Scaling AI Agent Productivity through a Single Smart MCP Proxy

1 Motivation and facts

Large-language-model agents thrive on rich toolsets, yet practical and provider-imposed ceilings cap how many function schemas we can feed them at once. The numbers below illustrate why naïvely loading *all* tools is a dead-end and set the stage for a smarter gateway.

- **Practical ceilings**: Cursor IDE is limited to 40 simultaneously loaded tools in real coding sessions.
- **Hard API caps**: Leading providers enforce a *128-function* limit per call (e.g., OpenAI Function Calling).
- **Growing public registries**: 4 400 + Model Context Protocol (MCP) servers on *mcp.so* (April 2025) already outpace fixed context windows (ScaleMCP paper).
- **Token cost of schemas**: Injecting thousands of function schemas bloats prompts—blank conditioning with 2 134 tokens yielded just **13.62 %** correct selections on MCPBench tasks (see RAG-MCP paper).

2 Smart MCP Proxy Benefits

The Smart MCP Proxy collapses entire tool catalogs behind a single intelligent endpoint, letting agents pull in only what they need, when they need it. This section highlights how that design translates to leaner prompts and higher success rates.

- **One tool, many endpoints**: The proxy exposes a single `retrieve_tools` function; sub-tool name & args travel inside the payload, eliminating schema clutter.
- **Token savings**: Hiding 450 + OpenAPI endpoints behind one proxy removes ~99 % of schema tokens (see RAG-MCP paper).
- **Accuracy retention**: When combined with RAG-MCP retrieval, the proxy maintains the **43 %** accuracy edge while staying within minimal prompt size.

3 Solution Design

At a glance, the proxy is a **thin federating gateway** that hides thousands of upstream MCP tools behind **one smart entry-point**.

1. **Startup pipeline**
   • Load JSON/YAML config → spin up FastMCP *clients* for every listed server (URL or local command).
   • Fetch each server's `tools/list`, hash & persist metadata in **SQLite**, embed descriptions with the selected backend (BM25 / HuggingFace /

OpenAI) and store vectors in **Faiss**.
• Nothing is exposed to the agent yet—only the single `retrieve_tools` function.

2. **Query-time flow (`retrieve_tools`)**
   • The agent passes its natural-language intent.
   • The proxy scores the corpus, picks the **top K** matches (default 5).
   • Depending on the routing mode:
   - **CALL_TOOL (default)** – returns metadata and lets the agent call the tool through the proxy's universal `call_tool(name, args)` method.
   - **DYNAMIC** – auto-registers lightweight wrappers for each match, fires a `tools/list_changed` notification, and the agent can invoke them directly.

3. **Execution path**
   • A wrapper simply forwards the call to the relevant upstream server via FastMCP **as-proxy**, streams the response back, and (optionally) truncates large payloads.
   • A bounded **tool pool** (env `MCPPROXY_TOOLS_LIMIT`, default 15) evicts the coldest+lowest-scoring wrappers to keep memory lean.

4. **Observability & Safety**
   • Built-in logging, per-origin rate limits, and optional OAuth tokens per server.
   • Optional `MCPPROXY_LIST_CHANGED_EXEC` hook executes any shell command (e.g., `touch ~/.cursor/mcp.json`) after tool list changes to refresh clients that ignore standard notifications.

This architecture trades a small, constant prompt footprint for on-demand discovery, keeping **accuracy high** while staying **well below provider function-schema limits**.

4 Client Compatibility

Smart MCP Proxy speaks pure MCP; any compliant client can call it unmodified.

Client / Framework	Status	Observation
Cursor IDE	Tested (v1.0.0)	Proxy registers as *one* function, bypassing 40-tool soft limit
Anthropic Claude Desktop	Tested (2025-05)	Desktop agent consumes proxy through standard MCP URI
Google ADK Framework	Tested (v1.3)	Registered as canonical MCP server; end-to-end calls succeed

5 References (for more details)

- **RAG-MCP** – T. Gan & Q. Sun, "Mitigating Prompt Bloat in LLM Tool Selection via Retrieval-Augmented Generation" (arXiv:2505.03275)
- **ScaleMCP** – E. Lumer *et al.*, "Dynamic and Auto-Synchronizing MCP Tools" (arXiv:2505.06416)
- **OpenAI** – Docs (OpenAI Function Calling))