Context CompressMCP

A hook-based compression layer that silently saves 40% of your context window on every MCP tool response

APRIL 20, 2026

THEDECIPHERIST.COM

TABLE OF CONTENTS

If you've ever hit Claude Code's context limit mid-session, watching the model forget earlier parts of your conversation because tool responses ate all the space, CompressMCP was built for exactly that problem.

It's a hook-based compression layer that silently intercepts MCP tool responses, compresses large JSON payloads using key abbreviation, and hands Claude a smaller-but-identical dataset to reason over. No configuration per-tool. No changes to your prompts. Just install once and it works.

Average savings: **40% fewer tokens** on JSON responses, with mathematically guaranteed lossless recovery.

THE PROBLEM IT SOLVES

Claude Code talks to the world through MCP (Model Context Protocol) tools, GitHub, databases, REST APIs, file systems. These tools often return large JSON payloads: lists of repositories, database query results, API responses with hundreds of records.

Here's the thing: Claude has to read all of that. Every field name, every bracket, every nested key. When you're asking GitHub for 80 repositories, you get back something like:

```
[
  {
    "repositoryId": 1000,
    "repositoryName": "project-alpha",
    "repositoryDescription": "A backend service for...",
    "repositoryOwner": "theDecipherist",
    "isPrivate": true,
    "stargazerCount": 12,
    "forkCount": 0,
    "createdAt": "2024-01-15T10:22:00Z",
    ...
  },
  ...79 more objects
]
```

The field names, `repositoryId`, `repositoryName`, `repositoryDescription`, repeat 80 times. They contribute thousands of tokens that carry zero information after the first occurrence. Claude already knows what `repositoryId` means. The 79 subsequent repetitions are pure overhead.

CompressMCP's answer: replace those repeated keys with single-character abbreviations, and hand Claude a dictionary to decode them.

```
[compressmcp: 9,719→5,319 tokens (-45%) | lossless]
Keys: {"repositoryId":"a","repositoryName":"b","repositoryDescription":"c","rep
[{"a":1000,"b":"project-alpha","c":"A backend service for...","d":"theDecipheri
```

9,719 tokens becomes 5,319. Claude reads the `Keys` line once, then works through the abbreviated data, and it understands it perfectly, because LLMs handle this kind of dictionary-key encoding trivially.

HOW THE COMPRESSION WORKS

The core mechanism is **TerseJSON**, a lossless key-abbreviation algorithm. It builds a dictionary mapping each unique field name to a short identifier (starting from `a`, `b`, `c` ...), then replaces every occurrence throughout the JSON structure with the abbreviated key.

Here's the full pipeline:

1. Receive the raw JSON string from an MCP tool response
2. Count tokens using the official Anthropic tokenizer
3. Check the threshold, if it's below 500 tokens, skip it (the overhead isn't worth it for small responses)
4. Compress using TerseJSON: build the dictionary, abbreviate all keys at all nesting levels
5. Format the output as three lines (header, dictionary, data)
6. Return it to Claude Code as a replacement tool output

The output is always those same three lines:

```
[compressmcp: {original}→{compressed} tokens (-{pct}%) | lossless]
Keys: {"originalFieldName":"a","anotherField":"b",...}
{abbreviated JSON data}
```

The header shows you the savings so you can actually see it working. The `Keys` line is the decode dictionary. The third line is the compressed data. Claude reads all three and works with the data as if it received the original, because it has everything it needs to reconstruct it perfectly.

Nothing is dropped. Nothing is summarized. Every value is byte-identical to the original.

ARCHITECTURE: THREE COMPONENTS WORKING TOGETHER

CompressMCP isn't a single thing, it's three coordinated components that install into Claude Code's hook system.

1. PostToolUse Hook (The Main Compressor)

This is the heart of it. It registers as a `PostToolUse` hook with the matcher `mcp__.*`, which means it intercepts every single MCP tool response before Claude sees it.

When a tool returns a response, CompressMCP receives the raw output, checks if it's JSON, counts the tokens, and compresses it if it's above the 500-token threshold. Below that? It passes through completely unchanged, no processing, no overhead, Claude never knows it was there.

2. PreToolUse Hook (The curl/wget Interceptor)

This one watches Bash commands before they execute. If Claude tries to use `curl` or `wget` to fetch data, this hook blocks it and redirects:

```
Use mcp__compressmcp__fetch instead of curl/wget to get lossless JSON compressi
URL: https://api.example.com/data
```

Why bother? Because `curl` and `wget` return raw HTTP responses directly into the context, no hook can intercept them after the fact. By redirecting to the MCP fetch tool, CompressMCP makes sure the response goes through the compression pipeline instead of landing raw.

3. MCP Fetch Server

A standalone MCP server (`compressmcp --server`) that exposes a single tool: `mcp__compressmcp__fetch`. This is the HTTP client that the pre-hook redirects curl/wget calls to.

It supports GET, POST, PUT, PATCH, DELETE, custom headers, and request bodies. When a JSON response comes back, it's automatically compressed before being handed to Claude.

So nothing slips through. Whether it comes from GitHub tools, database tools, file tools, or HTTP APIs, JSON has to go through compression before it reaches Claude's context.

REAL-WORLD BENCHMARKS

These numbers come from actual test runs against realistic datasets using the official Anthropic tokenizer.

Token Savings by Dataset

DATASET	ORIGINAL TOKENS	COMPRESSED TOKENS	SAVINGS
Orders (100 records)	6,673	4,123	**-38%**
GitHub repos (80)	9,719	5,319	**-45%**
Users (200 records)	13,741	7,941	**-42%**
Analytics events (500)	28,781	17,656	**-39%**
Products catalog (60)	6,748	4,198	**-38%**
Average	–	–	**-40%**

What Savings Scale With

The longer your field names, the better the compression. That makes sense when you think about it: longer names mean more redundant repetition across hundreds of records, which means more tokens to recover.

AVERAGE KEY LENGTH	SAVINGS
4 characters	-12%
6 characters	-21%
10 characters	-31%

The minimum key length for abbreviation is 3 characters. Keys shorter than that pass through unchanged, there's no real savings to be had abbreviating `id` to `a`.

What This Means in Practice

A 40% reduction in JSON tokens doesn't just let you squeeze in one more tool call. It compounds across a session.

Every token CompressMCP saves is a token available for more of your conversation history, more code in the context, more reasoning on the current turn, more tool calls before hitting the limit.

On a Claude Pro plan with a 200K context window, consistently saving 40% on MCP responses means you can work significantly longer on a problem before needing to start a new session. That's the real value.

THE LOSSLESS GUARANTEE

"Lossless" is the word CompressMCP uses, and it's not just marketing, it's a hard technical requirement that the test suite enforces.

The safety tests verify byte-identical round-trip recovery for:

- All primitive types: strings, numbers (integers and floats), booleans, null
- Empty strings (distinct from null)
- Empty objects and empty arrays
- Deeply nested structures (5+ levels)
- Special characters and Unicode in values
- Duplicate values across different keys
- Arrays of 500+ items
- Error records at any position in an array
- Short keys (2 chars or less, which pass through unchanged)

There are 22 explicit lossless test cases plus a battery of passthrough corruption tests that verify non-JSON content arrives unchanged. None of the 262 total tests involve any data transformation or summarization, only key abbreviation and dictionary recovery.

If a future code change broke lossless recovery on any of those cases, CI would catch it before it ships.

THE STATUS BAR

CompressMCP includes a live status bar that appears in Claude Code's footer during every session.

```
[███████▒] 73% | 145K/200K  sonnet · ⚡ -4.7M tok · 40% · 2,341 calls | 5h [█
```

Reading left to right:

ELEMENT	MEANING
`[███████▒] 73%`	Context window fill (color-coded: green, yellow, red)
`145K/200K`	Tokens used / total context size
`sonnet`	Current active model
`⚡ -4.7M tok`	Total tokens saved this session via compression
`40%`	Average compression efficiency
`2,341 calls`	Total tool calls this session
`5h [████▒▒▒▒▒▒] 23%`	5-hour rate limit utilization
`7d [██████▒▒▒] 45%`	7-day rate limit utilization
`main`	Current git branch

The context fill bar turns yellow when you hit 50% and red at 80%, giving you a heads-up before you actually start losing context. It reads live data from Claude Code's internal pipes and the session JSONL logs in `~/.compressmcp/`.

INSTALLATION

CompressMCP installs as a global npm package and registers itself into Claude Code's settings automatically.

```
npm install -g compressmcp
compressmcp install
```

That's it. Restart Claude Code after installation and everything activates on the next session.

The install command adds five components to `~/.claude/settings.json`:

1. **PostToolUse compress hook,** matcher `mcp__.*`, runs `compressmcp --hook`
2. **PostToolUse tracker hook,** matcher `.*`, runs `compressmcp --track` (silent context logging)
3. **PreToolUse curl interceptor,** matcher `Bash`, runs `compressmcp --pre-hook`
4. **MCP server,** `compressmcp --server`
5. **Status line,** `compressmcp --status`

Verify Installation

```
compressmcp check
```

This tells you whether each hook, the MCP server, and the status line are all properly registered.

Uninstall

```
compressmcp uninstall
```

Removes all five components from `~/.claude/settings.json`. Clean removal with no leftovers.

THE SETTINGS IT PRODUCES

Here's what gets added to `~/.claude/settings.json` after installation:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "mcp__.*",
        "hooks": [
          { "type": "command", "command": "compressmcp --hook" }
        ]
      },
      {
        "matcher": ".*",
        "hooks": [
          { "type": "command", "command": "compressmcp --track" }
        ]
      }
    ],
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          { "type": "command", "command": "compressmcp --pre-hook" }
        ]
      }
    ]
  },
  "mcpServers": {
    "compressmcp": {
      "command": "compressmcp",
      "args": ["--server"]
    }
  },
  "statusLine": {
    "type": "command",
    "command": "compressmcp --status"
  }
}
```

No per-project configuration needed. The hooks fire globally across all projects.

SESSION TRACKING

CompressMCP logs every compression event to `~/.compressmcp/` in JSONL format. Each line is either a compression event or a context event.

Compression event:

```
{"type":"compress","tokensSaved":2550,"tokensIn":6673,"ts":1713607200000}
```

Context event:

```
{"type":"context","inputTokens":45000,"cacheCreation":5000,"cacheRead":1000,"mc
```

These logs drive the status bar's lifetime statistics. The tracker hook (`compressmcp --track`) writes a context event after every tool call, even non-MCP ones, so the status bar has accurate context window data regardless of whether compression actually fired.

THE TECHNICAL STACK

- **Language:** TypeScript, compiled to CommonJS
- **Runtime:** Node.js 18+
- **Compression:** `tersejson` library (key abbreviation algorithm)
- **Token counting:** `@anthropic-ai/tokenizer` (official Anthropic tokenizer, accurate to Claude's actual token budget)
- **Protocol:** JSON-RPC 2.0 over stdio (MCP standard)
- **Hook integration:** Claude Code PostToolUse / PreToolUse hook system

The switch to the Anthropic tokenizer in v0.3.0 was a deliberate accuracy call. The earlier `length / 4` estimate was fast but could be off by 10-15% depending on the content. The token counts you see in the status bar and savings header now match what Claude Code actually charges against your context limit.

WHEN COMPRESSMCP FIRES (AND WHEN IT DOESN'T)

CompressMCP compresses when:

- An MCP tool response contains valid JSON

- The JSON is above 500 tokens (Anthropic tokenizer)
- The JSON has field names with 3 or more characters that benefit from abbreviation

CompressMCP passes through unchanged when:

- The response is below the 500-token threshold
- The response isn't JSON (plain text, HTML, binary)
- The tool isn't an MCP tool (file reads, code execution, etc.)
- All field names are 2 characters or less

The pre-hook only fires when a Bash command starts with `curl` or `wget`. Everything else runs normally.

LIMITATIONS

Compression overhead. Each compression run spins up a Node.js process. The baseline is around 60ms per tool call. For large responses (500+ items, 10K+ tokens) that overhead is easily outweighed by the tokens Claude doesn't have to read. For small responses below the threshold, the hook doesn't fire at all.

Claude still needs to decode. Claude reads the dictionary line and then works with abbreviated keys. LLMs handle this without any issues in practice, the dictionary is small and the mapping is consistent. But there is a small token cost to parsing the `Keys` line. It's minor relative to the savings, but it's there.

Key abbreviation only. CompressMCP only shortens keys, not values. String values, numbers, everything else passes through intact. That's intentional, value abbreviation would require semantic understanding to be safe, and the goal here is lossless.

No configuration UI. The 500-token threshold and the 3-character minimum key length are baked in at install time. Changing them means editing the source directly.

WHERE TO GET IT

CompressMCP is open source and available on npm:

```
npm install -g compressmcp
```

Source code: github.com/TheDecipherist/compressmcp

THE BOTTOM LINE

Claude Code's context window is a finite resource. Every token your tools consume is a token not available for your code, your conversation, and your reasoning. CompressMCP is a simple, automatic, lossless way to get roughly 40% of those tokens back on every MCP tool response.

Install once. Works everywhere. Nothing to think about.

```
npm install -g compressmcp && compressmcp install
```