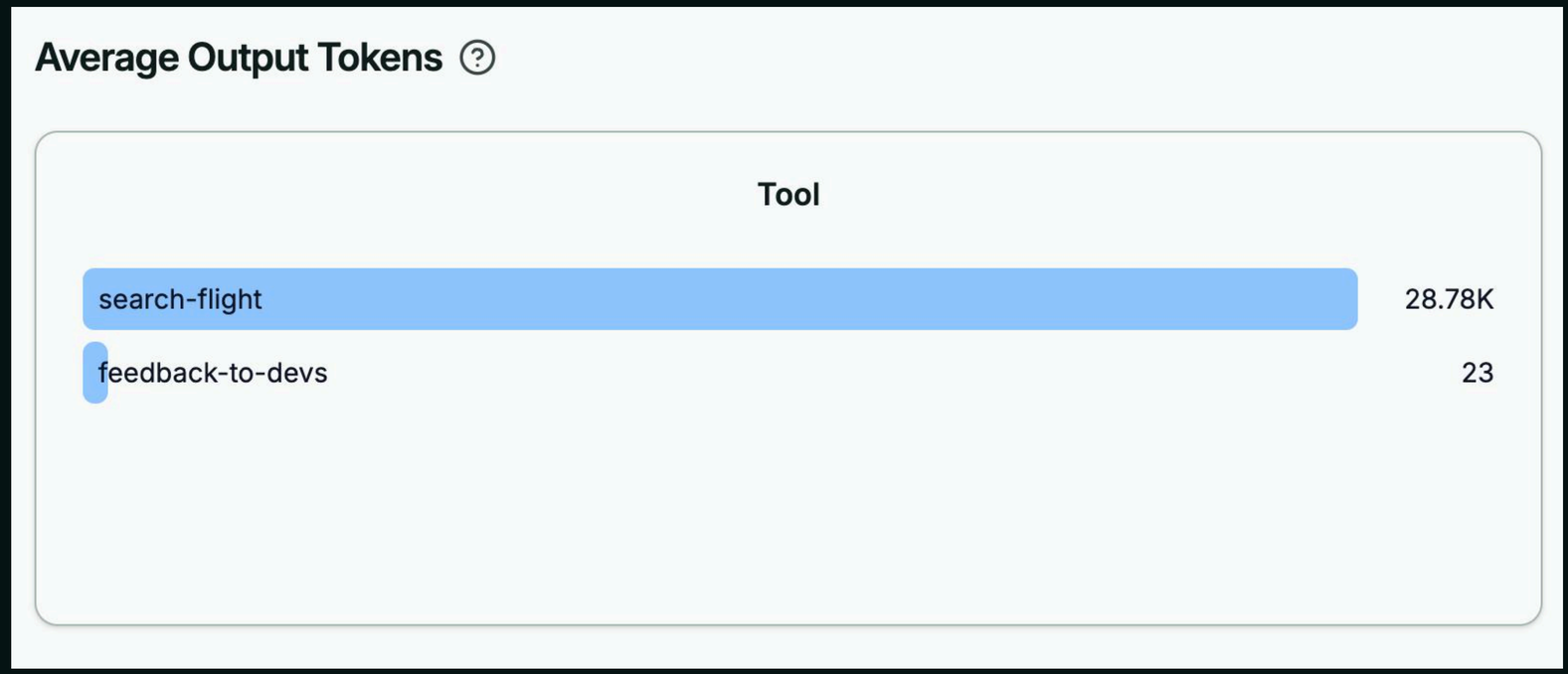
Github MCP Server:
~100 tools, 60K tokens

```
server.registerTool(
  "create_ticket",
  {
    description:
      "Example tool definition for creating a support ticket from structured input.",
    inputSchema: {
      title: z.string().min(1).describe("Ticket title."),
      priority: z
        .enum(["low", "medium", "high", "critical"])
        .describe("Ticket priority.")
        .optional(),
      labels: z.array(z.string()).describe("Labels to attach to the ticket.").optional(),
      reporter: z
        .object({
          id: z.string().optional(),
          name: z.string().optional(),
          contact: z
            .object({
              email: z.string().optional(),
              phone: z.string().optional(),
            })
            .optional(),
        })
        .optional(),
      due_date: z.string().describe("Due date string.").optional(),
      escalation: z
        .object({
          level: z.number().int().optional(),
          notify_manager: z.boolean().optional(),
          sla_hours: z.number().int().optional(),
        })
        .optional(),
    },
  },
```

Problem: MCP uses too much context!

***All* tool result content**

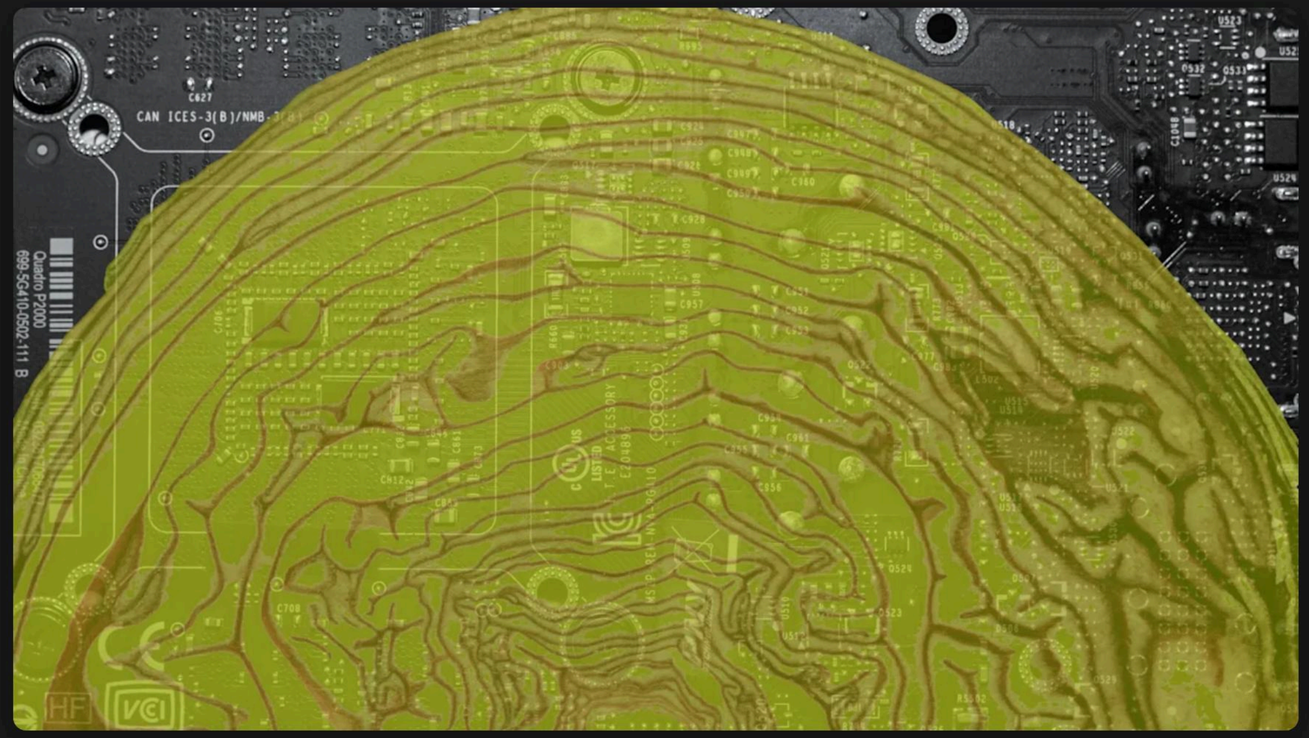
goes into the model's
context window!



Solutions:

- ~~Client dynamic tool discovery~~
- Less but smarter tools
- Onion pattern:
  - search_tool
  - get_documentation
  - execute_tool
- Code mode?

$cd .. ←



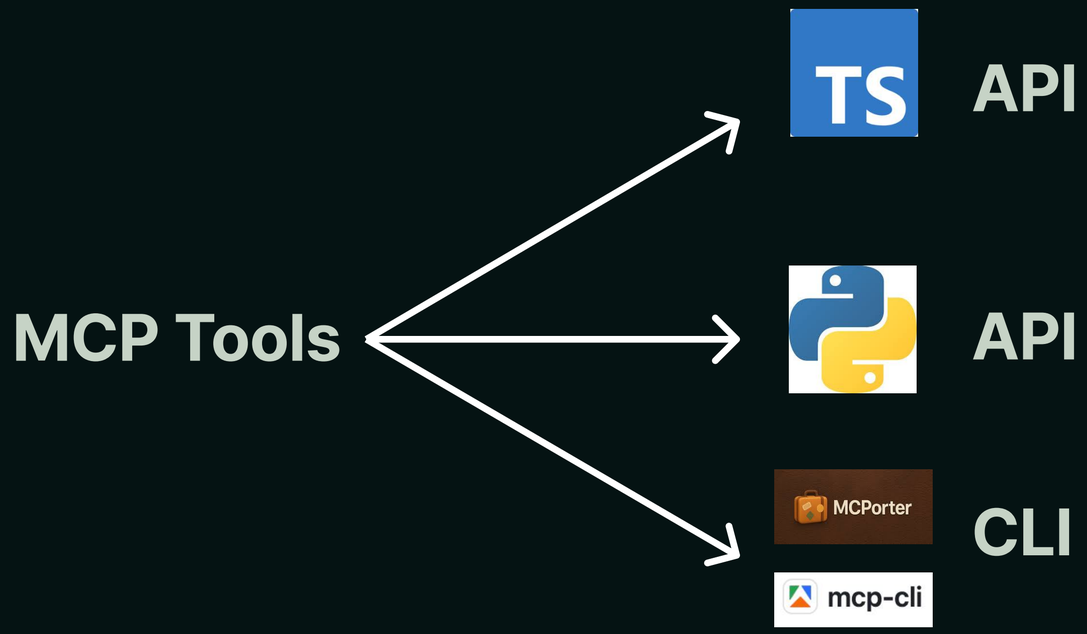
Engineering May 9, 2025

Build MCP Tools Like Ogres... With Layers

An Old Pattern Made New for Building MCP Server Tools

Code Mode:

Step 1:



Code Mode:

Step 2:
search docs
& generate code

Server-side

```
[
  {
    "name": "search",
    "description": "Search the Cloudflare OpenAPI spec. All $refs are pre-resolved inline.",
    "inputSchema": {
      "type": "object",
      "properties": {
        "code": {
          "type": "string",
          "description": "JavaScript async arrow function to search the OpenAPI spec"
        }
      },
      "required": ["code"]
    }
  },
```

Client-side

```
servers
├── google-drive
│   ├── getDocument.ts
│   ├── ... (other tools)
│   └── index.ts
├── salesforce
│   ├── updateRecord.ts
│   ├── ... (other tools)
│   └── index.ts
└── ... (other servers)
```

Copy

Code Mode:

Step 3:

execute code!

Server-side

Client-side

Code mode: summary

Pros:

- Fixed number of tools
- Allows complex workflows
- solves context bloat by intermediary tool responses

Cons:

- ~2k token minimal footprint
- needs code sandbox access
- no interactivity (UI, elicitations)
- can decrease Task completion rate

Code mode: summary

Great if you have >20 tools and complex workflows

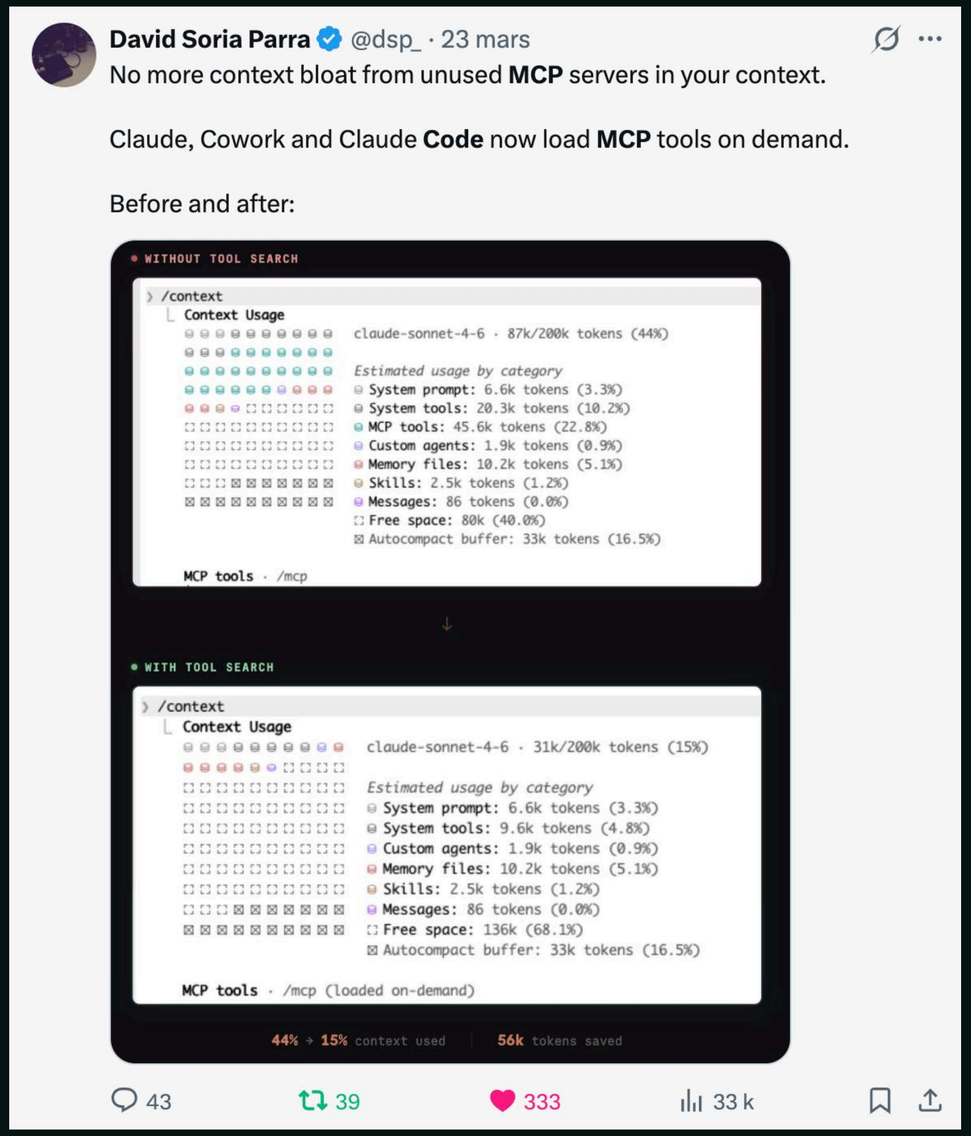
Not really needed otherwise

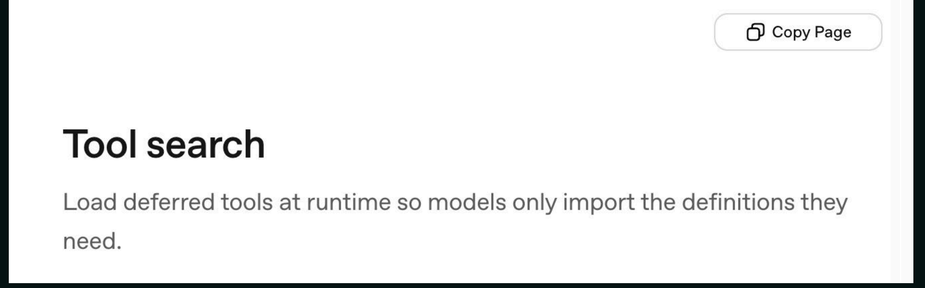
Code mode: did he do it?

NO!

Context bloat was never an MCP problem,
it was a MCP Client implementation issue!

What about... Client Tool search?





Suspect #2:
Skills

Skills are... markdown files

name	frontend-design
description	Create distinctive, production-grade frontend interfaces with high design quality. Use this skill when the user asks to build web components, pages, artifacts, posters, or applications (examples include websites, landing pages, dashboards, React components, HTML/CSS layouts, or when styling/beautifying any web UI). Generates creative, polished code and UI design that avoids generic AI aesthetics.
license	Complete terms in LICENSE.txt

This skill guides creation of distinctive, production-grade frontend interfaces that avoid generic "AI slop" aesthetics. Implement real working code with exceptional attention to aesthetic details and creative choices.

The user provides frontend requirements: a component, page, application, or interface to build. They may include context about the purpose, audience, or technical constraints.

Design Thinking

Before coding, understand the context and commit to a BOLD aesthetic direction:

- **Purpose**: What problem does this interface solve? Who uses it?
- **Tone**: Pick an extreme: brutally minimal, maximalist chaos, retro-futuristic, organic/natural, luxury/refined, playful/toy-like, editorial/magazine, brutalist/raw, art deco/geometric, soft/pastel, industrial/utilitarian, etc. There are so many flavors to choose from. Use these for inspiration but design one that is true to the aesthetic direction.
- **Constraints**: Technical requirements (framework, performance, accessibility).
- **Differentiation**: What makes this UNFORGETTABLE? What's the one thing someone will remember?

CRITICAL: Choose a clear conceptual direction and execute it with precision. Bold maximalism and refined minimalism both work - the key is intentionality, not intensity.

Then implement working code (HTML/CSS/JS, React, Vue, etc.) that is:

- Production-grade and functional
- Visually striking and memorable
- Cohesive with a clear aesthetic point-of-view
- Meticulously refined in every detail

Frontend Aesthetics Guidelines

Focus on:

- **Typography**: Choose fonts that are beautiful, unique, and interesting. Avoid generic fonts like Arial and Inter; opt instead for distinctive choices that elevate the frontend's aesthetics; unexpected, characterful font choices. Pair a distinctive display font with a refined body font.
- **Color & Theme**: Commit to a cohesive aesthetic. Use CSS variables for consistency. Dominant colors with sharp accents outperform timid, evenly-distributed palettes.
- **Motion**: Use animations for effects and micro-interactions. Prioritize CSS-only solutions for HTML. Use Motion library for React when available. Focus on high-impact moments: one well-orchestrated page load with staggered reveals (animation-delay) creates more delight than scattered micro-interactions. Use scroll-triggering and hover states that surprise.
- **Spatial Composition**: Unexpected layouts. Asymmetry. Overlap. Diagonal flow. Grid-breaking elements. Generous negative space OR controlled density.
- **Backgrounds & Visual Details**: Create atmosphere and depth rather than defaulting to solid colors. Add contextual effects and textures that match the overall aesthetic. Apply creative forms like gradient meshes, noise textures, geometric patterns, layered transparencies, dramatic shadows, decorative borders, custom cursors, and grain overlays.

NEVER use generic AI-generated aesthetics like overused font families (Inter, Roboto, Arial, system fonts), cliched color schemes (particularly purple gradients on white backgrounds), predictable layouts and component patterns, and cookie-cutter design that lacks context-specific character.

Interpret creatively and make unexpected choices that feel genuinely designed for the context. No design should be the same. Vary between light and dark themes, different fonts, different aesthetics. NEVER converge on common choices (Space Grotesk, for example) across generations.

IMPORTANT: Match implementation complexity to the aesthetic vision. Maximalist designs need elaborate code with extensive animations and effects. Minimalist or refined designs need restraint, precision, and careful attention to spacing, typography, and subtle details. Elegance comes from executing the vision well.

Remember: Claude is capable of extraordinary creative work. Don't hold back, show what can truly be created when thinking outside the box and committing fully to a distinctive vision.

Before coding, understand the context and commit to a BOLD aesthetic direction:

NEVER use generic AI-generated aesthetics like overused font families (Inter, Roboto, Arial, system fonts), cliched color schemes

Skills are... markdown files

...with short descriptions

and optional:

- **scripts** (executable code)
- **references** (documentation)
- **assets** (templates)

- **SKILL.md** (required): Instructions in Markdown with YAML frontmatter
- **scripts/** (optional): Executable code (Python, Bash, etc.)
- **references/** (optional): Documentation loaded as needed
- **assets/** (optional): Templates, fonts, icons used in output

A Skill in practice:

SKYBRIDGE

name	mcp-app-builder
description	Guide developers through creating and updating MCP apps. Covers the full lifecycle: brainstorming ideas against UX guidelines, bootstrapping projects, implementing tools/widgets, debugging, running dev servers, deploying and connecting apps to ChatGPT. Use when a user wants to create or update a MCP app, MCP server or use the Skybridge framework.

Creating MCP Apps

Those are conversational experiences that extend AI assistants through tools and custom UI widgets. They're built as MCP servers invoked during conversations.

⚠️ The app is consumed by two users at once: the **human** and the **AI Assistant LLM**. They collaborate through the widget—the human interacts with it, the LLM sees its state. Internalize this before writing code: the widget is your shared surface.

SPEC.md keeps track of the app's requirements and design decisions. Keep it up to date as you work on the app.

No SPEC.md? → Read discover.md first. Nothing else until SPEC.md exists.

SPEC.md exists? → Read SPEC.md, then follow architecture.md to design the change. Update SPEC.md, then read the relevant Implementation references below before writing code.

Setup

1. **Copy template** → copy-template.md: when starting a new project with ready SPEC.md
2. **Run locally** → run-locally.md: when ready to test, need dev server or ChatGPT/Claude connection

Architecture

Design or evolve UX flows and API shape → architecture.md

Implementation

- **Fetch and render data** → fetch-and-render-data.md: when implementing server handlers and widget data fetching
- **State and context** → state-and-context.md: when persisting widget UI state and updating LLM context
- **Prompt LLM** → prompt-llm.md: when widget needs to trigger LLM response
- **UI guidelines** → ui-guidelines.md: display modes, layout constraints, theme, device, and locale
- **External links** → open-external-links.md: when redirecting to external URLs or setting "open in app" target
- **OAuth** → oauth.md: when tools need user authentication to access user-specific data
- **CSP** → csp.md: when declaring allowed domains for fetch, assets, redirects, or iframes

Deploy

- **Ship to production** → deploy.md: when ready to deploy via Alpic
- **Publish to ChatGPT/Claude Directories** → publish.md: when ready to submit for review

Full API docs: https://docs.skybridge.tech/api-reference.md

Name
..
architecture.md
copy-template.md
csp.md
deploy.md
discover.md
fetch-and-render-data.md
oauth.md
open-external-links.md
prompt-llm.md
publish.md
run-locally.md
state-and-context.md
ui-guidelines.md

Where Skills shine

Progressive Disclosure

Skills use a three-level system:

- **First level (YAML frontmatter):** Always loaded in Claude's system prompt. Provides just enough information for Claude to know when each skill should be used without loading all of it into context.
- **Second level (SKILL.md body):** Loaded when Claude thinks the skill is relevant to the current task. Contains the full instructions and guidance.
- **Third level (Linked files):** Additional files bundled within the skill directory that Claude can choose to navigate and discover only as needed.

This progressive disclosure minimizes token usage while maintaining specialized expertise.

Frontend Skill description:

name	frontend-design
description	Create distinctive, production-grade frontend interfaces with high design quality. Use this skill when the user asks to build web components, pages, artifacts, posters, or applications (examples include websites, landing pages, dashboards, React components, HTML/CSS layouts, or when styling/beautifying any web UI). Generates creative, polished code and UI design that avoids generic AI aesthetics.
license	Complete terms in LICENSE.txt

Where Skills shine

Pros:

- low initial token footprint
- very easy to create and edit
- great for multi-step workflows

Cons

- **Only instructions, no tools**
- **Local, not shareable**

Did Skills do it?

No!

They don’t serve the same purpose!

MCP Tools are connectors that expose new capabilities to your models

Skills are recipes that tell your model how to use these capabilities

Suspect #3:
CLI

CLI: pros and cons

Pros:

- Models love bash
- progressive discovery with --help
- Fast and token efficient (usually)
- Very composable!

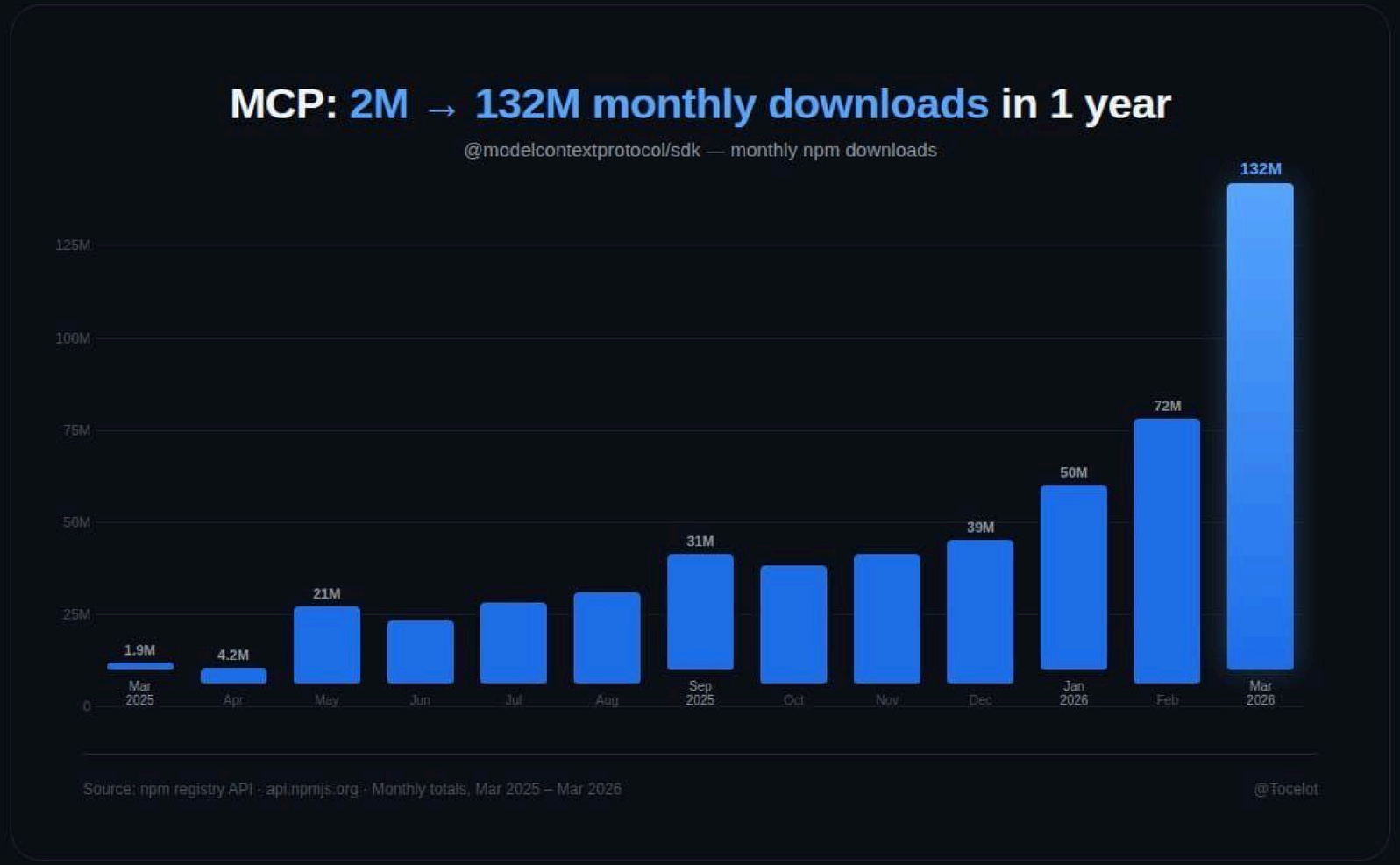
Cons

- Non-tech people can’t use them
- No standard auth, no standard in general
- Local, no multi-tenancy
- CLI providers are blind!

When to use what?

What	What for	For whom	Example
MCP only	Simple flows, UI (MCP Apps)	Everyone!	Instacart MCP App
Skills only	Reusable prompts	Writers, developers	writer-skill, frontend-design-skill
Skills + MCP	Complex workflow with access to external services	Business Operators, Developers	Inbound contact enrichments into CRM
Skills + API	Specific automation jobs	People OK API key security	Linear with GraphQL APi
Skills + CLI	Local workflows	Terminal-friendly users	PR publish/review flows

Is MCP Dead?


MCP: 2M → 132M monthly downloads in 1 year
@modelcontextprotocol/sdk — monthly npm downloads
125M
100M
75M
50M
25M
0
1.9M
4.2M
21M
31M
39M
50M
72M
132M
Mar 2025
Apr
May
Jun
Jul
Aug
Sep 2025
Oct
Nov
Dec
Jan 2026
Feb
Mar 2026
Source: npm registry API · api.npmjs.org · Monthly totals, Mar 2025 – Mar 2026
@Tocelot

These slides

Thank you!

SKYBRIDGE

Star us...

And win Ski goggles!