The Token Tax: measuring the ceiling on read-gating in agentic coding sessions

Hud Wahab · Multimode AI · 15 September 2026

Abstract

A published Claude Code plugin reports token savings of roughly 90% (~96 million tokens) by intercepting whole-file reads and delegating them to a cheaper model. An independent JetBrains benchmark of a comparable third-party tool (`rtk`) measured end-to-end cost **increasing by 7.6%**. We reproduce the structural cause of this discrepancy on four production repositories using a clean-room size gate and a transcript-derived measurement tool.

We find that read-gating is bounded far below the reported figure. Reads a gate would intercept represent **2.8–7.7%** of all reads, while cache reads — the prior conversation re-sent on every turn — represent **over 99.99% of input-side tokens in all four repositories, with no exceptions**. We introduce a *ceiling* metric: an upper bound on what a gate could save, computed from sessions already run and deliberately constructed to favour the gate. The ceiling is **0.022%–0.319% of all tokens**.

Against this, we measure the cost of a single gate-induced round trip at **6.2x the saving from the read it prevented**. We identify the dominant cost driver as conversation length rather than file content: a query in a fresh session costs $0.07 versus $0.43 in an established one (**6.6x**), an effect that plateaus under context compaction rather than growing without bound.

We conclude that self-reported savings counters and billed cost are not measuring the same quantity, and that the discrepancy is structural rather than a defect in either measurement. Tooling is released under MIT at github.com/multimodeai/token-tax-kit.

1. Motivation

Token-reduction tooling for agentic coding assistants is typically evaluated by the tool's own counter: the tool estimates the tokens that *would have been* consumed absent its intervention, and reports the difference as savings.

This is a counterfactual about a session that did not occur. The tool both generates the hypothesis and scores it. Where such figures have been checked against billed cost by independent parties, they have diverged — in the JetBrains case, in sign as well as magnitude.

This paper asks a narrower and checkable question: **for a size gate on file reads, what is the maximum possible saving, measured from sessions that actually happened?**

2. Method

2.1 Instruments

Size gate (`hooks/size-gate.sh`). A Claude Code `PreToolUse` hook on the `Read` tool. Clean-room reimplementation from described behaviour; no vendor source was consulted. In `observe` mode it logs what it would have intercepted and allows the read; in `enforce` mode it denies the read and instructs a targeted re-read. It never gates reads that already carry `offset` / `limit`, missing files, files at or under threshold, or binary files.

Reporter (`bin/token-tax`). Node, zero dependencies. Parses Claude Code's own session transcripts, summing `message.usage` fields per assistant turn and classifying every `Read` tool call. By design it **emits no savings figure** — the critique motivating this work applies to our tooling as well.

2.2 Corpus

Four repositories in active use, chosen for stack diversity, not for result. Two are public and independently reproducible; two are shown but not re-runnable, and are reported in a separate trust tier for that reason.

Repo	Stack	Sessions	Public
mm-cli	TypeScript CLI	8	yes
prowasl	Supabase edge functions + Node/TS	2	no
bayanlab	Python	1	no
kaya-editor	JS/TS editor	1	yes

2.3 The ceiling metric

For each read the gate would have intercepted we compute two bounds. **Counted once** is the file's token cost charged one time, estimated at 4 characters per token; this under-counts, because content read into a session persists and is re-sent on subsequent turns. **Counted every re-read** adds a cache read of those tokens on every subsequent turn the file is plausibly still in context. The second is the strongest honest case *for* the gate.

Two bounds keep it defensible: accumulation stops at a context reset (a turn whose cache-read volume falls below half the preceding turn's, indicating compaction), and stops if cached context is smaller than the file, since the file cannot reside in a context smaller than itself.

Both bounds assume the gate is free — never refuses wrongly, never triggers a re-read, never adds a turn. This is false in practice. It is assumed deliberately, so the result is an upper bound rather than an estimate.

2.4 Pricing

Published list prices for Claude Opus 5 ($5/M input, $25/M output), verified 12 September 2026. Cache reads bill at 10% of input (verified). The cache-*write* multiplier was not independently verified and is exposed as a tunable parameter (default 1.25) rather than embedded as a constant. Models without verified published rates report token counts and no dollar figures.

3. Results

3.1 Read gating is a small share of reads

On mm-cli the gate would intercept **34 of 1,233** reads (2.8%). Of the remainder, 513 were already targeted reads carrying `offset` / `limit`, which a gate ignores by design.

3.2 Cache reads dominate input-side tokens, universally

Repo	Gate-eligible reads	Cache-read share of input tokens
mm-cli	2.8%	99.991%
prowasl	5.6%	99.996%
bayanlab	4.3%	100.000%
kaya-editor	7.7%	99.999%

The gate-eligible proportion varies by a factor of roughly 2.7 across repositories. The cache-read share does not vary meaningfully at all — a single-session repository is indistinguishable from an eight-session repository to four decimal places. We had predicted the opposite: that low-history repositories would show materially less re-sending. That prediction was wrong.

3.3 Tokens and dollars are differently distributed

On mm-cli, cache reads are about 96% of tokens but about 63% of cost; output tokens are about 0.22% of tokens but about 7.25% of cost. Any argument conducted purely in tokens is therefore not an argument about the invoice.

3.4 The ceiling

Repo	Counted once	Counted every re-read
mm-cli	0.0032%	**0.028%**
prowasl	0.0094%	**0.059%**
bayanlab	0.0030%	**0.022%**
kaya-editor	0.0206%	**0.319%**

Even the bound constructed to favour the gate does not exceed one third of one percent in any repository.

Threshold sensitivity. Loosening the threshold from 350 to 100 lines on mm-cli raises intercepted reads from 35 to 178 (5.1x) and the ceiling from 0.028% to 0.069% (2.5x). The conclusion is insensitive to the threshold across this range.

3.5 The cost of enforcement

With the gate in `enforce` mode a denied read is recoverable — the model reissues a targeted read. Observed recovery, from transcripts: one denied full read followed by two ranged reads. **Three round trips to obtain a file requested once.** The complete file was ultimately read; the transfer was restructured, not avoided.

Quantity	Value
Mean context re-sent per turn (mm-cli)	472,617 tokens
Mean model output per turn	1,630 tokens
Mean intercepted file	8,885 tokens
Saving from blocking one file	$0.0444
Cost of one additional round trip	$0.2771
Ratio	**6.2x against**

This is a sufficient mechanism for an independently measured cost *increase*, without requiring the savings counter to be wrong about what it counts.

3.6 The dominant cost driver

Position in session	Mean cost per query
Fresh (first 10 turns)	$0.0653
Established	$0.4299
Ratio	**6.6x**

This effect does not compound indefinitely. Cost per query in the final fifth of a session is only 1.1x the first fifth — compaction bounds context growth. The increase is front-loaded and then plateaus. We flag this explicitly because the intuitive model (longer session → monotonically rising per-query cost) is wrong, and we initially held it.

4. Errors made and corrected

Reported because the paper's claim is about measurement discipline.

Binary line-counting. An early reporter counted newline *bytes* in binary files, reporting the five largest intercept candidates as four photographs and a PDF. Corrected via `file --mime-encoding`; the honest count fell from 98 to 34. A first attempt using a NUL-byte grep returned inverted results on macOS.

Unbounded persistence. The first "counted every re-read" implementation assumed a read file remains in context for the remainder of the session, producing $722 — 47% of billed cost. This ignores compaction. Bounding at context resets yielded $13.56, a 53x correction, in the direction that weakened our thesis.

Unvalidated hook output. The gate initially emitted `{"decision":"block",...}`, verified by piping JSON to the script by hand. Claude Code rejects that schema; the correct form wraps `permissionDecision` in `hookSpecificOutput`. The rejected hook **failed open** — reads proceeded. Piping into a script validates the script, not the integration.

Silent observe mode. Observe mode logged and allowed with no session-visible output, making correct operation indistinguishable from a failed hook. It now emits a message on every would-be interception.

Permission widening. The gate's allow path initially emitted `permissionDecision: "allow"`, which does not mean "do not block" — it auto-approves, bypassing the permission prompt the user would otherwise receive. A measurement instrument must not widen permissions as a side effect of observation. It now exits silently.

5. Discussion

The two numbers measure different things. A savings counter measures tokens not transmitted under an assumed counterfactual. An invoice measures tokens actually billed. These coincide only when the intervention has no second-order effects. Read-gating has a large one: denial induces additional turns, and each turn re-transmits the full conversation. Reported token savings and observed cost increase are therefore both consistent and both correct.

Why the target is small. Input-side tokens in agentic sessions are dominated — to better than 99.99% in every repository measured — by re-transmission of accumulated conversation. File reads are a minor and non-recurring contributor. An intervention confined to file reads addresses under 0.32% of the total in the best case.

6. Limitations

This is not a refutation of the published savings figure. Delegating a large read to a cheaper model plausibly does avoid most of those tokens; that claim may be accurate for the workload on which it was measured. What we measured is a different question: how often that arises on these machines, and how much of the bill it could touch.

Nor is it a general claim about gating. It is four repositories, one machine, one assistant, one pricing regime, over a bounded period. Two of the four are independently checkable; two are not. Token counts for files are estimated at 4 characters per token, not tokenised exactly. The cache-write multiplier is assumed, not verified.

Whether carrying conversation history is *worth* 6.6x per query is a judgement about the work being done, and this paper does not answer it. It establishes the price.

7. Reproduction

The public repositories in our corpus are `multimodeai/mm-cli` and `multimodeai/kaya-editor`. Results on those are directly checkable.

```
git clone https://github.com/multimodeai/token-tax-kit
cd token-tax-kit
chmod +x hooks/size-gate.sh bin/token-tax

./bin/token-tax report  --project /path/to/your/repo
./bin/token-tax ceiling --project /path/to/your/repo

# real before/after:
./bin/token-tax snapshot before.json
#   ... enable the gate, do comparable work ...
./bin/token-tax snapshot after.json
./bin/token-tax compare before.json after.json
```

`compare` reports change in turn count alongside change in tokens. A token reduction accompanied by a turn increase is the failure mode this paper describes.

MIT licensed. We are specifically interested in counter-examples: repositories where the ceiling is materially higher. Four is a small n.

Related

Step-by-step instructions for running this on your own repository: the guide. The narrative version: the Substack.