running 4× Claude Code Max still isn't enough — here's what actually helps

A power user's read on caveman, context compression, and every "save tokens" plugin your feed is pushing. Most of them optimize the cheapest slice of the bill.

Yiğit Konur · Apr 21, 2026 · 10 min read

▶ The useful bits

i run **four 20× Claude Code Max subscriptions** in parallel. roughly 80× the default allotment. and i still hit the wall.

not because i'm doing anything exotic. because Claude Code, at serious production volume, burns tokens faster than you can throw money at it. the subscription cap isn't really a "cap" — it's the speed at which a well-orchestrated agent can shovel context into an LLM. past a certain workload threshold, a single Claude Code instance doing everything end-to-end becomes the bottleneck.

so i built gossip — a small orchestration layer where **Claude plans**, **Codex executes**, and the two talk to each other over a structured channel. Claude is the architect, Codex is the labourer. horizontal scaling across providers, not vertical scaling on one.

but before you get to that, most people hit the token ceiling and start Googling. they find **caveman**. they find **context compression plugins**. they find twenty Medium posts promising 75–95% savings. and they install all of them.

so the question this post is trying to answer, honestly:

Skip to content

> *…e tokens" stuff is real, which is bait, and how do you actually make a Claude Code workflow survive production load?*

i spent a day reading every top Reddit thread, every benchmark i could find, the actual caveman source, and the Anthropic docs. here's the deal.

what caveman actually is

caveman is a Claude Code skill by Julius Brussee. 14k stars. the pitch: strip all the "Certainly! I'd be happy to help..." fluff from Claude's output. keep the code. drop the preamble.

mechanically, it injects a system rule on session start that tells Claude to respond in compressed, fragment-heavy prose. no articles, no hedging, no pleasantries. three intensity levels (`lite`, `full`, `ultra`). code blocks, file paths, commit messages, tool calls — all untouched. auto-disables for security warnings and anything where ambiguity is dangerous.

install is one line:

```
npx skills add JuliusBrussee/caveman
```

the README claims ~75% output-token reduction. that number is where it gets interesting.

the math the README doesn't show you

here's the thing nobody in the viral threads mentions: **caveman only touches output tokens**. in a real Claude Code session, output is the small slice of the bill.

the cleanest teardown i found was Mejba's, which actually instrumented a real session. roughly:

bucket	tokens per session	what caveman does to it
inp	~15,000	nothing
input: conversation history (re-read every turn)	~35,000	nothing
input: files Claude reads into context	~25,000	nothing

Skip to content

bucket	tokens per session	what caveman does to it
output: Claude's prose + tool calls	~25,000	cuts ~75% of the prose portion
total	**~100,000**	**~4,500 tokens saved**

that's a **~4.5% reduction on the actual bill**, not 75%. maybe $15–$20/month if you're on heavy API usage. nice to have. not a revolution.

the caveman author, to his credit, admitted this on Hacker News: the 75% number came from preliminary testing, not a rigorous benchmark, and the skill was never intended to reduce hidden reasoning tokens.

where the other 95% of your tokens actually go:

```
┌──────────────────────────────────────────────┐
│  CLAUDE CODE TOKEN BURN — REAL DISTRIBUTION  │
├──────────────────────────────────────────────┤
│  repo exploration / file scanning     ~35%   │  ← biggest sink
│  conversation history re-reads        ~25%   │  ← compounds every turn
│  MCPs + skills loaded into context    ~15%   │  ← quietly brutal
│  extended thinking / reasoning        ~15%   │  ← the real expense
│  output prose (caveman hits this)     ~10%   │  ← the small slice
└──────────────────────────────────────────────┘
```

what Reddit actually thinks

i went in expecting the typical cargo-cult worship. what i found was a pretty sober community, at least under the surface-level hype posts.

r/ClaudeCode: *"does caveman plugin really help with context usage?"*

small thread, 14 comments, but the signal is tight. the top reply from **u/ConnectTransition660** reported actual usage — about 30% savings in practice, not 75 e README.

Skip to content

the most-upvoted critical comment, from **u/Kaskote**:

> *cool idea, but this optimizes the cheapest part of the bill.*

that single line is the whole analysis. output tokens are the cheap part. input context — repos, history, tool schemas — is where the money actually goes, and caveman doesn't touch any of it.

u/Revolutionary-Tough7 dropped the other key insight:

> *it's not the prompts that cost the money. it's the thinking.*

on subscription plans, you get ~19M tokens per 5-hour window. saving a few thousand on output prose doesn't move that needle. disabling extended thinking when you don't need it does.

r/ClaudeAI: *"taught Claude to talk like a caveman to use 75% less tokens"*

this is the viral post that put caveman on the map. 12.6k upvotes, 581 comments. the top comment is **u/fidju** with the joke that basically explains the whole project:

> *why waste time say lot word when few word do trick?*

12.4k upvotes on that alone. but underneath the meme layer, the substantive critiques landed. one of the higher-ranked serious replies:

> *forcing Claude to talk like a caveman might actually make it dumber.*

the argument: by forcing the model into a "less intelligent persona," you're potentially degrading reasoning quality along with the prose. sounds plausible. is it true? short answer, based on the actual benchmarks: **no**. Mejba's side-by-side testing showed first-attempt success rates actually went *up* slightly (64% → 71%) with caveman mode, an[Skip to content]rXiv paper on brevity constraints found forcing concise responses can improve accuracy in large models by up to 26 percentage points on some

benchmarks. counterintuitive, but there's a real effect — verbose defaults seem to encourage fluff-as-reasoning.

r/ClaudeCode: *"I saved $60 by building this tool to reduce Claude Code token usage"*

this is where the conversation gets more sophisticated. the tool is a pre-indexing layer that keeps Claude from re-exploring your repo on every task. the author's benchmark showed **54% fewer tokens**, and the comments mostly agreed that repo exploration — not prose — is where the real waste lives.

a recurring comment pattern across this thread and the Kilo Code discussion: **CLI output is the hidden killer**. test runners, compilers, linters, dev servers — all of them spew verbose output that gets fed back to the LLM verbatim. one thread reported 10M tokens saved over two weeks just by filtering CLI noise before it hit the model. that's ~89% savings on a narrow but common workflow.

r/ClaudeCode: *"don't use Claude Code's default system prompt"*

different angle: skip the plugin ecosystem entirely, override the system prompt with `--system-prompt` and keep it under 500 tokens of your own rules. the consensus: **CLAUDE.md is already doing 90% of what matters** for most workflows, and the default system prompt is bloated because it's trying to serve everyone.

u/AgreeableFall5530 — a comment that combined the install pitch with honest math:

> *75% is not realistic for normal English in my experience.*

their follow-up recommendations (short CLAUDE.md, rip out MCPs and replace with CLI flows, avoid pasting huge logs, hook-based PDF-to-markdown conversion) got more upvotes than the caveman pitch itself. the community, when you read carefully, is already a step ahead of the viral content.

Skip to content

the actual hierarchy of things that save tokens

if the Reddit sentiment and the benchmarks agree on anything, it's this: **caveman is fine but it's #7 on the list**. here's the impact-to-effort ranking based on what the threads and the measurements actually support:

rank	intervention	effort	realistic savings	notes
1	turn off extended thinking for routine tasks	1 min	10–20%	Reddit's most underrated lever. re-enable for architecture work.
2	audit and nuke unused MCPs + skills	30 min	15–25%	some people have 160+ skills registered. each one taxes every call.
3	pre-index your repo (ai-codex, GrapeRoot, ContextKing, Serena)	15 min	30–50% on exploration-heavy work	stops Claude from grep-ing your codebase on every task.
4	filter CLI output before it enters context (RTK, Headroom)	20 min	up to 89% for test/build-heavy loops	the hidden killer for anyone running `npm test` in a loop.
5	start fresh sessions for unrelated tasks	0 min	10–15%	do not chat-continue a massive session for a one-line fix.
6	use `/model haiku` for simple tasks, opus only when needed	0 min	20–40% on cost (not token count)	routing is cheaper than compression.
7	caveman plugin	5 min	~4–5% total	funny, harmless, marginal. install it, move on.
8	short CLAUDE.md with concise directives	5 min	5–10%	"be concise. no filler. conclusions first." — does 80% of caveman for free.
9	custom `--system-prompt` override	20 min	variable, mostly behavioural	more about quality than tokens.

caveman is not bait. it works, it's free, it's a 5-minute install. it's just not the thing that will save you if you're hitting the wall at production volume. the list above is roughly in order of what will actually make a difference.

when none of this is enough

here's the uncomfortable truth for anyone running Claude Code at real volume: **token optimization plugins are a rounding error compared to the load of a serious workflow**.

if you're running multiple parallel coding agents, shipping features daily, doing research + r[Skip to content]n the same pipeline — a single subscription is not going to cut it, and stacking every caveman-style plugin in existence isn't going to change that. you

can maybe squeeze 30–40% more runway out of optimizations. you cannot 10× throughput by being clever with prose.

what actually works at that scale is **horizontal scaling**:

1. multiple subscriptions running in parallel on separate workloads. annoying to orchestrate but real.
2. split planning and execution across different models/providers. planning is cheap, execution is expensive. let the expensive model do less thinking.
3. offload noisy work to cheaper agents. have a Haiku-tier model summarize test output before it hits your main agent's context. have Codex do grunt edits while Claude supervises. use what each model is good at.
4. cache aggressively and avoid cache-busting moves. changing the model mid-conversation, toggling thinking settings, re-ordering tool lists — all of these can invalidate prompt caching and re-cost you the entire session's history.

this is basically why i built gossip — Claude plans, Codex executes, they communicate over a structured channel. not because caveman is bad. because the problem caveman solves is at the wrong altitude for this kind of workload.

actionable, in one screen

if you skimmed the whole thing, here's what to do, in order:

Skip to content

```
WEEK 1 — free wins, zero risk
├─ [ ] turn off extended thinking by default (huge, underrated)
├─ [ ] run /doctor, audit installed skills and MCPs, remove anything unused
├─ [ ] add 4 lines to CLAUDE.md: "be concise. no filler. no hedging.
│       conclusions first. skip pleasantries."
├─ [ ] start a new session for any task that isn't a direct continuation
└─ [ ] stop changing models mid-conversation (cache-busts everything)

WEEK 2 — light tooling
├─ [ ] install a repo pre-indexer (ai-codex, Serena, ContextKing)
├─ [ ] if you run tests/builds in loops, add a CLI output filter
├─ [ ] install caveman if you want the joke — it does help a little
└─ [ ] measure with /usage before and after every change

WEEK 3 — structural
├─ [ ] if still hitting limits, look at horizontal scaling —
│       multiple subs, multi-provider orchestration
├─ [ ] split planning vs execution across models
├─ [ ] consider moving the noisy stuff off-agent entirely
└─ [ ] only now is caveman's 4-5% actually worth optimizing for
```

tldr

caveman is a clever skill. it's fun. it works as advertised — *on the specific thing it targets*. the problem is that the thing it targets is the cheapest slice of your bill, and the viral content implied otherwise.

the Reddit community, once you read past the meme replies, already knows this. the serious comments keep pointing at the same handful of actual levers: kill extended thinking when you don't need it, stop loading 100 skills you never use, pre-index your repo, filter CLI noise, start fresh sessions. those are the things that buy you 50%+ breathing room, not 4%.

and if you're at the scale where even all of that combined isn't enough — welcome to the [Skip to content] going to plugin your way out. you're going to architecture your way out. multiple accounts, multiple models, smart orchestration. that's where the actual headroom lives.

install caveman. have a laugh. then go do the real work.

Sources & further reading

1. caveman by Julius Brussee
2. Mejba's caveman teardown — real token instrumentation
3. r/ClaudeCode: Does caveman plugin really help with context usage?
4. r/ClaudeAI: Taught Claude to talk like a caveman to use 75% less tokens
5. r/ClaudeCode: I saved $60 by building this tool to reduce Claude Code token usage
6. r/ClaudeCode: Don't use Claude Code's Default System Prompt
7. arXiv: Brevity constraints and reasoning accuracy in large models (March 2026)
8. gossip — Claude plans, Codex executes

Questions & answers

- So caveman is bait?
- Which single change gives the biggest win?
- Does forcing Claude to be terse make it dumber?
- When does plugin stacking stop working?

See you down the next rabbit hole.

More notes →