Clear Every Third Task: A Measured U-Curve in the Context Economy of Coding Agents

Evgenii Arsentev
ARSENTEV.AI, Barcelona, Spain
hello@arsentev.ai · ORCID 0000-0002-9120-7298 · https://arsentev.ai

Abstract

Practitioners running LLM coding agents face a question with no published answer: how often should an agent's context be cleared? We ran a fixed suite of twelve programming tasks under six session-length policies — a fresh session every task, every two, three, four, six, and one session for all twelve — with six replicates each, holding the model, the tasks, their order, and the verification suite constant. Cost is not monotone in session length. It falls from $2.68 per run when the context is cleared after every task to $2.14 at every third task, then rises again to $2.47 when the context is never cleared. Exact permutation tests show that clearing every third task is significantly cheaper than clearing after every task ($p = 0.0022$, the smallest value this design can produce) and than never clearing ($p = 0.0108$), while the differences between clearing every three, four and six tasks are within noise ($p = 0.10$ and $p = 0.21$). The optimum is therefore a plateau, not a point, and both of the policies teams actually use sit outside it. Task quality was identical everywhere: 4086 tests, zero failures, in every condition. The U-shape decomposes cleanly. Cache writes fall monotonically with longer sessions (253k → 88k tokens) because each fresh session re-primes the cache, while context carried per model call rises monotonically (38k → 71k tokens) because nothing is ever dropped. Their product, cache reads, is U-shaped with a minimum at three. Because cache writes are priced at 12.5× cache reads per token on the model we used, neither term can be ignored. We release the run-level token counters and the harness.

Keywords: LLM agents, inference economics, prompt caching, context management, reproducible measurement

1 Introduction

Every team running coding agents in production develops a folk rule about session hygiene. Some clear the context after every task, reasoning that a short prompt is a cheap prompt. Others never clear it, reasoning that the model should keep what it has learned about the codebase. Both rules are defended with confidence and neither, to our knowledge, has been measured.

The question is not academic. In agent workloads the dominant cost is not the tokens the model writes but the tokens it re-reads. In our runs, output accounts for roughly a third of spend and cached context for the rest. Session length is the one lever that changes how much context is re-read, and it is a lever every operator already has: it costs nothing to change and requires no model access.

We measured it. Holding everything else fixed, we varied only how many tasks share a session, and we report what happened to cost, to the structure of that cost, and to whether the work still got done.

Our contributions:

1. A controlled measurement showing that agent cost is **U-shaped** in session length, with an interior optimum that beats both intuitive policies at conventional significance.
2. The finding that the optimum is a **plateau rather than a point**: clearing every three, four or six tasks is statistically indistinguishable, while clearing every task and never clearing are both significantly worse.
3. A **decomposition** of that U into two monotone components pulling in opposite directions, which explains why the optimum is interior and why it is not obvious.
4. An **open dataset** of run-level token counters and the harness needed to reproduce the design on another model or task suite.

2 Setup

Task suite. Twelve independent programming tasks that build a small JavaScript utility package. Each task is stated in one sentence and each is verified by unit tests executed with `node --test`. Tasks and their order are identical in every run; nothing about the suite depends on the session policy.

Agent. The Claude Code CLI in non-interactive mode, one invocation per task. Continuation within a session uses the CLI's continue flag; a new session is started simply by omitting it. This is the same mechanism an operator would use in production, not a research harness that simulates clearing.

Conditions. Six session lengths: 1, 2, 3, 4, 6 and 12 tasks per session, giving 12, 6, 4, 3, 2 and 1 sessions per run respectively. Six replicates per condition, 36 runs, 432 task executions in total.

Model. A single model held fixed across all conditions. We deliberately do not compare models here: the claim is about the shape of the curve, and mixing models would confound it.

Measurement. Token counters are read per model call and aggregated per run. Two details matter for correctness. First, records must be de-duplicated by message identity, because streaming produces multiple records for the same call. Second, when merging duplicates the element-wise maximum must be taken across usage fields: the first record of a streaming call carries an incomplete output count, and taking the first rather than the maximum understates output by roughly a factor of two. We flag this because it is easy to get wrong and it silently biases exactly the quantity under study.

Cost. We report a modeled cost computed from the measured counters and the model's published per-token prices, separating input, output, cache write and cache read. We report modeled rather than billed cost so the numbers can be recomputed from the released counters under any price list.

Statistics. With six replicates per condition we use exact two-sided permutation tests on the difference of means, enumerating all 924 splits. This is assumption-free about the shape of the cost distribution, which matters because agent runs are not normally distributed. The smallest two-sided p this design can produce is 2/924 = 0.0022.

Quality control. Every run ends with the full test suite. A cost comparison is meaningless if the cheaper policy also does less work, so we treat the test outcome as a gate, not as a secondary metric.

3 Results

Table 1 gives the mean over six replicates for each condition.

Tasks per session	Sessions	Model calls	Cache write (tok)	Cache read (tok)	Context per call (tok)	Cost (USD)
1	12	94.0	252,533	3,622,795	38,478	**2.680**
2	6	84.3	169,670	3,578,718	42,415	2.399
3	4	71.5	131,100	3,160,517	44,192	**2.138**
4	3	71.0	123,478	3,466,808	48,795	2.279
6	2	71.0	105,746	3,796,905	53,309	2.281
12	1	63.0	87,846	4,451,367	70,712	2.473

Table 1: Means over six replicates. Cost is modeled from measured counters.

The curve has an interior minimum. Cost falls by 20.2% going from clearing every task to clearing every third, then rises again by 15.6% going from every third to never. The worst policy is the one many teams adopt by default — a fresh session for every task — and it costs 25.3% more than the best.

The extremes are significantly worse; the middle is a plateau. Clearing every third task beats clearing after every task with p = 0.0022, the smallest value 6-versus-6 permutation can yield, and every replicate at the optimum was cheaper than every replicate at the worst condition. It also beats never clearing (p = 0.0108) and clearing every second task (p = 0.0108). But against clearing every fourth or sixth task the difference does not clear conventional thresholds (p = 0.097 and p = 0.212). The honest reading is that anything from three to six tasks per session sits in the same flat bottom, and the two policies people actually use — one task per session, or one session for everything — both sit outside it.

Quality is flat. Across all 36 runs, 4086 tests executed and 4086 passed. Not a single condition traded correctness for cost. Wall-clock time showed no consistent trend with session length either: condition means span 482–543 s while individual runs span 409–690 s, so replicate variance exceeds the between-condition spread and the cost differences are not latency differences in disguise.

The saving is not marginal. A 25% gap on a lever that costs nothing to pull is larger than most prompt-level optimizations report, and it compounds: the same policy applies to every agent run a team makes.

4 Why the curve is U-shaped

The U is the product of two monotone effects that pull in opposite directions.

Longer sessions mean fewer model calls. Calls fall from 94.0 to 63.0 as session length grows. A fresh session has to rediscover the state of the working directory, so short sessions spend calls on orientation that long sessions do not.

Longer sessions mean more context per call. Context carried per call rises from 38,478 to 70,712 tokens. Nothing is ever dropped inside a session, so every task's transcript is paid for again on every subsequent call in that session. A token that entered the context on the fifteenth call of a hundred-call run is charged roughly eighty-five more times.

Cache reads are the product of these two terms, and a product of a falling and a rising factor is U-shaped: 3.62M tokens at every-task clearing, 3.16M at the optimum, 4.45M with no clearing at all.

Clearing is not free either. Cache writes fall monotonically with session length, from 252,533 to 87,846 tokens, because every fresh session must re-prime the cache. On the model we used, a cache write costs 12.5× a cache read per token ($3.75 versus $0.30 per million). That ratio is what keeps aggressive clearing expensive even though it does reduce context per call: at every-task clearing, cache writes alone account for $0.95 of the $2.68 run.

So the two ends of the range are expensive for different reasons, which is precisely why the failure is hard to notice in practice. A team that clears constantly sees a small context and concludes it is being frugal; it is paying in cache writes. A team that never clears sees no re-priming and concludes it is being frugal; it is paying in context re-read. Neither sees the other's bill.

At the optimum the spend splits roughly 44% cache read, 23% cache write, 33% output — no single term dominates, which is the signature of a balanced point rather than a corner solution.

5 Threats to validity

Resolution of the plateau. Six replicates are enough to separate the extremes from the middle but not to rank conditions inside the middle. Distinguishing three from four or six tasks per session would need either more replicates or a task suite with a larger effect, and we do not claim three is better than four.

One model, one suite. The location of the minimum almost certainly depends on the cache write to cache read price ratio and on how much per-session orientation a task suite demands. We claim the U-shape and the mechanism, not that three is a universal constant. The released harness is designed to let others place their own minimum.

Modeled cost. We compute cost from counters and published prices rather than from invoices. This is a deliberate trade: it makes the numbers recomputable under any price list, at the cost of not capturing billing-side effects.

Task independence. Our twelve tasks are largely independent. A suite with strong sequential dependencies would give longer sessions a larger advantage and shift the minimum to the right.

Agent nondeterminism. The agent chooses how many calls to spend per task, so model calls are an outcome rather than a controlled variable. This is why we report replicate spread and use permutation rather than parametric tests.

6 Related work

Work on long-context models focuses on whether a model can use a long context; we ask what it costs to keep giving it one. Work on prompt caching reports cache hit rates and latency; we treat the cache as a priced resource and ask how session policy moves spend between its two sides. Agent benchmarks measure task success at fixed cost or cost at fixed success; the session-length lever is orthogonal to both and, unlike model choice or prompt engineering, is free to change.

7 Availability

Run-level counters for all 36 runs — token counts by category, model calls, sessions, wall clock, and test outcomes, with run identifiers hashed and no session content — are released together with the measurement methodology. A companion report decomposing where agent spend actually goes is archived at https://doi.org/10.5281/zenodo.22688706. The research index is kept at https://arsentev.ai/research.

8 Conclusion

Session length is a free lever with a 25% spread, and the best setting is neither of the two settings people actually use. Clear the context, but not every task: on our suite anything from every third to every sixth task sits in the flat bottom, while clearing after each task costs 25% more and never clearing costs 16% more. More useful than any single number is the reason for the shape — clearing trades cache writes against re-read context, the two move in opposite directions, and the price ratio between them decides where the balance falls. Any team can measure their own minimum in an afternoon, and on current prices it is worth the afternoon.