CYCLR

RESEARCH REPORT

MCP Server Design and Token Efficiency

What 30 controlled tests reveal about building MCP servers that are cheaper, faster, and more reliable.

HubSpot | Oracle NetSuite | QuickBooks | Claude Haiku 4.5 | GPT-5-mini

Prepared for software teams evaluating or building MCP servers · Connector platform: Cyclr

THE EXPERIMENT

Three variables, thirty configurations

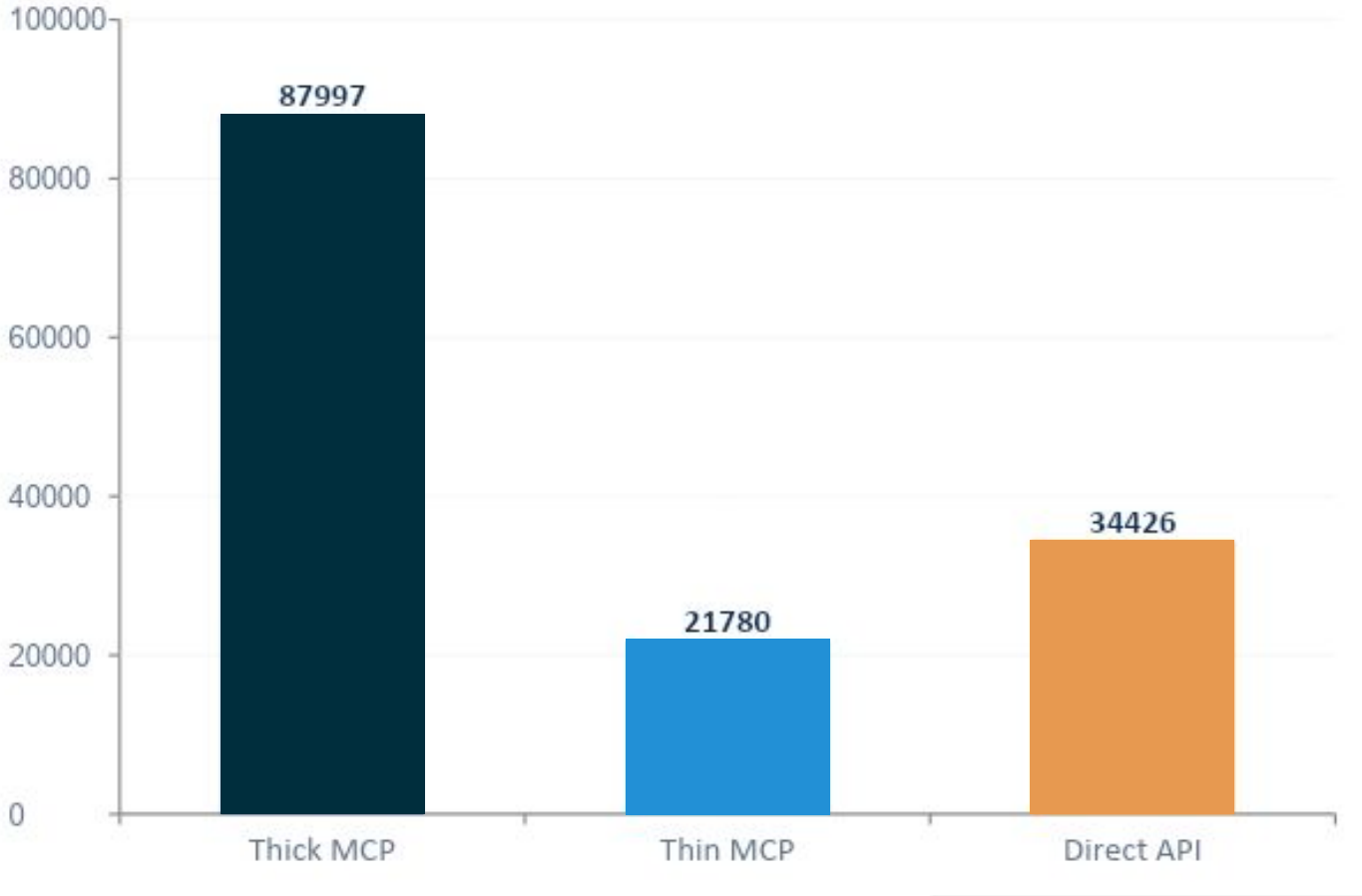
We asked identical business questions three different ways, across three SaaS systems and two models — and measured the tokens each path consumed.

Application	Model	Connection method
• HubSpot — CRM • Oracle NetSuite — ERP • QuickBooks — accounting	• Claude Haiku 4.5 • GPT-5-mini	• Thick MCP — full API (24–38 tools) • Thin MCP — task-scoped (4–7 tools) • Direct API

30 configurations **2** questions each **1** metric that matters most: tokens

HEADLINE RESULT

The connection method swings cost by up to 4X



75%

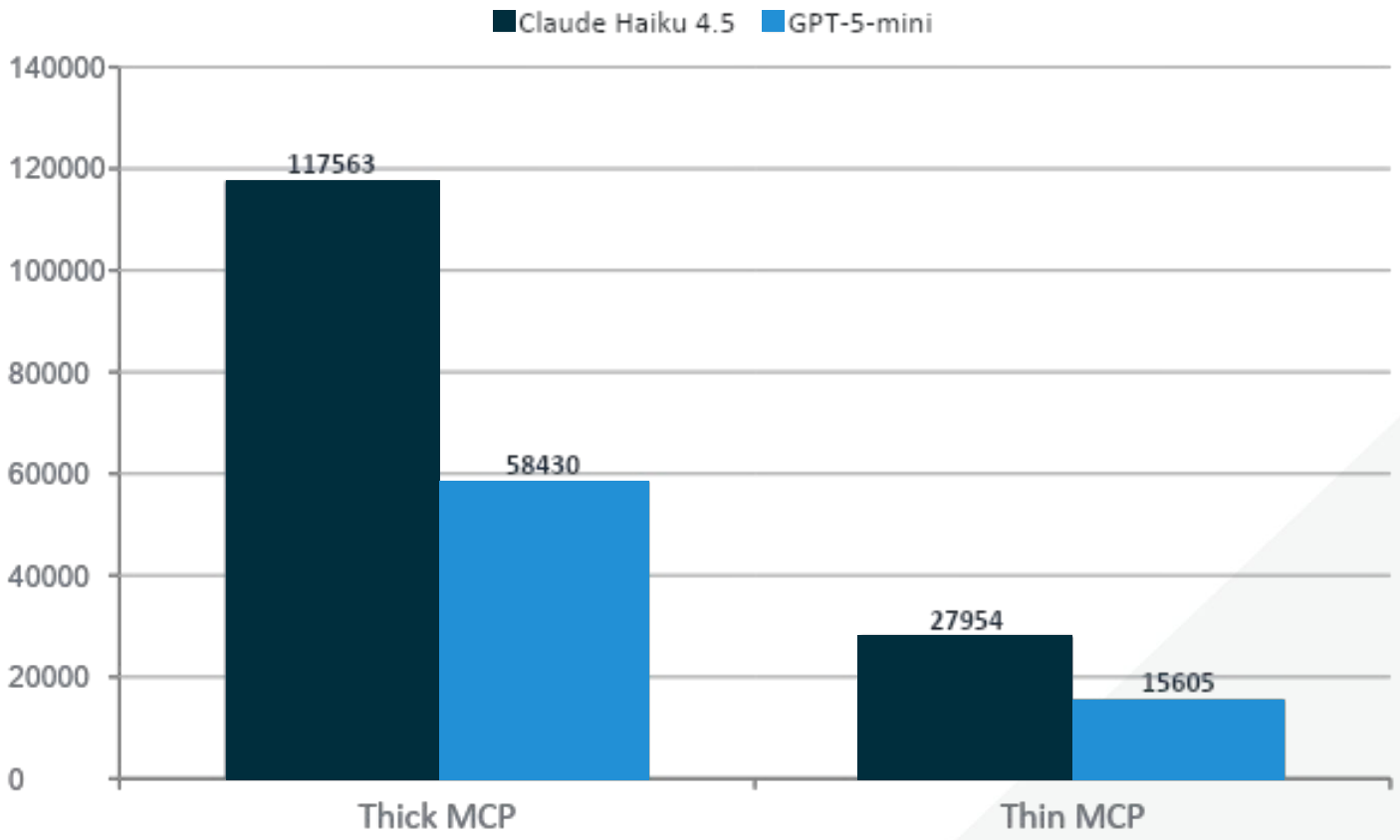
fewer tokens per task with a Thin MCP server vs. a Thick one

Output is not the cost.

Across every run, model output was just 2.6% of all tokens. The bill is the context you send in — exactly what server design controls.

COST DRIVER #1

Surface area beats model choice



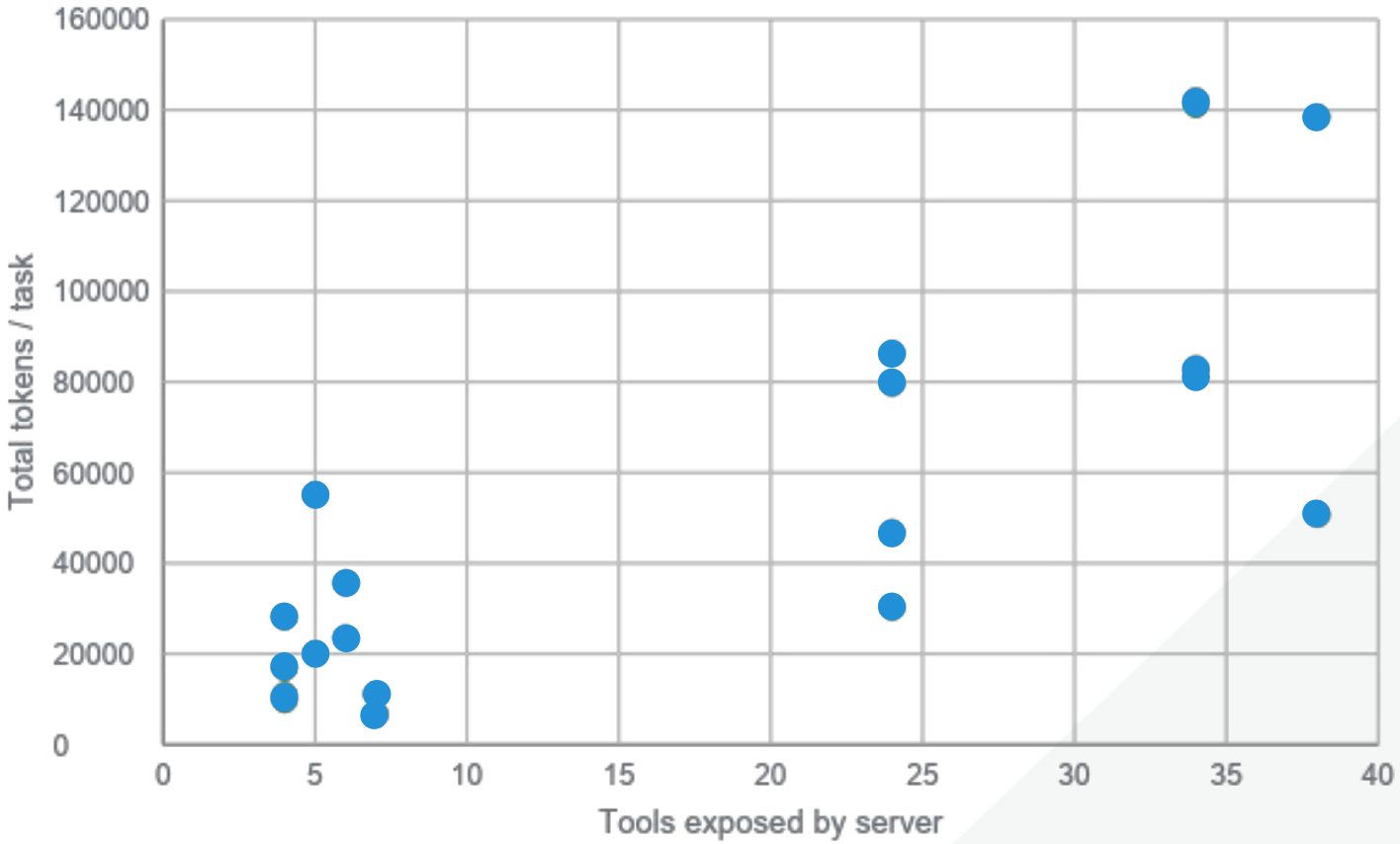
76% saved on Claude Haiku

73% saved on GPT-5-mini

A wasteful design hurts every model — and hurts the priciest model the most.

COST DRIVER #2

Every exposed tool is a “Recurring Tax”



The pattern is near-monotonic

Thin servers (4–7 tools) cluster at the bottom-left.

Thick servers (24–38 tools) sit at the top — paying schema cost on every turn, then pulling big payloads through context.

THE COUNTER-INTUITIVE FINDING

“Direct API” wasn't the cheap option

With no tool definitions to load, Direct API should be cheapest. It wasn't — and it was the least reliable.

+58%

more tokens than Thin MCP, despite zero schema overhead

33%

clean first-answer accuracy — worst of the three methods

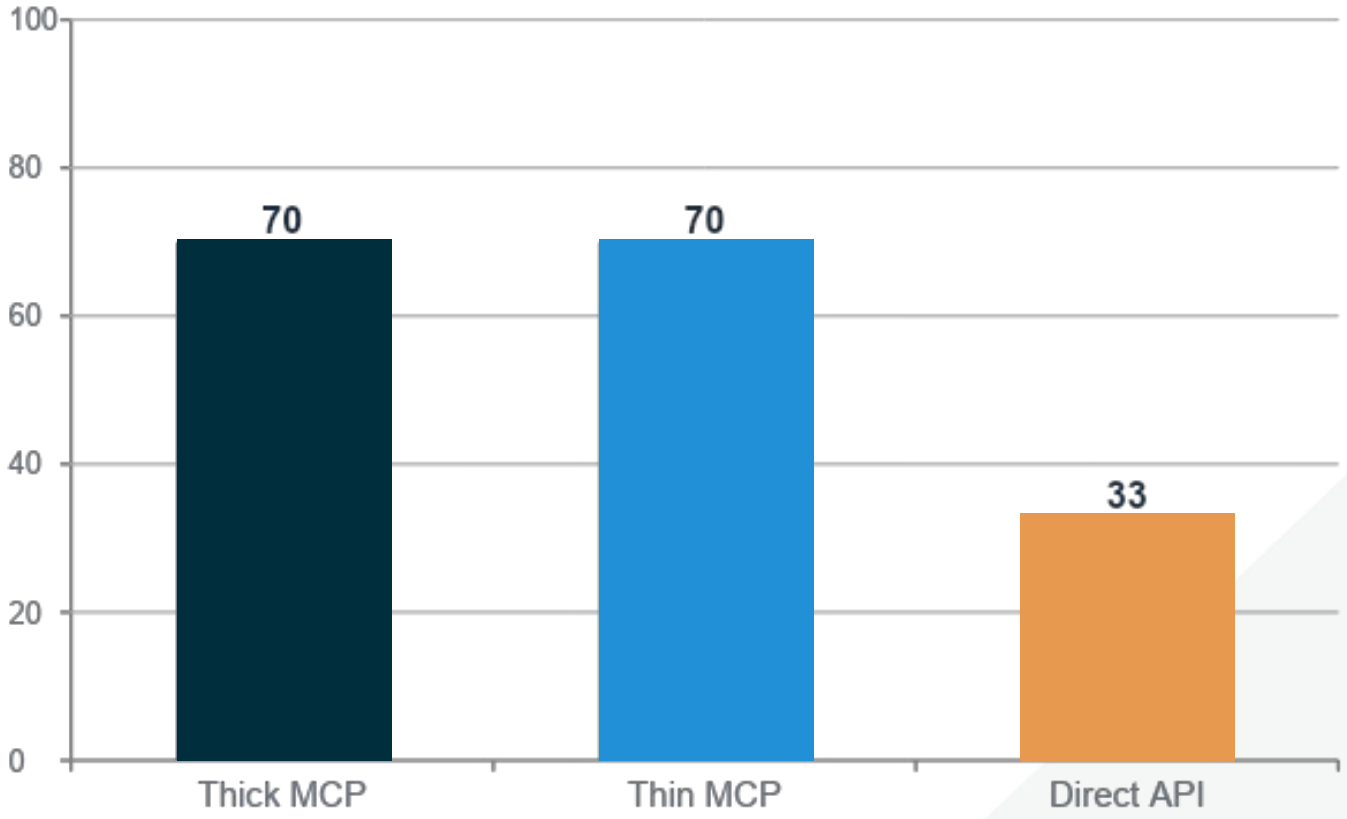
40–60k

tokens burned on some NetSuite runs that still didn't finish

The retry tax. Without a typed schema, models guessed method IDs and parameter names, hit errors like “method requires a PurchaseOrderId,” and looped through corrections. The tokens you save on schema, you spend several times over on trial-and-error — and still get a worse answer.

CHEAPER, AND JUST AS CORRECT

Trimming tools didn't cost accuracy



Clean first-answer accuracy (Q1)

Design lesson: the base-price miss

Every configuration failed one NetSuite question — the item's base price.

The models behaved correctly; the value simply lived in an unexposed price sub-resource. **That's a server design gap, not a model fault — surface the fields users ask about.**

WHY IT HAPPENS

Backed by independent research

200–1,400

tokens per tool definition

Loaded into context on every request. A 50-tool server burns 10k–25k+ tokens before the user speaks. [StackOne, MindStudio]

17,600

tokens for one real-world server

GitHub's official MCP server spends ~17,600 tokens on tool definitions every request; stack a few servers and you're past 30,000 before any work begins. [StackOne]

Up To 85%

accuracy drop as tools grow

Large tool sets don't just cost tokens — they make models choose worse. Showing only relevant tools tripled selection accuracy. [LongFuncEval; RAG-MCP]

Our 30-test result is the same mechanism at smaller scale: tool definitions are a fixed, per-request tax — on cost and on correctness.

WHAT TO BUILD

Six rules for an efficient MCP server

Expose the fewest tools that cover real tasks

~75% fewer tokens, no accuracy loss.

Keep the typed MCP layer — don't go raw

The schema prevents costly retry loops.

Design endpoints around the questions asked

Surface the fields users actually request.

Control response bloat

Compact, field-filtered, paginated payloads.

Treat every tool call as a recurring tax

Each one costs tokens per request — and too many cut accuracy.

Measure tokens per tool and per task

Instrument, track schema share, set alerts.

CYCLR

THE BOTTOM LINE

Design for scarcity.

Expose only the tools a task needs, surface the fields users ask about, keep responses compact, and keep the typed layer in place.

75%	No	Avoid
lower token cost with a Thin MCP design	accuracy penalty for trimming the tool set	the retry tax of raw, untyped API access

A well-scoped MCP server is the single most effective cost-and-quality control available to an AI product team.