The Context Economy of Agentic LLM Sessions: Where the Money Actually Goes

Evgenii Arsentev, MD, PhD

ORCID: 0000-0002-9120-7298

ARSENTEV.AI — <https://arsentev.ai>

Date: 10 September 2026 (version 1.0); corrected 15 September 2026 (version 1.1)

Version: 1.1 (technical report / preprint). Supersedes version 1.0 (DOI 10.5281/zenodo.22688706); see the Correction section.

License: CC BY 4.0

Correction (2026-09-15)

Version 1.0 of this report (DOI 10.5281/zenodo.22688706) priced the logged tokens with two errors. Both are the author's. They were found on 15 September 2026, when the dollar figures were recomputed against Anthropic's published price list (https://platform.claude.com/docs/en/about-claude/pricing, retrieved 15 September 2026).

1. **Claude Fable 5 cache reads were priced at $0.25 per million tokens instead of $1.00.** Cache reads are billed at 0.1x the input price. The reduced read rate (0.025x, i.e. $0.25) applies only to Claude Fable 5.1, which was introduced on 1 September 2026, and not to Fable 5.
2. **Every cache write was priced at the 5-minute rate (1.25x input).** The logs record the duration of each cache write, and 1-hour writes are billed at 2x input. On machine B, 82.9% of cache-write tokens in the observation window are 1-hour writes and 17.1% are 5-minute writes. On machine A, all cache writes after 15 August 2026 are 1-hour writes; for earlier logs the split cannot be checked.

Token counts, call counts, session counts, the token composition (section 3.2), input amplification (section 3.6) and the deduplication procedure (section 2.4) do not depend on prices and are unchanged. Every dollar figure and every cost share changes. This version assumes that all cache writes are 1-hour writes. The last column shows the alternatives, so that the uncertainty about write duration stays visible.

Quantity	Version 1.0	Version 1.1 (1-hour writes)	Other write-duration assumptions
Context handling / generation, share of cost	83.5% / 16.5%	87.8% / 12.2%	87.6% / 12.4% with machine B's write-duration mix applied to all writes; 86.2% / 13.8% if every write were a 5-minute write
Cache writes, share of cost	27.0%	31.9%	30.5% (mix); 22.6% (all 5-minute)
Modeled total at list rates	$25,790	$34,977	$34,262 (mix); $30,794 (all 5-minute)

Re-reading context / all output, cost ratio	3.42x	4.59x	4.59x (does not depend on write duration)
Claude Opus 5, share of cost	60.5%	49.9%	50.0% (mix); 50.7% (all 5-minute)
Claude Fable 5, share of cost	25.3%	37.8%	37.8% (mix); 37.5% (all 5-minute)
Sessions carrying 80% of cost	24 of 722 (3.3%)	about 21 of 722 (2.9%), estimate	about 22 (3.0%) if all 5-minute
Cost share of sessions over 200 calls (8.0% of sessions)	92.2%	about 92.8%, estimate	about 92.8%
Median / mean session cost	$1.92 / $35.72	about $2.30 (estimate) / $48.44	about $2.01 / $42.65 if all 5-minute

Why the session-level figures are estimates. The released dataset (runs_final.json) records tokens per session but not the model, and the correction changes the price of two models' tokens only. 446 of 722 sessions (23.9% of the version 1.0 cost) could be assigned to one model exactly, because only one model's price vector reproduces their version 1.0 cost. The remaining sessions were re-priced with corpus-wide per-class ratios. The concentration figures (sections 3.4 and 3.5), Figures 2 and 3 and the median are therefore estimates. The estimated session costs sum to $33,490, about 4% below the exact corpus total, which indicates the size of the estimation error. The corpus totals and shares (sections 3.3 and 3.7) are computed exactly from the per-model aggregate, given the assumption about write duration.

What the correction changes in the conclusions. The direction of the findings does not change. Context handling remains the dominant cost, and its share rises under every write-duration assumption (86.2% to 87.8%, against 83.5% in version 1.0). Cost is at least as concentrated as version 1.0 reported. The size of the corrected figures depends on how long cache writes were kept, which cannot be verified for machine A before 15 August 2026; the plausible range is given above. The recomputation scripts and their outputs are released with this version (recompute-2026-09-15.zip, session_costs_recomputed.json).

Abstract

Agentic coding assistants are usually reasoned about as *generators*: the mental model is that the model writes code, and the code it writes is what you pay for. This report measures that assumption against a complete local log corpus of one practitioner's agentic work — 722 sessions, 150,902 model calls, 34.56 billion tokens — and finds that the assumption is close to inverted.

Under published per-token API rates, 87.8% of the modeled spend is context handling (re-reading cached conversation state and writing new state into the cache) and 12.2% is generation of new text. Re-reading context alone costs 4.59x as much as everything the models produced. Token counts

are more extreme still: 97.05% of all tokens billed are cache reads, and only 0.38% are output.

Cost is also highly concentrated. By estimate, eighty percent of the total falls on 21 of 722 sessions (2.9%),

and sessions longer than 200 model calls — 8% of sessions — account for an estimated 92.8% of spend. A derived

quantity, *input amplification* (total input tokens billed in a session divided by that session's

peak context size), has a median of 23.7x and a 90th percentile of 124.5x (linear interpolation between order statistics): the same working state

is paid for dozens of times over the life of a session.

This is a single-practitioner case study, not a sample of a population, and the dollar figures are

a model applied to logs rather than an invoice. The contribution is a reproducible measurement

procedure — including two non-obvious log-deduplication traps that change the answer by roughly a

factor of two in each direction — and an aggregated, de-identified dataset released alongside the

report.

Keywords: large language models, agentic systems, prompt caching, cost analysis, telemetry,

context window, software engineering economics

---

1. Introduction

When practitioners discuss the cost of an LLM coding agent, the discussion is almost always about

generation: which model writes the patch, how many tokens the patch is, whether a cheaper model

could have written it. Budget conversations follow the same shape — "use the small model for easy

tasks" — because generation is the visible part of the interaction. It is what appears on screen.

But an agent session is not a sequence of independent generations. It is a loop. At every step the

model is re-sent the accumulated state of the session: the system prompt, the standing instruction

files, the tool definitions, every previous tool call, and every previous tool result. A session of

500 steps does not send its context once; it sends a growing prefix of it 500 times. Prompt caching

reduces the unit price of that resend substantially, but it does not reduce the count. The resend is

still billed, per step, at cache-read rates.

The consequence is that the dominant cost driver in a long agentic session is not what the model

writes but how many times the session's own history is pushed back through the model. This is

structurally invisible in day-to-day use:

- **It is not on screen.** The user sees the assistant's replies; the re-sent prefix is rendered nowhere.
- **The unit price is small.** Cache reads are cheap per token, which is precisely why their volume grows unnoticed until the multiplier dominates.
- **Subscription pricing hides it.** Under a flat plan there is no per-session invoice to inspect, so the internal distribution of cost is never surfaced.
- **Sub-agents multiply it.** Delegated sub-agent runs carry their own contexts, and in this corpus they are the majority of all model calls (52%, 77,971 of 150,902).

This report asks a narrow empirical question: **in a real, uncurated body of agentic work, what fraction of token spend is context handling versus generation, and how is that spend distributed across sessions?** It does not attempt to answer whether the context spend is *avoidable*, which is a question about counterfactual workflows that observational logs cannot settle.

---

2. Method

2.1 Data source

The corpus consists of the local session logs written by a command-line agentic coding assistant during ordinary work. Each session is stored as a newline-delimited JSON transcript; every record corresponding to an assistant turn carries a usage object reporting, for that model call: fresh (uncached) input tokens, cache-read input tokens, cache-creation input tokens, and output tokens, plus the model identifier.

Nothing beyond these counters and identifiers is used in the analysis. Session content — prompts, file contents, tool arguments, model replies — is neither analysed nor published.

Two machines contributed logs, and they represent two *different operating regimes* rather than two samples of the same regime:

Machine	Observation window	Character
A	26 Jun 2026 - 10 Sep 2026	Interactive, human-in-the-loop work
B	31 Aug 2026 - 8 Sep 2026	Predominantly long unattended batch runs

This asymmetry is deliberate — both regimes are part of the practice being described — but it must be carried through every reading of the results. See §5.4.

2.2 Unit of analysis

The unit is the **session**: one continuous agent run with its own identifier. Within a session we distinguish **main-thread calls** from **sub-agent calls** (delegated runs spawned by the main thread and logged under the same session). Both are counted as model calls; the split is retained because sub-agent runs carry independent contexts and therefore independent amplification.
Session identifiers were replaced with truncated hashes before the dataset was written out. No mapping from hash to original identifier is published.

2.3 Cost model

Token counts were converted to dollars using **published per-token API list rates for each model, by token class** (fresh input, cache read, cache write, output), summed per session. This is a *modeled* cost — see §5.3 — and the four token classes are reported separately so that the mapping from tokens to dollars stays inspectable and can be recomputed under different rate assumptions from the released dataset.
Cost is attributed to four categories, which map one-to-one onto the token classes:

- **Re-reading context** — cache-read input tokens.
- **Writing context to cache** — cache-creation input tokens.
- **Generating new text** — output tokens.
- **Fresh input** — uncached input tokens.

"Context handling" is the sum of the first, second and fourth; "generation" is the third.

2.4 Deduplication: two traps that change the answer by ~2x in each direction

This is the part of the procedure most likely to be got wrong, and the reason that naive numbers circulating in practitioner discussions are unreliable in both directions.
The logs are streaming transcripts, not a ledger of billed calls. A single model call is frequently written to disk as *several* records sharing the same request identifier: as the response streams, the assistant message is re-serialised, and each serialisation carries a usage object.
Two facts about those repeated usage objects matter.

- **Input-side counters are repeated, not incremental.** The cache-read and cache-creation figures

are the same numbers restated on each record for that call. Summing every record therefore *double-counts input*. In this corpus, the naive sum over all records overstates total cost by a factor of **1.90** (65.92 bn tokens naively summed against 34.72 bn after correct merging).

- `output_tokens` **is a cumulative snapshot of the stream, not a per-record delta.** The value grows across the records belonging to one call and reaches its true value only on the last one.

The obvious correction for trap (1) — deduplicate by request identifier and keep the *first* record — therefore truncates generation: it **understates output by a factor of 1.82** across the corpus (exactly 2.00 on the larger of the two machines).

Neither naive approach is safe. Summing inflates the total; first-record deduplication deflates generation, which is exactly the quantity the headline result is about — so that error moves the context-versus-generation ratio in the direction of this report's own conclusion, and would flatter it.

The correct procedure is an element-wise maximum. Group records by `requestId` (falling back to `message.id` where the request identifier is absent) and, for each token class, take the maximum observed value across the group rather than the sum or the first value. This is correct for both classes at once: repeated input counters are idempotent under `max`, and a cumulative output counter reaches its final value at its maximum. One group is then exactly one billed model call.

Symlinked log directories. The log root contained symlinked directories, so a directory walk that follows links enumerates the same session file more than once under different paths. Because deduplication here is keyed on request identifiers rather than on file paths, those duplicates collapse correctly — but any pipeline that deduplicates *per file* and then sums across files will double-count entire sessions. Traversal must either not follow symlinks or must canonicalise real paths before reading.

2.5 Verification

The pipeline was implemented twice, independently (different language, different traversal and grouping code), and the two totals were compared. Agreement was within **0.02%** on total cost. The residual is attributable to floating-point accumulation order and to a small number of records with malformed usage objects handled slightly differently by the two implementations. All aggregates in §3 were recomputed directly from the released dataset.

2.6 Derived measure: input amplification

For each session, **input amplification** is defined as

> (fresh input + cache-read + cache-write tokens) / peak context size observed in that session

That is: how many times over the session's largest working state was paid for. A session that sent its context exactly once would score 1x. Sessions with fewer than three model calls are excluded, because the ratio is dominated by start-up effects and is not meaningful there; this leaves **n = 590** sessions of 722.

Amplification is a ratio of billed input volume to peak state size. It is not a measure of waste: a long session *must* re-send its state in order to keep working. It quantifies the multiplier, not its justification.

---

3. Results

3.1 Corpus

Quantity	Value
Sessions	722
Model calls	150,902
— of which sub-agent calls	77,971 (52%)
Total tokens billed	34.56 billion
Modeled cost at API list rates	$34,977
Median session cost	about $2.30 (estimate)
Mean session cost	$48.44

The gap between the median (about $2.30, an estimate) and the mean ($48.44) — a factor of about 21 — is the first indication that the distribution is not merely skewed but concentrated in a small tail. Typical sessions are inexpensive; the corpus total is not made of typical sessions.

3.2 Token composition

Token class	Share of all tokens
Cache read (re-reading context)	97.05%

Cache write (storing context)	2.57%
Output (generation)	0.38%
Fresh input	0.01%

For every token the models produced, roughly 256 tokens of cached context were read back in. The token mix of an agentic workload is not a conversation; it is a repeated scan of accumulated state with a thin margin of new text on top.

3.3 Cost composition (Figure 1)

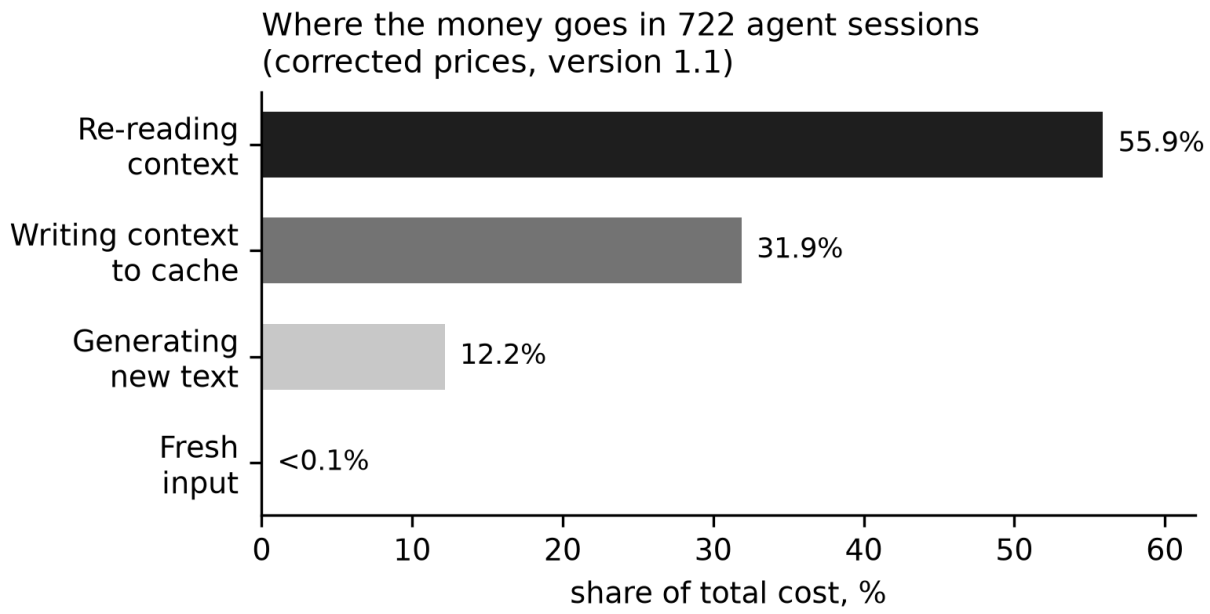

Figure 1. Share of total modeled cost by category, 722 sessions, at corrected prices (version 1.1).

Category	Share of cost
Re-reading context	55.89%
Writing context to cache	31.89%
Generating new text	12.18%
Fresh input	0.04%

Cache pricing compresses the extreme token ratio of §3.2 into a less extreme but still decisive cost ratio: **context handling 87.8%, generation 12.2%**. Re-reading context alone is **4.59x** the cost of everything the models wrote.

Two secondary observations from the same split:

- **Cache writing is not a rounding error.** At 31.89% of cost it is the second-largest line — larger

than generation. Caching does not make context free; it shifts part of the price to the moment

state is stored.

- **Fresh input is negligible** (0.04%). Under a cached agentic loop, what the human types is

economically invisible. The cost is the machine re-reading its own history.

3.4 Concentration (Figure 2)

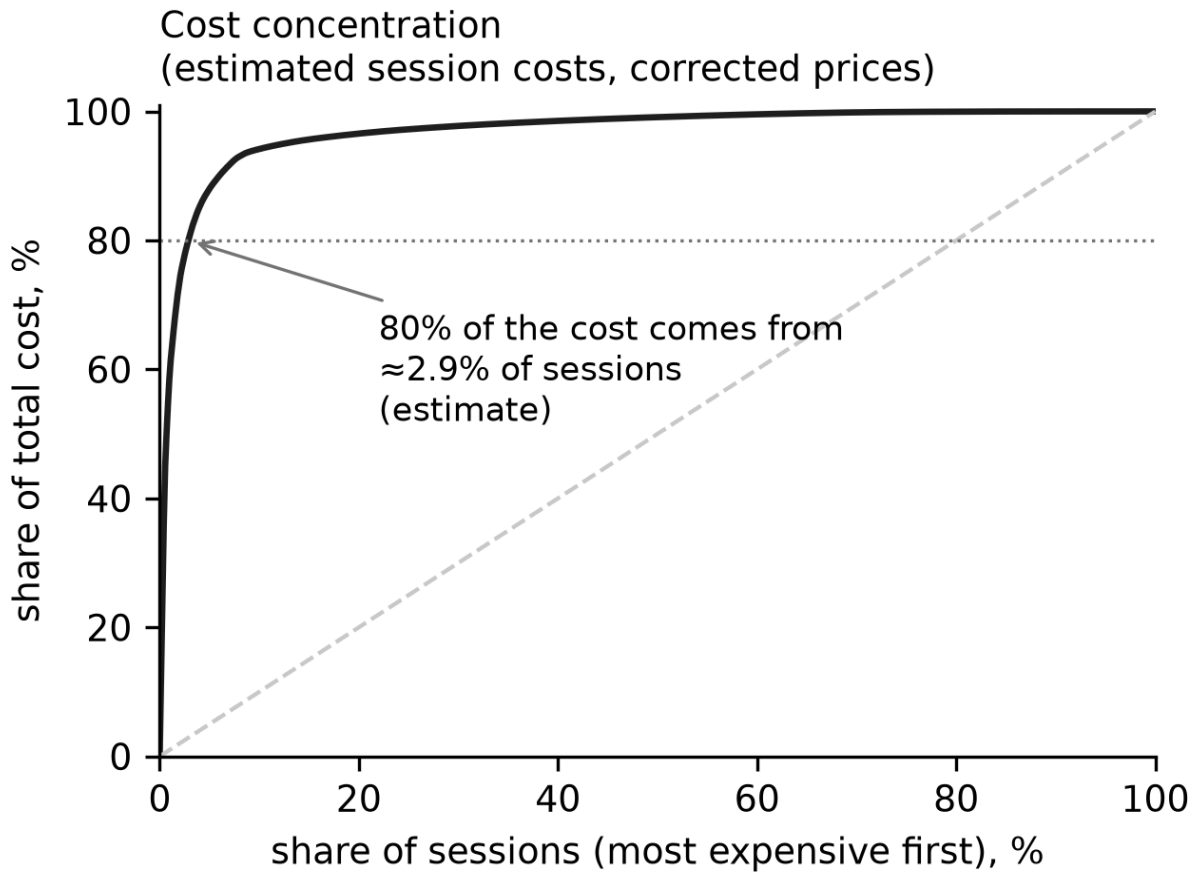

Figure 2. Cumulative share of total cost (estimated session costs at corrected prices, version 1.1) against share of sessions, most expensive first. The

diagonal marks a uniform distribution.

- **80% of total cost comes from an estimated 21 sessions out of 722 — 2.9%.**
- By estimate, the single most expensive session is 12.5% of the corpus total and the ten most expensive are 65.8%.
- The cheaper half of all sessions (361 sessions) accounts for **0.89% (estimate)** of total cost.

The distribution is close to degenerate. In budget terms, the cheaper half of this practitioner's

sessions is not worth optimising at all; the question is entirely about what happens in the top few percent.

3.5 Session length and cost (Figure 3)

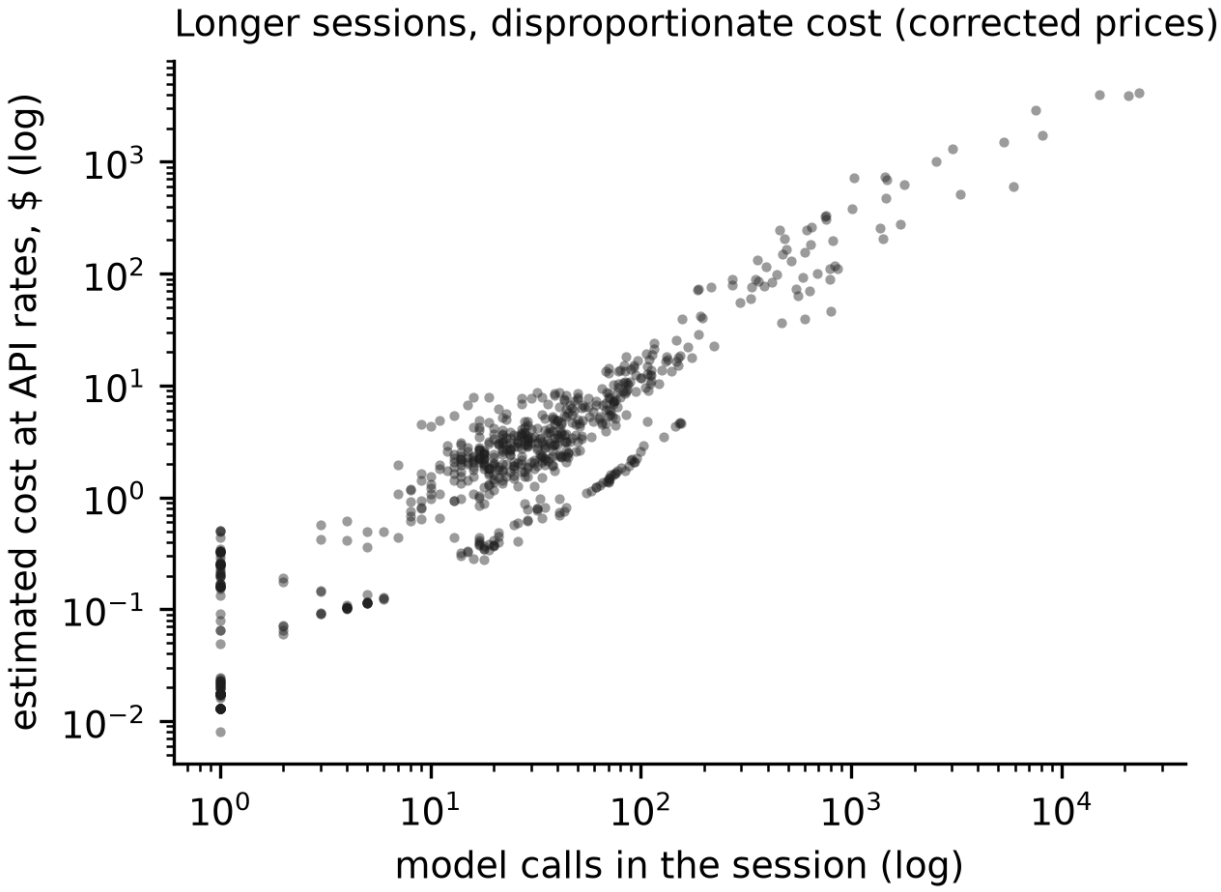

Figure 3. Estimated modeled cost at corrected prices (version 1.1) against model calls per session, both axes logarithmic. Each point is one session.

Sessions with more than 200 model calls are **8.0% of sessions and an estimated 92.8% of cost**.

On log-log axes the relationship is visibly steeper than proportional: cost per call is not constant but rises with session length, which is the expected signature of a growing re-sent prefix. Two structural features of the scatter are worth naming, since both are artifacts of how agent sessions work rather than noise:

- The **vertical stripe at one call** — single-call sessions spanning two orders of magnitude of cost — is the start-up cost of a session, dominated by how much instruction and tool material is loaded before any work happens.

- The **lower band** running parallel to the main cloud is sessions with unusually low cost for their call count. Cheap-model calls and short-context sub-agent calls both land there.

The correlation between length and cost is not, on its own, evidence that length *causes* disproportionate cost: long sessions are also the sessions given hard, sprawling tasks, and task difficulty is not observed in these logs.

3.6 Input amplification (Figure 4)

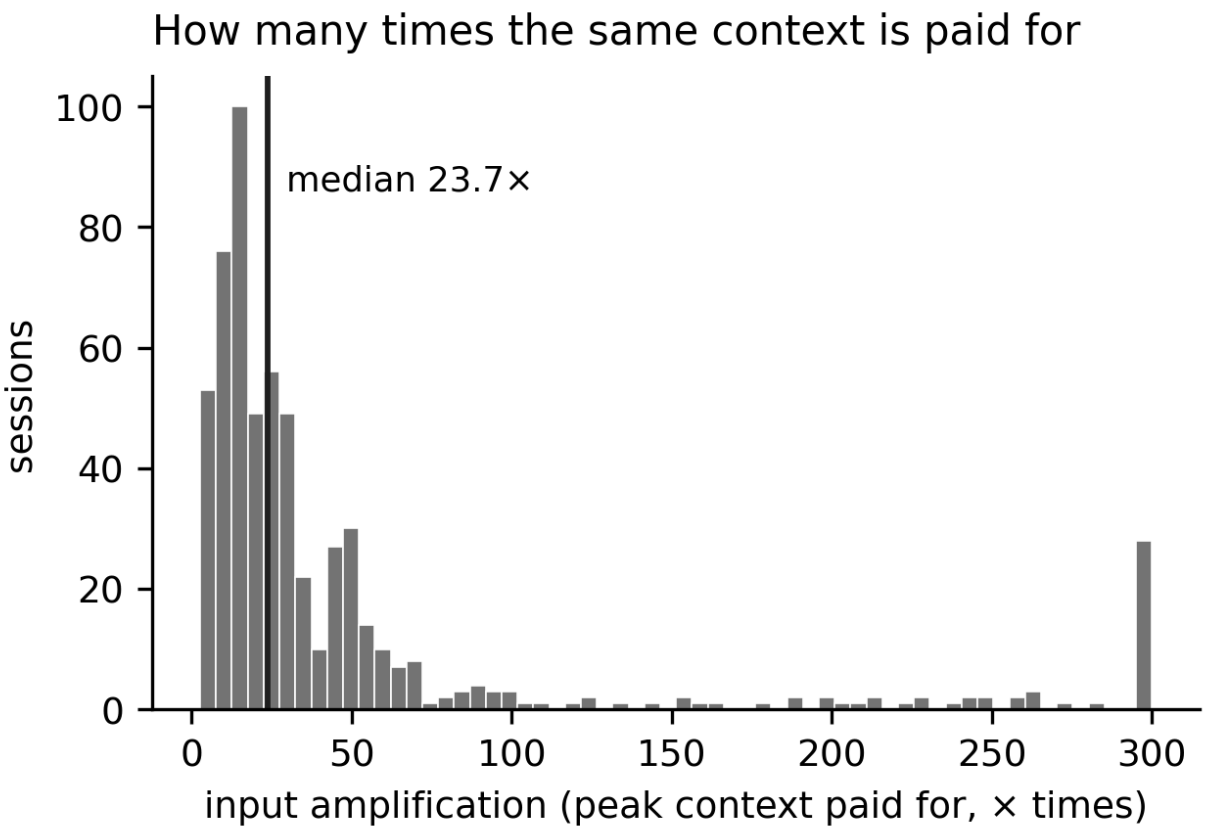

Figure 4. Distribution of input amplification across 590 sessions of at least three model calls. The rightmost bar is an overflow bin collecting all sessions above the axis limit.

Statistic	Value
Median	23.7x
90th percentile	124.5x
Maximum	4,182x
n	590 sessions

The median session pays for its peak working state roughly **24 times**. The distribution has a long

right tail: one session in ten pays for it more than 125 times, and the extreme case exceeds four thousand.

Amplification is the mechanism behind Figures 2 and 3. A session's cost is not primarily set by the size of its context, nor by the amount of output; it is set by the product of context size and the number of times that context is traversed. Sub-agents contribute to the tail — a delegated run maintains its own context and its own multiplier — and sub-agent calls are the majority of calls in this corpus (§3.1).

3.7 Model mix

Model	Calls	Share of modeled cost
Claude Opus 5	87,583	49.9%
Claude Fable 5	37,997	37.8%
Claude Opus 4.8	9,857	9.1%
Claude Fable 5.1	2,069	1.9%
Claude Sonnet 5	13,000	1.3%
Claude Haiku 4.5	388	< 0.1%
Claude Sonnet 4.6	8	< 0.1%

Models are named because the price card is the input to the cost model, and a reader cannot check the arithmetic without knowing which rates were applied. A per-model aggregate — calls and the four token counters for each model — is released as `per_model_aggregate.json` so that §3.3 and §3.7 are recomputable from the released data rather than taken on trust.

Note the ordering: Sonnet 5 accounts for more calls than Opus 4.8 but a seventh of its cost. Cheap models absorb a large share of the *work* while contributing little to the *bill*, which is the practical form of the cost concentration reported in §3.4.

Because the cost split of §3.3 holds across the corpus while three models account for 96.8% of it, the context-versus-generation ratio reported here is a property of the *workload shape*, not of any single model's price card. The absolute dollar figure would move under a different model mix; the ratio is far less sensitive to it.

---|---|

Frontier model, current generation	**49.9%**

Mid-tier model	37.8%
Frontier model, previous generation	9.1%
All others	< 2% each

Because the cost split of §3.3 holds across the corpus while three model classes account for 96.8% of it, the context-versus-generation ratio reported here is a property of the *workload shape*, not of any single model's price card. The absolute dollar figure would move under a different model mix; the ratio is far less sensitive to it.

---

4. What these numbers do and do not mean

Stated plainly, with the boundaries of the claim attached:

- **In this corpus, agentic spend is a context-handling cost, not a generation cost** — 87.8% versus 12.2%. This is a measurement of one workload, not a law.
- **Cost is concentrated to the point where averages mislead.** Median session about $2.30 (estimate), mean $48.44, 80% of the total in an estimated 2.9% of sessions.
- **Session length is the strongest observed correlate of cost.** Whether length is a cause, a proxy for task difficulty, or both, cannot be determined from observational logs.
- **Caching changes the price of context, not its volume.** Cache writes alone (31.9%) exceed generation (12.2%).

What is *not* shown: that any of this spend was avoidable, that shorter sessions would have done the same work, or that any particular workflow change would have reduced the total. Those are claims about counterfactuals, and this study observed a workflow rather than manipulating one.

---

5. Threats to validity

This section is deliberately long, because the headline numbers are striking and the design is weak in ways a reader should not have to reconstruct.

5.1 One practitioner, one toolchain — a case study, not a sample

All 722 sessions come from **one person**, working with **one agentic CLI**, under **one set of personal working habits and standing instruction files**. This is an n-of-1 case study. There is no sampling frame, no random selection, and no second practitioner to contrast against.

The measured ratios depend on things that vary enormously between practitioners: how much standing instruction material is loaded into every session, how aggressively work is delegated to sub-agents, how long sessions run before being restarted fresh, how large the files being read are, and how often the agent re-reads rather than remembers. A practitioner with short sessions and thin instruction files would report a different split, plausibly a much less extreme one. **None of the percentages here should be treated as an industry figure.**

What generalises, if anything, is the *shape* of the accounting — four token classes, an amplification multiplier, a concentrated tail — and the measurement procedure, not the values.

5.2 Observational, not experimental

No condition was manipulated, randomised or held constant. Sessions were long or short, delegated or not, expensive or cheap, because of what the work demanded that day. Every association reported here — length with cost, sub-agent use with amplification — is a correlation within a single practitioner's uncontrolled workflow, and is confounded at minimum by task difficulty, which is not recorded in the logs.

Specifically: **no claim is made that ending sessions earlier, delegating less, or trimming context would have reduced spend.** Testing that requires an intervention study — the same tasks executed under different session policies — which this is not.

5.3 Cost is a model, not an invoice

The dollar figures were produced by applying **current published per-token API list rates** to logged token counts. They are not a bill and were not reconciled against one. Consequences:

- The work was performed under subscription plans, not metered API billing. **$34,977 is what this workload would have cost at list rates**, not what was paid.
- Rates change. The figure is anchored to the rate card in effect at the date of writing, and the ratio between cost categories shifts whenever the ratio between cache-read, cache-write and output prices shifts. Anyone re-running this analysis at a later date will get a different split from the same tokens.

- Discounts, batch pricing, negotiated rates and free-tier effects are not modeled. Version 1.0 also applied two wrong rates, for Fable 5 cache reads and for cache-write duration; see the Correction section.

The **token** measurements (§3.2) are direct and rate-independent; the **dollar** measurements (§3.3) inherit every assumption in the rate card.

5.4 Two operating regimes inside one window

Machine A (26 Jun - 10 Sep 2026) is interactive, human-in-the-loop work. Machine B (31 Aug - 8 Sep 2026) is predominantly long unattended batch runs and contributes only nine days of data. These are different workloads pooled into one corpus.

Long unattended runs are exactly the sessions that produce high call counts and high amplification, so machine B is over-represented in the tail that drives §3.4-§3.6, while machine A dominates the session count that drives the median. **The corpus is therefore a mixture, and neither the mean nor the median describes a single coherent regime.** The pooled figures should be read as "this practice, over this window", not as a characterisation of interactive agentic coding.

The windows also do not coincide. Any change over time — in tooling defaults, in model availability, in caching behaviour, in the practitioner's own habits — is entangled with the machine split and cannot be separated from it.

5.5 Long-context surcharges are not modeled

Some providers price requests above a context threshold at a premium. The cost model applies a single per-token rate per class per model and **does not apply any long-context multiplier**. Because the most expensive sessions in this corpus are precisely the ones running at very large context sizes, this omission biases the estimate **downward**, and does so hardest in the tail that dominates the total. The true list-rate cost of the top sessions is likely higher than reported here; by how much depends on threshold and multiplier assumptions that were deliberately not invented.

5.6 Only what was on disk

The corpus is the set of session logs present on the two machines at the time of collection. Logs deleted, rotated, truncated or never written are absent, and there is no independent record against which completeness could be checked. Work done on other machines, in other tools, or through web interfaces is not included. The corpus is a convenience sample of surviving artifacts.

Records with malformed or missing usage objects were skipped rather than imputed; they are a small

fraction, but they are a small *unmeasured* fraction.

5.7 Measurement-definition risks

- **Amplification depends on "peak context"** as recorded in the logs, which is the largest observed context for any single call in that session. For sessions mixing a large main thread with many small sub-agent contexts, dividing total input by the *single largest* context understates the multiplier relative to a per-thread calculation. The measure is a session-level summary, and different reasonable definitions give different numbers.
- **The three-call threshold** for including a session in the amplification analysis is a judgement call, not a derived cutoff. It excludes 132 of 722 sessions.
- **The 200-call threshold** in §3.5 is likewise a chosen round number, selected for legibility after seeing the distribution. It is descriptive, not a fitted breakpoint.
- **The deduplication procedure is inferred from log structure**, not from provider documentation of billing. The element-wise maximum is the reading of the records that is internally consistent across both token classes and that two independent implementations agree on — but it is an inference about what was billed, and it has not been validated against a provider invoice.

5.8 Sub-agent attribution

Sub-agent calls are attributed to the session that spawned them. This is the right unit for asking "what did this piece of work cost", but it means a session's cost is not a property of its visible transcript length. Comparisons against tooling that reports only main-thread usage will not match.

---

6. Practical implications

Offered as engineering observations that follow from the measurements, with the causal caveat of §5.2 applying throughout. None of these is demonstrated to save money; each is a place where these numbers suggest looking.

Measure the four token classes separately. Any cost dashboard that reports a single "tokens" figure, or that reports only input and output, cannot distinguish the 87.8% from the 12.2%. Cache-read and cache-write have to be separate columns or the dominant term is invisible.

Report cost distributions, not averages. With a median of about $2.30 (estimate) and a mean of $48.44, a mean is

an actively misleading summary. Percentiles and a top-N list are the useful views. Instrumentation that flags per-session cost will be more informative than instrumentation that tracks a running average.

If cost matters, the tail is where it lives. The cheaper half of sessions is under 1% of total spend. Effort spent shaving cost from routine sessions is, in this corpus, effort spent on a rounding error.

Model choice is one lever among several, and not obviously the largest. Choosing a cheaper model changes the per-token price across all four classes, but 87.8% of spend is incurred on context that a session carries regardless of which model reads it. The size of that context, and the number of times it is traversed, are separate levers.

Standing context is paid for repeatedly, not once. Material loaded into every session — global instruction files, tool definitions, boilerplate preamble — enters the cache-read term and is then re-read at every step. Its cost scales with session length, not with how often it is useful. That is a reason to know the size of such material; it is not, by itself, a reason to delete it, since its benefit is not measured here.

Sub-agent delegation has an accounting consequence. Each delegated run carries its own context and its own multiplier. Delegation may well be worth it — parallelism, isolation, keeping the main thread small — but in this corpus it is where the majority of calls live, and any cost model that counts only the main thread will be wrong by a large factor.

Instrument before optimising. The most transferable finding is §2.4: two plausible ways of reading these logs give answers differing by roughly 2x in opposite directions. A team that has not verified its deduplication is not measuring what it thinks it is measuring.

---

7. Data availability

An **aggregated, de-identified dataset** is published with this report as `runs_final.json`. It contains **one record per session**, with the following fields:

Field	Meaning

`sid`	Truncated hash of the session identifier (not reversible; no mapping published)
`steps`	Main-thread model calls
`side_steps`	Sub-agent model calls
`all`	Total model calls (`steps` + `side_steps`)
`input`	Fresh (uncached) input tokens
`cache_read`	Cache-read input tokens
`cache_write`	Cache-creation input tokens
`output`	Output tokens
`max_ctx`	Peak context size observed in the session, tokens
`cost`	Modeled cost in USD at the rates applied in version 1.0, left uncorrected in this file; corrected estimates are in session_costs_recomputed.json

The dataset contains no session content. No prompts, no user messages, no model output, no file paths, no file contents, no tool arguments, no repository or project identifiers, no hostnames, and no per-record timestamps. It is counters only. Every aggregate in §3 is recomputable from this file, with the exception of the per-model breakdown in §3.7, which was computed before the per-model dimension was collapsed during aggregation. Since version 1.1, session costs at the corrected rates can only be estimated, because this file does not record the model of each session. The estimates and the recomputation scripts, including a counters-only log scanner, are released as session_costs_recomputed.json and recompute-2026-09-15.zip.

Figures 1-4 are published alongside the report as PNG files.

The processing code is not published, because it is coupled to a local log layout and directory structure. §2.4 describes the procedure in enough detail to reimplement it: the element-wise maximum over records grouped by `requestId` / `message.id`, and canonicalisation of paths before traversal.

---

8. Conclusion

Across 722 agentic sessions and 34.56 billion tokens, 87.8% of the modeled spend went to handling context and 12.2% to generating text, with re-reading context alone costing 4.59x as much as all output. Eighty percent of the total fell on an estimated 2.9% of sessions, and the median session paid for its own peak working state 23.7 times over.

These are measurements of one practitioner's workflow, with one toolchain, over one period, priced by

a model rather than by an invoice, and they should not be read as general constants. What they do establish is that "how much did the model write?" is the wrong accounting question for agentic workloads, that the right accounting has at least four token classes rather than two, and that the two most obvious ways of reading agent logs each get the answer wrong by about a factor of two — in opposite directions.

Citation

> Arsentev, E. (2026). *The Context Economy of Agentic LLM Sessions: Where the Money Actually Goes.*

> Technical report, version 1.1 (corrected 15 September 2026). Zenodo. DOI 10.5281/zenodo.22688704 (all versions). ORCID: 0000-0002-9120-7298.

Corresponding author: Evgenii Arsentev, MD, PhD — ARSENTEV.AI — <https://arsentev.ai>