The two skill-auditors: what they cost, and what they fixed

green = saved ≥20% · red = cost ≥20% more · black = within ±20% = direction-at-best. The ±20% band is the noise floor this doc demonstrates (cross-session drift ~8pp + N=1 task variance), so within-band magnitudes are not coloured as signal. Colours the "% saved" columns only; swing/pp and token columns stay neutral.

A results sheet for `mikko-skills-quality` and `mikko-skills-freshness`: the two skills whose whole job is auditing your other *skills for token waste. Two honest questions: are they cheap to run, and did running them actually save tokens elsewhere?, and the measured answer to each. 2026-06-01.*

tl;dr: in plain English

These two skills are meta-tools. They don't ship a feature; they read your other skills and tell you which ones are quietly burning tokens. So the fair test is two questions, and they want two different numbers:

- **Are the auditors themselves cheap to run?** Yes. Measured cold-vs-with-skill on three models, both save on every one. `mikko-skills-quality` runs at **~17K tokens/use** and saves **38% / 48% / 54%** (Haiku / Sonnet / Opus). `mikko-skills-freshness` runs at **~24K/use** and saves **11% / 30% / 36%**. Aggregate across both skills and all three models: **+36%**.
- **Did running them actually save tokens elsewhere?** Yes, but the honest, durable result is one big *swing*, not a portfolio headline. The methodology these two skills embody (a taxonomy of three cost-traps) was pointed at skills across the portfolio, shipped two fixes, and turned `skills-freshness` on Haiku from **−70% → +20%**: a **+90pp** reversal with an identified cause.

The one caveat worth reading twice: **quality saves the most because it reads the least, and that is exactly why it missed two real bugs** on the stronger models. Freshness saves less, reads a little more, and stayed faithful. Cheap and thorough pull against each other; this sheet shows where each skill sits on that line.

> Full method lives in the two backing docs: the A/B cost measurement in `skill-calibration-2026-06-01.md`, and the downstream optimization in `skills-optim-study-2026-05-31.md`. This sheet is the synthesis; the percentages here are quoted from those two.

What the two skills do

Skill	What it does	Why it costs what it does
`mikko-skills-quality`	Token-economy hygiene audit. A deterministic Python pre-pass scores each SKILL.md on line-count + loop-prose smells, then the model reviews only what it flags.	**Cheapest auditor in the kit**, its triage explicitly says "don't open the SKILL.md by default," so the model reads almost nothing. That is the saving, and the blind spot.
`mikko-skills-freshness`	sha256 staleness detector. Hashes each skill against a manifest, flags drift, and reads each flagged file at `limit=80` to confirm.	Saves less than quality because it *does* open each flagged file, but that extra reading is why it stays faithful.

Question 1: are the auditors cheap to run? (the A/B)

Two arms, same task, same fixture of four installed skills. Arm A solves cold (no skill awareness). Arm B reads the SKILL.md, runs the skill's Python pre-pass, and reviews only what it flags. Tokens are the harness's `subagent_tokens` (input + output + cache-creation). N = 1 per cell. A 12-agent adversarial workflow then checked whether the cheaper arm hid any missed findings.

Skill	Model	Cold (A)	Skill (B)	Saved	%	Outcome-equivalent?
quality	Haiku	38,975	24,338	14,637	38%	yes
quality	Sonnet	33,540	17,486	16,054	48%	**no**
quality	Opus	44,316	20,227	24,089	54%	**no**
freshness	Haiku	39,300	34,869	4,431	11%	yes
freshness	Sonnet	34,535	24,127	10,408	30%	yes
freshness	Opus	48,437	31,161	17,276	36%	yes

Per-skill (ratio-of-sums): quality **+47%** (116,831 → 62,051); freshness **+26%** (122,272 → 90,157). **Per-model (both skills):** Haiku **+24%**; Sonnet **+39%**; Opus **+45%**. **Overall: +36%** (239,103 → 152,208; 86,895 saved).

The trade in one data point

4 of 6 pairs were outcome-equivalent. The two that weren't are both `mikko-skills-quality`, both on the stronger models:

- **quality / Sonnet** passed `mikko-skills` as clean, and missed a hardcoded Windows path (`C:/Users/vandr/.claude/skills/ …`) that silently fails on any other machine. Its pre-pass checks line-count and loop-prose; a hardcoded path is neither, and it never opened the file to see it.
- **quality / Opus** rated `mikko-audit` a generic MEDIUM and named nothing: where the cold arm rated it HIGH and pointed at ~400 lines of duplicated embedded prompt templates.

The kicker: `mikko-skills-freshness` **caught the same hardcoded path the quality arm missed**, precisely because it reads each flagged file at `limit=80`. Read-light saves more and misses more; read-some saves less and stays faithful. That contrast *is* the result.

Question 2: did running them save tokens elsewhere? (the downstream)

Round 1 of the optimization study ran these two skills (it refers to them by their unprefixed library names, `skills-quality` / `skills-freshness`), then did a forensic pass over the transcripts. It found three concrete ways a procedural SKILL.md turns *against* token economy: each visible in the JSONL, each with a one-line guard that neutralises it. **This taxonomy is the transferable artifact.**

#	Cost-trap	How the prose triggered it	The guard that fixes it
1	**Unlimited read**	"read each SKILL.md" → the skill arm read **233,876 chars** across 14 files in full (cold arm: 116,731)	`limit=80`, frontmatter + a section or two is all you need

#	Cost-trap	How the prose triggered it	The guard that fixes it
2	**Uncapped follow-up**	a missing cap let the agent chase broken path refs into the source repo: +4 Bash calls, **+28K tokens**	"one `ls` per finding, max 3 traces, don't spelunk into source repos"
3	**Batch invitation**	"read in parallel" got staged as a 6+8-file batch, paying **+27K cache-creation tokens**	cap the batch; don't pre-stage a large parallel read for the model

The common root: each fired because the procedure *removed the scoping pressure* an unguided model still feels. A model rations its own reads when it's paying for them; "read each X" hands it permission to stop rationing. The guard puts the ceiling back.

Fixes that shipped, and what they moved

- `claude-skills` PR #18: added the three cost-trap rules and tightened `skills-freshness/SKILL.md` step 3 with `limit=80` + "don't spelunk."
- `Spacepotatis` PR #280: applied the same trace-cap to `content-audit`.

The headline is a **swing**, not an aggregate, because across rounds the cold arm holds its role and only the SKILL.md changes, so the swing targets the fix:

Cell	Before	After	Swing	Mechanism / fix
skills-freshness / Haiku	−70%	+20%	**+90pp**	unlimited read → `limit=80` (PR #18)
content-audit / Haiku	−5%	+11%	**+16pp**	uncapped tracing → trace cap (Spacepotatis #280)
skills-quality / Haiku	+17%	+54%	,	same investigation-collapse pattern

The durable claim: bounded, scoped procedural language in a SKILL.md *reliably* cuts token use for **Haiku-class** models. For **Opus-class**, the rounds 1–5 *swings* (before-fix minus after-fix, a difference of two N=1 ratios) sat inside the measurement noise; that was a property of the fragile N=1 swing, not a verdict on Opus. **Round 6 re-measured both auditors' Opus cells as a *level* (skill-vs-cold) at depth (N=5/arm, stable), and found a real +76% save each:** going cold, Opus does the full audit (116–200K tokens), while the script-backed skill short-circuits to ~30–46K, and the earlier near-zero/negative reads were N=1 artifacts of an unusually shallow cold arm. So the honest read: **Haiku-class benefits at any depth; Opus-class benefits too, once the task is deep enough that going cold actually does the work**: trust direction and magnitude, not the exact %, since cold-arm cost is task-framing-sensitive. Full round-by-round tables and the noise-floor discussion are in `skills-optim-study-2026-05-31.md`.

What this sheet does NOT claim

- **No portfolio-wide save rate.** Both halves are small-N: the A/B is two skills on a four-skill fixture, N=1 per cell; the downstream is three skills, N=1 per cell. Direction and rough magnitude only.
- **Quality's 48–54% is partly a coarser read**, not pure efficiency. It missed two real findings on Sonnet and Opus to get there.
- **The +36% aggregate is ratio-of-sums** (volume-weighted), not a typical-cell figure, and the downstream "+16%" round was concentrated in two Haiku cells.
- **A re-run yields different absolute numbers at N=1.** Opus single-cell *swings* ran up to 67pp between rounds, which is exactly why round 6 re-measured the noisiest cells as deep *levels* (N=5/arm) instead of trusting N=1 swing differences.

- **These two are meta-skills (they audit other skills rather than ship a portfolio feature**), so their results live in this sheet rather than as portfolio skill-registry rows. (A registry *scan* keys on the library name and may still list them; the value story is here, not in the catalog.)

Appendix: cost of the measurements

- The A/B: 12 arms = **391,311** tokens; the 12-agent equivalence workflow = **401,602** tokens.
- The downstream study: 42 sub-agents across 5 rounds, **~4.48M** subagent tokens.

Backing docs: `skill-calibration-2026-06-01.md` (full A/B method + per-pair equivalence verdicts) and `skills-optim-study-2026-05-31.md` (full five-round optimization method + the three mechanisms in detail). Raw A/B data: `SKILL-CALIBRATION-2026-06-01.json`.