MCP DEV SUMMIT 2026

Putting MCP on a Diet

A Proxy for Tool Scoping & Context Compression

Prathamesh Saraf · Senior Forward Deployed Engineer @ TrueFoundry
Author of *"My Adventures with LLMs"* · Research @ IISc

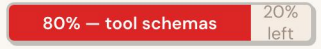
The MCP Tax

160,143

tokens loaded before your first message

Example: PostgreSQL MCP (248 tools) + GitHub MCP (41 tools).
Every tool schema, in full, at startup.

Why this hurts



Claude's 200K context window

80%	10x	289
context burned	compounding	tool schemas
before first user message	schemas resent every turn	most never used in a session

At Claude Opus 4.6 ($5/M tokens), 10-turn conversations:

$29K/yr

at 10K/day ·

$292K/yr

at 100K/day

The MCP spec has the answer

"Clients should implement progressive tool discovery — don't load all tools upfront."

— MCP Client Best Practices

Spec Recommendation

Every client must implement it

Client-side pagination & filtering

FastMCP Code Mode

Every server must opt in

Server groups tools in code

Proxy (mcp-guardian)

Nobody changes anything

Transparent layer in between

Works with unmodified clients *and* unmodified servers

One proxy. Three meta-tools.

~~160,143~~ tokens (direct)

456 tokens (proxy)

99.7% reduction

The three meta-tools

search_tools

`search_tools("tables")`

Pluggable search (keyword, semantic, …) over allowed tools. Returns names + one-line briefs.

get_schema

`get_schema("pg_list_tables")`

Full JSON input schema for one tool. Load only what you need.

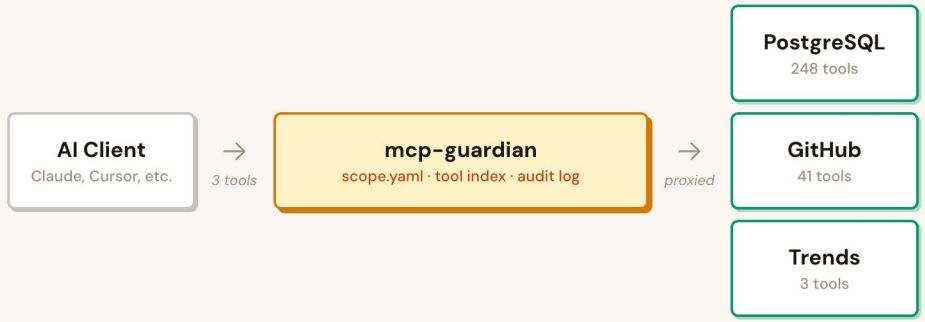
execute_tool

`execute_tool("pg_list_tables", {})`

Proxied call to the upstream server. Scope-checked + audit logged.

Progressive discovery:
search → schema → execute

Architecture



Client sees a normal MCP server · Upstream sees a normal MCP client

How a request flows

1. **`search_tools("list tables")`**
   Index lookup — no upstream call. Returns: `pg_list_tables`, `pg_describe_table`, …

2. **`get_schema("pg_list_tables")`**
   Index lookup — returns full inputSchema. Still no upstream call.

3. **`execute_tool("pg_list_tables", {})`**
   Scope check → audit log → forward to PostgreSQL MCP → return result.

Only step 3 hits upstream. Steps 1 & 2 are served from memory.

Scopes: fine-grained access control

ALLOWLIST — READ ONLY

support-agent

- ✓ pg_read_query
- ✓ pg_list_tables
- ✓ list_issues
- ✓ search_issues
- ✓ get_top_trends

14 tools out of 289 — everything else invisible

WILDCARD — MINUS DANGEROUS

developer

- ✓ Everything from 3 servers
- ✗ pg_drop_table
- ✗ pg_truncate
- ✗ delete_file
- ✗ push_files

282 tools — blocked tools can't be found, inspected, or called

Defined in

`scope.yaml`

· No code changes · Defense in depth across all 3 meta-tools

Five auth types, zero code

`none`	`bearer_env`	`static_header`	`token_passthrough`	`oauth`
Public servers	API key from env / dashboard / client	Custom header (X-API-Key)	Forward client's auth	Browser-based OAuth flow

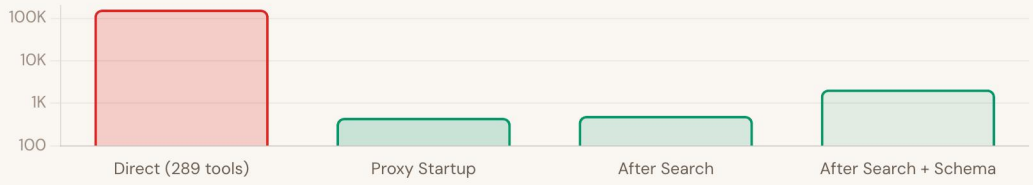
All configured in `scope.yaml` · Secrets stay in `.env` · Dashboard manages keys + OAuth sessions

LIVE DEMO

- Dashboard — server cards, connection status
- OAuth connect — GitHub via browser flow
- API key — Trends MCP from dashboard
- Tool search — find tools by natural language
- Chat Demo — talk to your tools, see token savings live

mcp-guardian dashboard

Benchmark: Token Savings (99.7%)



Tested against real servers: PostgreSQL MCP (248 tools) + GitHub MCP (41 tools)

Experiment 1 — How many tokens does the proxy save at session startup? (160,143 → 456 tokens)

Benchmark: Latency (−8ms overhead)

Does the proxy add overhead?

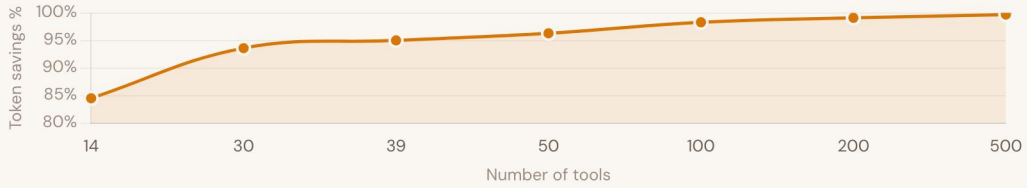
Mode	Mean	Median	P95
Direct	4,652ms	4,624ms	4,838ms
Proxy	4,644ms	4,640ms	4,709ms
Overhead	**-8ms**	**+19ms**	**-129ms**

Zero overhead

Within measurement noise. Sometimes faster — fewer schemas to load at connect.

Experiment 2 — Does proxying add latency to tool execution?

Benchmark: Scaling (95%+ at 39 tools)



Break-even at

39 tools

for 95%+ savings. Most real MCP servers are well above.

Experiment 4 — At what tool count does the proxy's savings dominate?

Benchmark: Scope Security (27/27)

9 dangerous tools tested across 3 attack vectors

Attack Vector	Result
search_tools	All 9 HIDDEN
get_schema	All 9 BLOCKED
execute_tool	All 9 BLOCKED

27/27 passed

Blocked tools can't be discovered, inspected, or executed — even if the exact name is guessed.

Experiment 5 — Are blocked tools truly invisible across all access vectors?

Adding a server: zero code

```yaml
# scope.yaml — just add this:
upstream_servers:
  github:
    url: https://github-mcp.example.com/mcp
    auth:
      type: bearer_env
      value_env: GITHUB_TOKEN

scopes:
  developer:
    servers:
      github:
        blocked_tools:
          - delete_file
          - push_files
```

Set the env var. Restart the proxy.

Done.

Complex auth? Use an MCP Gateway

OAuth, scopes, secrets, RBAC — offload to a gateway

```yaml
# Slack MCP — gateway config
name: slack-mcp
url: https://...truefoundry.cloud/slack-mcp/mcp
auth_data:
  type: oauth2
  grant_type: authorization_code
  authorization_url: https://slack.com/oauth/v2/authorize
  token_url: https://slack.com/api/oauth.v2.access
  client_id: "2879..."
  client_secret: tfy-secret://...
  scopes:
    - channels:read
    - chat:write
    - users:read
    # ... 10 more scopes
```

TrueFoundry MCP Gateway
handles:
OAuth flows · Secret management
RBAC · Rate limiting · Per-user isolation

Public MCP Playground

Open source. Try it today.

```
git clone github.com/S1LV3RJ1NX/mcp-guardian
cd mcp-guardian
uv sync --dev
cp examples/scope.direct.yaml scope.yaml
uv run mcp-guardian --scope support-agent
```

MIT
Licensed

Python
Built with FastMCP

Docker
Ready to deploy

Thank you

Questions?

leanpub.com/adventures-with-llms

"My Adventures with LLMs"

Prathamesh Saraf

Senior FDE @ TrueFoundry

Author · Researcher @ IISc

/S1LV3RJ1NX

/sarafpr

/s1lv3rj1nx

github.com/S1LV3RJ1NX/mcp-guardian