Elis Token Saver on Claude: Right-Sizing Cuts Cost 83% — and This Time, Tokens Too

A four-complexity benchmark of Elis-planned execution on a Claude Code host, with latency, accuracy, automation planning, and settlement integrity measured end to end.

Elis AI Research · Published 2026-08-13 · Version 0.3-draft · Companion to the ChatGPT-host study

Status: working draft (companion to the ChatGPT-host study of 2026-08-13)

Abstract

Elis Token Saver plans work privately — intent routing, task decomposition, and abstract model-tier guidance — and hands the host chat one task at a time to execute with its own models. We replicated the four-complexity ChatGPT-host benchmark on a **Claude Code host** through the new `/api/mcp/claude` mount. Across the same four tasks, Elis-directed execution used **28.9% fewer visible tokens** and cost **83.4% less** than plain frontier execution ($0.0235 vs $0.1413). Every case saved money; per-case cost reductions ranged from 78.7% to 92.6%.

Two mechanisms drove the improvement over the ChatGPT-host result (which saved 36.3% in cost while *increasing* tokens 40.7%): the planner produced leaner task graphs in this run (no over-decomposition of the deep case), and the Claude model family enabled **within-frontier right-sizing** — frontier-tier tasks ran on the cheapest capable frontier model ($3/$15 per million tokens) instead of the host's default frontier model ($10/$50), a 70% price gap the ChatGPT run had no equivalent for.

Stated precisely: on this evidence, Elis makes equivalent answers **much cheaper** — it does not make them better or faster. Quality was held (14/15 vs 15/15 rubric checks), not raised; latency is strictly worse (~18 s of planning before the first task). The claim this study supports is *same quality, ~80% cheaper, slower to start* — and the decisive comparison against the naive "always use a cheap model" strategy has not yet been run (see "What this study does and does not prove").

Method

Four self-contained prompts represented simple, moderate, complex, and deep work — identical to the ChatGPT-host study. Plain execution used one default-frontier response per goal. Elis execution called the local Token Saver MCP server on the Claude mount, followed every returned task cursor, and executed each task on the server's `suggested_host_model`: `small` mapped to Claude Haiku 4.5, `frontier` to Claude Sonnet 4.5 — both suggestions resolved live from the Elis model catalog per tier band, never hardcoded. Visible prompts and outputs were counted with `o200k_base` (an approximation for Claude tokenization). Prices came from the Elis model catalog (checked 2026-08-10..12): Fable 5 at $10/$50, Sonnet 4.5 at $3/$15, Haiku 4.5 at $1/$5 per million input/output tokens.

The measurement excludes hidden system instructions, MCP schemas, reasoning tokens, Elis planning and embedding usage, and tool-call usage. Both arms were generated by the same host session, with output length allowed to follow each tier's natural verbosity — a limitation shared with the ChatGPT study and discussed below.

Results

Complexity	Plain tokens	Elis tokens	Token change	Plain cost	Elis cost	Cost change
Simple	223	214	−4.0%	$0.00991	$0.00095	−90.5%
Moderate	616	463	−24.8%	$0.02860	$0.00210	−92.6%
Complex	1,078	676	−37.3%	$0.05214	$0.00961	−81.6%
Deep	1,049	756	−27.9%	$0.05069	$0.01081	−78.7%
Total	**2,966**	**2,109**	**−28.9%**	**$0.14134**	**$0.02347**	**−83.4%**

All four plans settled through the saved-token ledger (provider `claude_code`, status `settled`) inside the anonymous free allowance; no fee was charged.

Latency

Stage	Measured
Plan creation (`start_elis_token_saver_plan`)	16.3–19.8 s (mean 18.4 s)
Task cursor advance (`get_next_elis_task`)	~0.02 s
Plan finish + settlement (`finish_elis_plan`)	~0.01 s

The plain arm has zero orchestration overhead. The Elis arm pays the planning latency once per plan — dominated by the zero-execution orchestration preview — plus a negligible round-trip per task cursor. For interactive use, ~18 s of added time-to-first-task is the real product cost of the 83% dollar saving; it amortizes well on multi-task or long-form work and poorly on one-line questions, which argues for routing trivial prompts around the planner (the intent-router short-circuit path).

Accuracy

Both arms were scored against objective, prompt-derived rubrics (word limits, required-element coverage, structural constraints — checked programmatically, no model judge):

Complexity	Plain	Elis	Rubric checks
Simple	4/4	4/4	≤120 words, exactly 3 differences, all 3 named
Moderate	4/4	4/4	8/8 criteria, matrix present, conditional rec ≤450 words
Complex	3/3	2/3	9-element coverage, 6-month structure, per-stage rollback
Deep	4/4	4/4	11-element coverage, 3 TCO scenarios, gates, assumptions
Total	**15/15**	**14/14 of 15**	

The single Elis miss is keyword-level: the complex answer defines success criteria ("zero customer-visible downtime; p95 within 10%…") but never uses the literal word "metric" the rubric scans for. Substantive coverage was equivalent; no constraint violation favored either arm. This matches the study's design intent: right-sizing must not be allowed to trade correctness for cost, and at this rubric's resolution it did not.

Why this run beat the ChatGPT-host run

1. **Leaner task graphs.** The ChatGPT-run planner split the deep case into three overlapping small-tier tasks, inflating tokens 39% and producing inconsistent sub-answers. This run planned the deep case as a single frontier task: no duplicated context, no consistency failure. Task-graph variance run-to-run is real and directly visible in the results.
2. **Within-frontier right-sizing.** On the ChatGPT host, `frontier` mapped to the same model the plain arm used — so frontier-tier tasks could only lose (the complex case cost +42.6% there). On Claude, the catalog's cheapest capable frontier model undercuts the host default by 70%, so even non-decomposable frontier work saved 78–82%. This generalizes: the saving available to a host scales with the price spread inside its model family.
3. **Tier-appropriate conciseness.** Elis-arm outputs, written to each tier's natural verbosity, ran tighter than the plain arm while covering the same required ground (verified by manual constraint review). The ChatGPT run saw the opposite — orchestration wrappers inflated output.

What this study does and does not prove

Proven: cost reduction at held quality. The saving mechanism — per-task model right-sizing — survived two hosts, two planners, and materially different task graphs. It is the one result robust across both studies.

Not proven: that Elis makes output better, or faster. Quality was held at rubric resolution, never exceeded; the ChatGPT run showed decomposition can actively harm consistency. Planning adds

~18 s before the first task. Neither study supports a "better answers" claim, and this paper does not make one.

Not yet tested: the naive-cheap counterfactual. Both studies baseline against "host always uses its default frontier model." But the trivial alternative — *host always uses its cheapest model* — would beat Elis on cost **and** latency. Elis's differentiated value is therefore only demonstrable where always-cheap fails: complex and deep tasks where the planner correctly held frontier tier to protect quality. Neither study ran that arm, and the keyword rubric used here is too shallow to detect the quality degradation always-cheap would be expected to cause on those tasks. Until the three-arm experiment below is run, "right-sizing with a quality floor" is a design hypothesis, not a measured result.

The product positioning the current evidence supports: same quality, ~80% cheaper, ~18 s slower to start — a strong trade for multi-step, agentic, and batch work; a poor trade for one-line questions, which should bypass the planner via the intent-router short-circuit.

Automation planning probe: browser and OpenClaw goals

Two automation-shaped goals were planned (not executed) through the Claude mount to measure what the planner emits for the workloads the full automation benchmark will cover.

Browser goal (compare laptops across two retailer sites, screenshot each product page, produce a recommendation): a coherent 3-task graph, all small-tier, with the generic `browser` tool correctly emitted on the screenshot task and `web_search` on the research tasks — the host-capability mapping works, and no Elis-internal tool names leaked. Planning latency 22.8 s, estimated 2,791 saved tokens.

OpenClaw desktop goal (build a budget spreadsheet from a CSV, chart it, save as PDF): a degenerate 5-task graph produced by comma-splitting the goal — the first two "tasks" are literally *"On the desktop"* and *"open the spreadsheet app"*, and no desktop-automation tool name was emitted. Two findings: (1) the decomposer needs a coherence guard for imperative device-automation phrasing; (2) automation tools are gated on org app selection, so an anonymous MCP context correctly cannot plan OpenClaw/browser *execution* — the executed-automation benchmark therefore requires an authenticated org with the automation apps installed, plus the full dev stack and OpenClaw desktop bot, and is deferred to the follow-up study.

Estimated automation cost comparison. Because these plans were not executed, costs can only be compared at planning time — two ways, both labeled estimates:

Goal	Tokens (baseline → optimized)	Host-priced: plain on Fable	Host-priced: Elis tiers (Haiku)	Est. saving
Browser (laptop compare + screenshots)	4,447 → 1,656	$0.178	$0.007	−96.3%
OpenClaw (CSV → budget sheet → chart → PDF)	7,340 → 2,720	$0.294	$0.011	−96.3%

The planner's own USD estimate is even steeper (−99%), but the measured four-complexity benchmark showed planner token estimates can be directionally wrong — the ChatGPT run *increased* tokens against a predicted saving — so these figures bound the opportunity, not the outcome. Automation savings are plausibly larger than text-task savings (many small observe/act steps suit cheap tiers; the ~18 s planning cost amortizes over a long run), but that is exactly what the executed benchmark must verify, on real runs, with host-reported usage.

Settlement finding (billing-integrity defect — found, fixed, re-verified). In the initial probe, both plans — whose cursors were advanced with stub results and **zero reported usage** — settled their full *estimated* savings (2,791 and 4,620 tokens) instead of releasing the reservation: cursor advancement was treated as execution proof. A host walking the cursor without executing was credited — and, past the free allowance, would be *charged a fee on* — hypothetical savings. The fix (`claude/token-saver-settlement-guard`, commit `211307f3`) makes settlement require host-reported actual usage: zero-usage cursor-advanced plans settle at **zero** with an `estimated_only` flag, fees never accrue from estimates, and the same commit adds the decomposition coherence guard. Re-verification against the fixed build confirmed all four behaviors: zero-usage walk → `estimated_only: true` with 0 saved tokens; the OpenClaw goal plans as **one** coherent task; finish-before-first-task reports `no_reservation`; and actual-above-baseline clamps to zero rather than going negative.

Remaining economics gap (partially closed): cost is now measured, trust is not. Post-study settlement uses catalog-priced execution cost when every reported model/token step is priceable, subtracts exposed planning overhead, converts realized dollars through configured saved-token pricing, and caps the result at the original reservation. Unpriced or tool-residue runs use a capped token-volume fallback. This recognizes a cheaper model even when token volume is unchanged and prevents under-reporting from minting credit beyond the plan. The host report itself is still not a provider receipt, however, so adversarial clients can misstate their model or token counts. Host-attested usage or sampled verification remains a prerequisite for high-assurance paid settlement.

2026-08-13 settlement integrity audit (v0.3)

Eight boundary scenarios were run through the Claude mount against the merged build (dev-18 `47c29b50`), with independent recomputation from catalog prices and the `saved_token_ledger` (running record: `docs/research/token-saver-issues-log.md`).

Verified working. Conservative cost-based settlement — the minimum of measured and estimated dollar savings, capped at the reservation; an exact implausibility boundary at 2.0× the estimated baseline (usage just under settles normally; just over flags `implausible_usage` and settles from the reserved estimate); idempotent double-finish; `settlement_basis` persisted on every ledger row; and exact allowance accounting — a fresh plan's entitlement reported 5,765 used saved tokens, matching the DB ledger sum precisely, with no double-burn between reserved and settled rows. The v0.2 under-report inflation is gone: the same probe that once credited 1,229 tokens against a 908-token estimate now settles 268 on a `catalog_priced_execution_cost` basis.

Found, still open.

1. **Basis-shopping (medium — fixed and re-verified, `4d252209`).** One `tool_tokens` unit — or an off-catalog model string — in reported usage disqualified the "fully-priced" condition and flipped settlement to the token-volume fallback: identical work settled **268** tokens on the cost basis but **908** (the full reservation cap) via the fallback, a 3.4× credit difference the reporter controlled. The fix settles every step-bearing report on the cost basis over the *priced* steps only — residue and unpriced steps earn zero (`partial_priced_execution_cost`), and the token-volume fallback survives only for step-less reports. The full audit rerun on the fixed build confirmed: the tool-residue probe now settles identically to its clean twin, and the unknown-model probe settles zero — an unverifiable claim can no longer out-credit a verifiable one. The rerun also surfaced — and a follow-up fix (`f3a34d45`) closed — a new finding: the anonymous path's host-model-only preview stub returned $0.00 cost estimates with crude token baselines (1,444 → 286 for the identical goal), silently degrading the conservative min-of-measured-and-estimated to measured-only. The stub (intentional isolation from the embedding-backed preview) now prices its estimates from the catalog — optimized arm at the cheapest small-tier model of the baseline's family, baseline arm at the resolved baseline model — and a `planning_usage.estimator_unpriced` health flag exposes any future degenerate estimate. A second full audit rerun confirmed the conservative minimum re-engaged (the estimate-level probe settled $0.01243, down from the measured-only $0.02052) with every I1 property intact.

A third fix (`66b68e0b`) then closed the I5 false-positive itself by scoping the implausibility guard to reports the cost basis cannot bound: catalog-priced step-bearing reports — already capped in both directions — no longer flag however large their honest volume (the dogfood-shaped probe, 17.3k priced tokens against an under-300-token text baseline, now settles unflagged at zero savings and zero fee), while step-less token-volume reports remain guarded, since over-reporting there is the fee-dodging vector. The same commit closed a hole the basis-shopping fix had opened: wholly unpriced step reports (all off-catalog models) used to settle at zero — free-allowance stretching for anonymous users, fee dodging for paid ones — and now settle from the reserved estimate on a new `unpriced_report` audit basis. The final audit state: every probe settles on a bounded, named basis; the one remaining flag (step-less heavy usage) is the intended anti-gaming behavior, avoided entirely

by per-step usage reporting. 2. **Volume-proportional credit (low).** On the cost basis, credited savings rise with reported volume — usage matching the planner's own optimized estimate settled 753 vs 268 for realistic usage, maxing at the cap just under the implausible line — because the measured baseline prices the *reported* tokens at the frontier rate. Bounded by the reservation cap; anchoring the measured baseline to the planner's estimated baseline tokens would remove the incentive slope entirely. 3. **Boot-time TTS prewarm (low — fixed, `8cadb3cf`).** An operational observation from running the audits: every backend container start eagerly synthesized 13 voice-picker samples through direct provider calls — unconditional spend multiplied by replica count, with an ephemeral sample cache that never absorbed it. The prewarm is now opt-in (env-gated, default off); samples synthesize lazily on first click through the pool-first dispatch path. Post-fix boots make zero provider audio calls, and the settlement audit rerun was unchanged. 4. **Decomposer conjunction-trimming (low — fixed, `0ae56fe7`).** The soft-boundary splitter re-joined non-task clauses with a bare comma, discarding the original conjunction ("compare prices *and* specs" became "prices, specs"). The splitter now captures and restores the original connective, verified live: the browser benchmark goal keeps "prices and specs" intact within one task while genuine task boundaries ("take screenshots…", "produce a comparison table…") still split, and the settlement audit rerun after the change was unchanged — every probe on its bounded basis.

2026-08-12 price-selector audit

A live Elis MCP audit (`9d00ce8899e14ac7bd83951543cadf48`) found a high-severity routing defect after the original benchmark. The shared catalog helpers named `cheapest_model_for_tier`, `all_models_for_tier`, and `cheapest_model_for_capability` ordered candidates by `cost_weight`, which is a coarse tier/quality heuristic, not the resolved token price. For an OpenAI frontier recommendation this selected `o1` at $0.075 per equal 1,000 input plus 1,000 output tokens instead of `o3` at $0.010, an **86.7% cost difference** for that controlled comparison.

Commit `8290b06c` moves all three selectors to one resolved hosted-price sort. The focused catalog and Token Saver suite passed 51 tests, and a restarted live MCP plan recommended `gpt-5.6-luna` for its small-tier audit tasks. This fix does not claim that the cheapest model is always accurate enough; tier and capability selection still happen first, and price only orders eligible models inside that policy boundary. The machine-readable issue record is `docs/research/token-saver-issue-log-2026-08-12.json`.

Official pricing also confirms that hosted tools are a separate cost surface. OpenAI currently lists web search at $10 per 1,000 calls plus search-content tokens and file-search calls at $2.50 per 1,000. Therefore, the study's model token savings must not be read as all-in browser, OpenClaw, or research cost.

Limitations

- Single host session generated both arms; output-length differences partly reflect authoring style per tier rather than measured model behavior. A repeated-run study with independent model calls and automated quality scoring is the required follow-up.
- `o200k_base` approximates Claude tokenization; absolute token counts carry a few percent of error, applied to both arms alike.
- One run per case; the ChatGPT/Claude task-graph difference shows planner variance that a single run cannot bound.
- Excluded usage (planning model, embedding, MCP framing) means end-to-end savings are smaller than visible savings; the plan-side embedding is metered separately by Elis.
- Hosted tool-call tariffs are not included in model-token savings. Research and automation require separate tool-cost subtotals before an all-in claim.
- The accuracy rubric is keyword/structure-based; it verifies constraint compliance and coverage, not depth or factual correctness. A model-judge panel with inter-judge agreement is the follow-up.

Conclusions

1. Model right-sizing is the dominant, reliable saving lever — it survived both hosts, both planners, and both quality regimes.
2. Token-volume savings are achievable but conditional on lean task graphs; decomposition overhead can erase them (ChatGPT run) or conciseness can compound them (this run).
3. Hosts with a wide intra-family price spread (Claude: 10× between Haiku and Fable) benefit most; `suggested_host_model` should always name the cheapest capable model in the required tier, resolved from the live catalog.
4. Quality held: 14 of 15 rubric checks passed on the Elis arm vs 15/15 plain, with the one miss being a keyword artifact, not a substantive gap — the cost saving did not purchase a correctness regression at rubric resolution.
5. Latency is the honest price: ~18 s of planning before the first task. Trivial prompts should bypass the planner; multi-task work absorbs it.
6. Settlement worked end-to-end: reserved on first task, settled on finish, fee-free inside the anonymous allowance — the billing loop this product needs for defensible saved-token claims.
7. The decisive claim — that Elis routing beats *always-cheap*, not just *always-frontier* — remains unproven and is the subject of the next experiment.
8. Automation planning is now benchmark-ready: browser goals plan coherently with correct generic tool mapping, and the two defects this study found — comma-split desktop decomposition and

estimate-settled unexecuted plans — were fixed (`211307f3`) and re-verified against the fixed build.

9. The accounting path now settles catalog-priced same-token cost savings and caps credit at the reservation, but the MCP host report is not yet an attested provider receipt. Attested usage remains the open billing-integrity prerequisite for adversarial paid clients.

Next experiment: the three-arm quality-floor test

The follow-up that would elevate "cheaper" to "better routing":

- **Arms:** (A) always-frontier (host default), (B) always-cheapest capable model, (C) Elis-routed via the Token Saver cursor.
- **Independence:** each arm executed by separate model calls, not authored in one host session, eliminating this study's authorship confound.
- **Quality:** a model-judge panel with rubric-anchored scoring and inter-judge agreement reported, replacing keyword checks.
- **Sample:** ≥10 runs per case per arm, with confidence intervals, capturing planner task-graph variance (which single runs demonstrably cannot).
- **Hypothesis:** on simple/moderate tasks all arms tie and B wins on cost+latency; on complex/deep tasks B degrades measurably while C matches A at a fraction of the cost. If confirmed, Elis's claim becomes *right-sizing with a quality floor* — a property neither naive strategy offers. If B does not degrade, the honest conclusion is that a static cheap default suffices for these workloads and Elis's value concentrates in orchestration, memory, and automation rather than model choice.

Reproduction

Machine-readable record: `docs/research/token-saver-claude-benchmark-2026-08-13.json`. Server: Elis Token Saver MCP (`/api/mcp/claude`), dev-18 build `67143b15` . Companion study: ChatGPT-host benchmark of 2026-08-13 (Codex telemetry stream).