The Token Tax: How AI Agents Get 60 to 95% Cheaper

Kabui, Charles

2026-06-30

Table of Contents

Introduction

Every time an AI coding agent reads a file, runs a command, or searches a codebase, it pays for the result twice: once to send that text to the model as input, and again for every word the model writes back. Those words are tokens, the chunks of text a model bills by. A single agent run can burn hundreds of thousands of tokens, and most of them are noise: stack traces, repeated file contents, verbose logs, and polite filler. You pay for all of it at the model's per-token rate, and on a busy team that bill compounds fast. A wave of 2026 tools now attacks this hidden tax from five different angles. Some squeeze the text going in, some trim the text coming out, some tighten the instructions, some compress the model's working memory, and some reroute traffic to dodge rate limits. This post maps all five layers with real numbers, and shows where each tool actually saves money and what it costs you in return.

---

1. Compress the input: Headroom

Headroom sits between your agent and the model and compresses everything the agent reads, including tool outputs, logs, search results, files, and chat history, before it reaches the model. The claim is **60 to 95% fewer tokens with the same answers**. Its own benchmarks show a code search dropping from **17,765 tokens to 1,408, a 92% cut**, and an incident-debugging

session falling from **65,694 to 5,118, also 92%** (benchmarks). It does this with content-aware compressors: one for structured data like JSON, one that understands code structure, and a small trained text model for prose.

The compression is reversible. Headroom keeps the originals on your machine and the model can call a retrieve tool if it ever needs the full text back. You can run it four ways: as a library you call in code, as a drop-in proxy that needs zero code changes, as a one-command wrapper around agents like Claude Code or Cursor, or as an MCP server (MCP is the open standard agents use to call outside tools). Two extras push the savings further. A cache-aligning step keeps the start of each prompt stable so the provider's own cache keeps hitting, which on Anthropic costs **$0.50 per million tokens instead of the $5 base rate**, and a `headroom learn` command mines your failed sessions and writes the fixes back into the agent's memory file so it stops repeating them. On accuracy tests it kept grade-school math (GSM8K) exactly even and nudged a truthfulness benchmark up by **+3%**, so the smaller prompts did not cost correctness. It is open source under the Apache 2.0 license.

---

2. Compress the output: Caveman

Headroom mostly shrinks what you send. Caveman attacks the other half of the bill: what the model writes back. This is the expensive half. Anthropic charges **$5 per million input tokens but $25 per million output tokens for Claude Opus 4.8**, so a word saved on the way out is worth five on the way in (Anthropic pricing).

Caveman is a skill, a plug-in set of instructions, that tells the agent to "talk like caveman": drop the "Sure, I'd be happy to help" preambles and the restated code, keep only the technical content. Across ten real prompts measured on the Claude API it cut output by an **average of 65%, ranging from 22% to 87%**, while keeping the actual fix intact. It has four levels of terseness, from a light mode that only drops the filler up to a near-telegraphic mode and even a classical-Chinese setting that packs the most meaning per character, and it leaves your language, code, commands, and error strings exactly as they were. A companion command, `caveman-compress`, rewrites a memory file like CLAUDE.md into the same terse style and trims about **46% of the input tokens** that file costs you on every session. So even an output-side tool reaches back to the input side.

---

3. Compress the instructions: SkillOpt

There is a third thing every agent carries: its skill files, the plain-language instructions that tell it how to behave. Microsoft's SkillOpt treats that text as something to optimize directly. It runs the agent on scored tasks, reflects on what worked and what failed, and proposes small edits to the instruction file, keeping an edit only if a held-out test improves. On GPT-5.5 that lifted average accuracy by **+23.5% with zero added inference cost**, because the skill is just context tokens at runtime, and the file it produces stays compact, typically 300 to 2,000 tokens. We covered the method in full in SkillOpt. The link to compression is direct: a tighter, better-tuned instruction file does more with fewer words, so the standing overhead of "how to act" shrinks while quality climbs.

---

4. Compress the memory and the index: TurboQuant and TurboVec

The layers above shrink text. The next two shrink numbers. While a model generates, it holds a working memory called the KV cache, and that cache is the main thing that fills up GPU memory on long contexts. Google Research's TurboQuant squeezes that cache to **3 bits per value with no measurable quality loss and no retraining, a 6x memory cut**, and computes attention up to **8x faster on H100 GPUs**, which we unpacked in Google TurboQuant. The same trick moves to retrieval. Most agents pull context from a vector database, which stores text as long lists of numbers. TurboVec uses that algorithm to pack a **10-million-vector store from 31GB down to 4GB, a 16x cut, with no training step** and faster search than the standard library. Cheaper memory and a smaller index mean longer context and more retrieved facts for the same hardware bill.

---

5. Route around the limits: 9router

Even with everything compressed, you still hit provider rate limits, the "you have sent too many requests" wall that stops a coding session cold. 9router is a local gateway you point your tools at, one address on your own machine that speaks the common OpenAI format, and it spreads requests across more than 60 providers. Its **three-tier fallback** tries your paid subscription first, then cheaper providers, then free tiers, so when one quota runs out it auto-switches instead of failing, and a session that would have died at 2am keeps running on a free tier instead. It also folds in compression: a built-in feature it calls RTK auto-compresses tool results like git diff and grep output before they reach the model, losslessly and on by default, and it ships Caveman mode for the output side too. One tool spans two of our layers at once: it routes around limits and compresses the input.

6. The tradeoffs: reversibility, accuracy, and security

These tools are not free wins, and the differences matter. The first axis is reversibility. Headroom's compression is reversible because it caches originals and lets the model fetch them back, while a terse-output skill like Caveman is one-way: the words it drops are gone. The second is lossless versus lossy. 9router's RTK and TurboQuant are lossless or near-lossless by design, while prose compression and caveman-speak are judgment calls that trade a little nuance for a lot of tokens, which is why every serious tool here ships accuracy benchmarks.

The third axis is the one to watch: security. A proxy that compresses your traffic is software sitting in the middle of every prompt, and a router that balances load across providers holds your API keys and sees your code. Running them locally, as all of these do, keeps your data on your machine, but you are still trusting a third-party process with everything your agent reads and writes. Pooling free tiers also raises terms-of-service questions, since some providers forbid sharing keys across accounts. If you only adopt one layer, start where your spend actually goes: input compression for agents that read large files and logs, output trimming for chatty models where the replies dominate the bill. Treat any middle-man tool the way you would treat a new dependency: read what it does with your data before you wrap your whole workflow in it.

7. Conclusion

The cheapest token is the one you never send. The tools of 2026 give you five distinct places to act on that, and the strongest setups stack them rather than picking one:

- **Input:** Headroom compresses tool outputs, files, and history before they reach the model, with reversible cuts of 60 to 95%.
- **Output:** Caveman trims the model's replies by about 65%, where each token saved is the costly kind.
- **Instructions:** SkillOpt tightens the skill files so the agent does more with fewer words.
- **Memory and index:** TurboQuant shrinks the KV cache 6x and TurboVec shrinks the vector store 16x.
- **Routing:** 9router dodges rate limits across 60+ providers and compresses tool output on the way through.

Different layers, stackable, with a real catch at each step: weigh reversibility, accuracy, and the trust you place in any tool that sits in the middle of your agent's traffic. Cutting cost is no longer about switching to a smaller model. It is about sending less to the one you already use.

References

1. Headroom Labs (2026). *Headroom: the context compression layer for AI agents.* GitHub.
2. Headroom (2026). *Benchmarks and methodology.* Headroom Docs.
3. Headroom (2026). *Kompress-v2-base model card.* Hugging Face.
4. Brussee, J. (2026). *Caveman: why use many token when few do trick.* GitHub.
5. Brussee, J. (2026). *caveman-code.* GitHub.
6. 9Router (2026). *9Router: free AI router with smart fallback.* 9router.com.
7. Microsoft Research (2026). *SkillOpt.* GitHub.
8. Google Research (2025). *TurboQuant.* arXiv:2504.19874.
9. Codrai, R. (2026). *TurboVec.* GitHub.
10. Anthropic (2026). *Model pricing.* Claude Platform Docs.