Union-Find Compaction: Provenance-Preserving Context Compression for LLM Agents at No Cost to Recall

June Kim
Independent Researcher
june@june.kim
ORCID 0009-0005-3153-9396

July 5, 2026

Draft. When a conversational agent's context window fills, the standard fix is flat summarization: run a cheap model over the old messages, replace them with a paragraph, discard the sources. That paragraph cannot trace a claim to its origin, cannot be re-expanded, and cannot be retrieved selectively, and every compaction stalls the session while it reprocesses the whole history. We compact instead through a union-find forest, where each old message is a node, similar messages merge into equivalence classes, and each merge is one cheap summarizer call. The result is a free lunch: at no recall regression, provenance-preserving compaction is also cheaper and faster, and it gains four properties flat summarization cannot offer. Provenance, recoverability, incremental graduation, and persistence hold by construction: `find` traces any summary to its sources, `expand` reinflates a cluster, and the forest serializes as integer parent pointers. In a feature-flagged gemini-cli integration over twelve real GitHub-issue conversations, union-find recalls at least as well as flat summarization (+8.3pp, 30.2% vs 21.9%, no significant difference favoring flat) at 0.79x the cost (21% cheaper) and sub-millisecond append and render latency; across a seven-trial controlled study it is tied-or-higher in every trial. Recall is directionally better, significant in one of seven controlled trials but not in the field, and we preregister a higher-powered replication that would upgrade non-regression to improvement. The standing result: the structural and cost wins come free, at no cost to what the agent remembers.

1 Introduction

Every conversational agent has the same bounded-context problem. Conversations grow, the window does not. When it fills, the dominant fix is flat summarization: run a cheap model over everything outside a hot window, compress it to a budget, and let the summary replace the sources. Gemini CLI compresses the oldest 70% of a session into one snapshot; comparable agents do the same.

Flat summarization works, and it destroys three things at once. The summary replaces every source, so no claim in it can be traced back to the message that produced it. The sources are discarded, so nothing can be re-expanded once compressed. And because compression runs over the whole history in a single pass, every compaction is a batch stall: twenty to thirty seconds of spinner while the session waits. Whatever the summarizer prompt did not prioritize is gone, invisibly, at compression time.

The three losses are not statistical. They are structural properties of replacing a set of sources with a paragraph, and no summarizer quality improvement recovers them. A better model writes a better paragraph; it still cannot cite, cannot expand, and cannot compact one message without reconsidering the rest.

We keep the sources. Represent the cold context as a union-find forest: each graduated message is a node, topically similar messages merge into equivalence classes, and each `union` is a single cheap summarizer call that folds two small summaries into one. The summarizer calls were going to happen regardless; routing them through a disjoint-set structure is what turns them into a provenance spine, because `find` recovers the source messages of any cluster and `union` merges one message at a time. That the operations are also near-O(1) amortized (Tarjan 1975, with path compression and union by rank) keeps the bookkeeping free, though at the scale we test the merge and lookup semantics matter more than the bound. The structure never forgets which messages belong together.

From that one substitution, four properties follow by construction rather than by measurement:

- Provenance. `find(m)` walks parent pointers to the cluster root, so every summary traces to its source messages. Any claim is auditable.
- Recoverability. `expand(root)` reinflates a cluster to its sources. Raw messages stay addressable; flat summarization discards them.
- Incremental. Messages graduate one at a time, each in near-O(1). No batch stall, no latency spike; flat summarization reprocesses the entire history per compaction.
- Persistent. The forest serializes as integer parent pointers. Save it, reload it next session, clusters intact.

These four hold by construction: a structure either preserves provenance or it does not, and no significance test establishes them. What remains empirical is the precondition that makes them free. Provenance would be a trade, not a gift, if it cost recall; so the decisive claim is that routing compaction through the forest costs no recall against flat summarization, at less money and lower latency. That precondition has to be earned, and earning it is non-regression, not improvement.

Contributions. (1) Union-find as the provenance spine for context compaction, with the four structural properties above. (2) A lazy-summarization design that keeps the per-session summarizer cost linear rather than quadratic in cluster growth. (3) Evidence from two studies, a controlled synthetic one and a feature-flagged gemini-cli field integration, that at matched token budget union-find recalls at least as well as flat summarization while costing less: the non-regression that makes the structural properties free. (4) A preregistered higher-powered replication that would upgrade non-regression to a positive recall improvement, stated as the confirmation this draft does not yet claim.

2 What Flat Compaction Destroys

Treat the agent's context manager as a cache with a fixed capacity and an eviction policy. Flat summarization's policy is: when full, replace the cold region with a single summary of it. Read as a cache, that policy fails three separate contracts.

Traceability. A cache that answers from a derived value should be able to name the sources the value came from. Flat summarization cannot: the paragraph is a lossy function of the whole cold region with no inverse and no index. When the agent later asserts "the scrape interval is 30s," nothing connects that to the message that set it.

Recoverability. Eviction from a flat summary already happened, silently, at compression time; there is no way to reinflate a detail the summarizer dropped. Eviction from a structure that keeps its sources is a deferred policy choice, not an irreversible event.

Selective retrieval. A single summary is retrieved whole or not at all. There is no way to pull the one cluster relevant to the current turn while leaving the rest compressed, because there are no clusters, only the block.

The union-find forest restores all three because it never overwrites the sources; it only adds structure over them (Table 1).

	Flat summarization	Union-find compaction
Provenance	none (paragraph has no inverse)	`find` → source messages
Recoverability	sources discarded	`expand` → reinflate cluster
Selective retrieval	whole summary or nothing	nearest cluster injected
Compaction unit	whole history, single pass	one message, incremental
Latency	batch stall (20–30s)	sub-millisecond append
Persistence	re-summarize each session	forest serialized as integers

Table 1. What each method preserves. The right column holds by construction; the evaluation asks whether it also costs any recall (it does not) and what it costs to run (less).

3 Method

3.1 The forest

Context splits into two zones. The hot zone is the last k messages (default k = 10), served raw. When the window overflows, the oldest hot message graduates to the cold zone, a union-find forest where each cluster is one summary over its source messages (Figure 1).

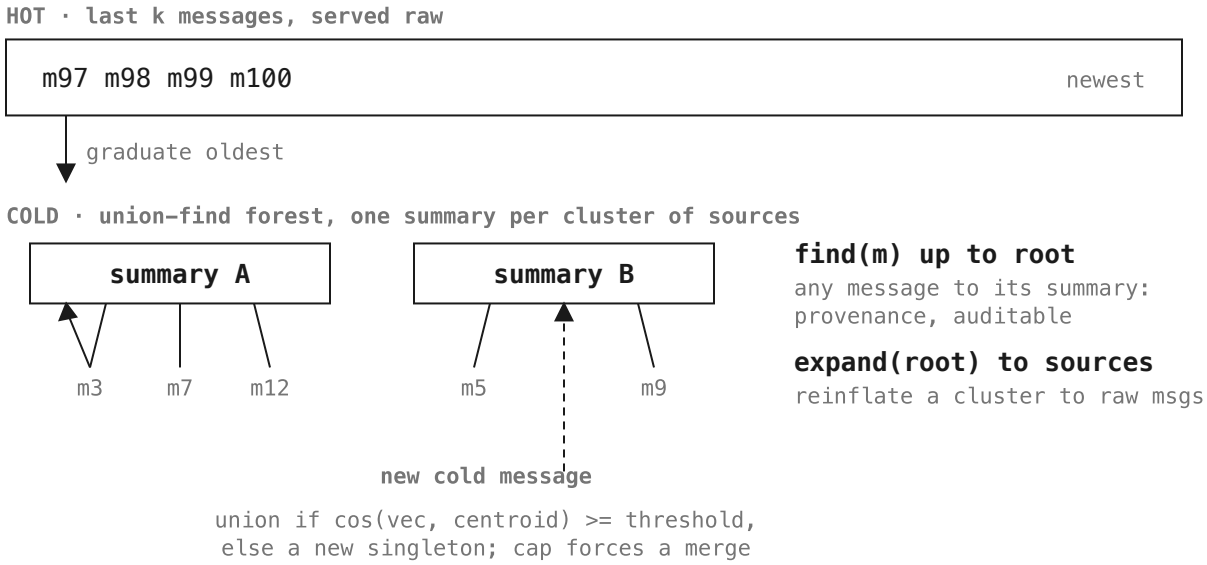

Figure 1. Messages graduate oldest-first from the hot window into the cold forest. Each cluster holds one summary over its sources; ***find*** recovers a summary's sources, ***expand*** reinflates a cluster, and a new message joins by ***union*** when it is close enough to a centroid.

The write path, per graduated message:

1. Keep its timestamp; compute a TF-IDF vector (local, no model call).
2. Cosine-compare against cluster centroids. Above the merge threshold (default 0.15), `union` into the nearest cluster and refold its summary; below, start a singleton.
3. If clusters exceed the cap (default 10), force the closest pair to merge. Centroids update as weighted averages, so the geometry stays current without re-vectorizing history.

The read path injects the nearest cluster's summary beside the hot window. Unlike retrieval-augmented generation, summaries are pre-merged at write time, so injected context stays bounded and no per-turn retrieval call is needed. Each of the four properties is one operation here (`find`, `expand`, the single-message write path, serializing the pointer array), and none depends on the recall results below; they hold for any forest this path produces.

3.2 Lazy summarization

The naive forest re-summarizes a cluster from its full membership on every merge. Message 90 re-reads 1 through 89; message 91 re-reads 1 through 90. That is quadratic in cluster growth. In an early build it produced roughly 80 summarizer calls per conversation against flat summarization's 2, a 5.2x cost premium.

The fix is to summarize lazily. `union` is synchronous and only records that a cluster's inputs are dirty; the actual re-summarization is deferred and coalesced, so a cluster that absorbs ten messages in quick succession pays one summarizer call, not ten. This keeps per-session summarizer cost linear in the number of clusters rather than quadratic in cluster size, and is what moves the method from a cost regression to a cost improvement (see §4.2). Folding two small summaries is cheaper than compressing the whole history, so a cluster of 5 to 20 messages summarizes with a small prompt and a small model.

4 Evaluation

The structure leaves one empirical question unsettled: at a matched token budget, does routing compaction through the forest cost any recall the agent would otherwise keep? The structural properties are free only if the answer is no, so non-regression is what the two studies below test, one controlled and one in production. A significant improvement, if it holds, is upside; non-regression is the result the free lunch rests on.

4.1 Controlled study

A synthetic 200-message DevOps conversation seeded with 40 verifiable facts. Both methods use the same cheap summarizer (Haiku), the same token budget, and the same retrieval machinery. A strict LLM judge scores binary recall: "PostgreSQL 16.2" counts, "PostgreSQL" does not. McNemar's test on the discordant pairs. Seven trials vary the summarizer, the compression ratio, retrieval, tuning, and timestamping.

#	Config	Flat	UF	p
1	Haiku, 50	90%	90%	1.000
2	Haiku, 200	65%	82%	0.039
3	Sonnet, 200	70%	78%	0.453
4	Haiku, 200, retrieval	68%	82%	0.180
5	Haiku, 200, tuned	62%	80%	0.065
6	Haiku, 200, timestamps	72%	90%	0.092
7	Haiku+Sonnet, 200	75%	90%	0.070

At low compression (50 messages) the methods tie: with little to discard, provenance structure buys nothing, and neither loses. At 200 messages union-find is higher in every trial, by 8 to 18 points, and lower in none. Trial 2 clears significance ($p = 0.039$); the rest are directional at $n = 40$ facts. The seven trials vary different axes (summarizer, compression ratio, retrieval, tuning, timestamps) rather than repeat one test, so they read as a consistent direction across conditions, not seven shots at a single hypothesis to correct for. What every trial shares is the floor: union-find never recalls less than flat. Trial 7 mirrors production, a cheap model summarizing and an expensive model answering, and shows why the floor matters: the expensive answerer cannot recover facts the cheap summary already dropped, so any recall difference has to be won at compaction time, not answer time.

4.2 Field study

We implemented the method as a feature-flagged fork of Gemini CLI and evaluated on 12 real GitHub-issue conversations of about 120 messages each, 8 factual questions per conversation (96 total), generated from the uncompressed content and scored by a blinded LLM judge. Flat compression runs on the same data. Three hypotheses were preregistered before any run.

Hypothesis	Result	Detail
Latency	PASS	append p95 = 0.33ms, render p50 = 0.006ms
Cost	PASS	0.79x flat, 21% cheaper
Recall	Trending	+8.3pp (30.2% vs 21.9%), $p = 0.136$

Latency and cost pass decisively. Lazy summarization makes 35 summarizer calls across 12 conversations where flat makes 24 and the naive quadratic forest would have made 960; the extra 11 calls over flat feed small clusters with small prompts, which is why total cost still lands below flat. The 0.79x figure is summarizer (API) cost, the dominant term; the forest's local operations (TF-IDF vectorization, cosine similarity, centroid updates, serialization) make no model call and are excluded as negligible against a summarizer round-trip. Append and render are sub-millisecond, so compaction no longer stalls the session.

Recall does not regress. Union-find is +8.3 points across 96 questions (30.2% vs 21.9%), winning 8 conversations, tying 2, and losing 2, with no significant difference favoring flat ($p = 0.136$). The point estimate favors union-find; the evidence is consistent with a moderate gain and with no difference, but not with a loss. That is the claim the free lunch needs, and no more than it: recall is not traded away for provenance. This is a non-inferiority reading of a two-sided test read for direction; the formal version, with a stated margin, is preregistered in §6, not yet run, and two of the twelve conversations did lose head-to-head. We do not yet claim a positive recall improvement from this study.

The integration is public and inspectable to any depth: the implementation submitted as PR #24736 (not merged), the issue, the design discussion, and the preregistration, raw data, and latency CSVs in the spec repository.

Google announced on May 19, 2026 that gemini-cli would transition to Antigravity CLI, stopping service for free and individual users on June 18, 2026; this study predates both. The method is not specific to gemini-cli. Any agent that compacts by flat summarization can adopt the same forest, so the platform's sunset dates the artifact, not the approach.

4.3 Why the footnotes survive

Where union-find leads, it leads on footnote facts. Flat summarization preserves headline facts (the database version, the auth scheme) and drops the scrape interval, the cron schedule, the webhook path, the filterable-attribute count: the details that separate having read the conversation from having read a briefing. The mechanism is competition. Flat summarization compresses the whole history in one pass, so every fact competes for space in a single budget and footnotes lose to headlines. Union-find compresses per cluster, 5 to 20 messages each, so the cron schedule is summarized alongside its neighbors, not against the database version. Facts compete only within their cluster, and most footnotes are the most important fact in some small cluster.

5 Related Work

Flat summarization is the deployed baseline in production agents (Gemini CLI, and comparable context managers) and the method we measure against. Its known failure, dropping detail under a single-pass budget, is what motivates a structured alternative.

Structured eviction and provenance memory. A concurrent line replaces single-summary compaction with structured eviction and provenance-tagged agent memory. These share the diagnosis: flat summarization drops fine-grained facts, and provenance should be preserved. They differ in the primitive: none uses a disjoint-set forest with equivalence-class merging as the compaction structure, which is the specific contribution here.

Submodular and diversity-aware selection. Context selection by submodular or determinantal objectives targets a related goal (keep a diverse, non-redundant subset) but selects among items rather than merging them into canonical, re-expandable clusters, and does not provide provenance back to sources.

Retrieval-augmented generation retrieves raw passages per query at read time; union-find pre-merges at write time, giving bounded injected context and no per-turn retrieval call, at the cost of committing to a clustering online.

Memory benchmarks. Standard public benchmarks already measure long-horizon recall over multi-session dialogue, notably LoCoMo and LongMemEval, alongside single-input long-context probes (Needle-in-a-Haystack, RULER, LongBench). We measure union-find against flat summarization on our own data rather than on these; running it on an established memory benchmark is the natural next evaluation (§6), not a gap to be filled by a new benchmark.

Union-find itself is Tarjan (1975); the contribution is not the algorithm or its complexity bound but its use as a provenance spine for context compaction, where `find` supplies message-level lineage and `union` supplies single-message incremental merge.

> Citations in this section are drawn from a prior-art sweep and name concurrent work by topic; the specific arXiv identifiers must be verified against the sources before submission. An earlier form of this work appeared on the author's blog and is the method's only prior public description; a preprint must cite it as such and clear the venue's prior-publication bar.

6 Limitations and Planned Confirmation

6.1 What is established

What this draft claims empirically is non-regression, not improvement. Union-find is tied-or-higher in all seven controlled trials and +8.3pp in the field with no significant difference favoring flat, which supports the precondition the free lunch needs; a positive recall improvement is not yet established (one of seven controlled trials at $p < 0.05$, the field study at $p = 0.136$). Formal non-inferiority testing, with a stated margin rather than a two-sided test read for direction, is the right frame and is not yet run.

6.2 The preregistered confirmation

We preregister it. A higher-powered replication holds the design fixed and scales to 200 or more paired questions across a larger conversation corpus, powered to detect an 8-point difference at the observed base rate, and adds two conditions that stress provenance directly: contradictory facts and stale facts, where a later message overrides an earlier one and the compaction must keep the correction traceable rather than blend the two. The hypothesis, the design, and the analysis are fixed here in advance; only the token budget to run it is pending. The gap is not ours alone: maintainers of the integration target independently flagged that a compression eval harness is badly needed, and this study is that harness. Should it fail to confirm, the structural contribution and the non-regression result stand, and only the improvement claim is withheld.

6.3 Further limits

The evaluation data is a proxy, not the deployment distribution. The controlled study is a synthetic DevOps conversation and the field study is scraped GitHub issue threads; neither is an agent-native session, which interleaves tool output, code, and agent turns at a density and structure a human issue thread lacks. Issue threads share the property under test (facts scattered across messages, footnotes competing with headlines), so they are a fair testbed, but whether the recall and cost results transfer to real agent context is untested. The fix is not a new benchmark, which this space already has, but a standard one: the higher-powered study should evaluate union-find on LongMemEval, preferred over LoCoMo for its evidence-tagged questions and cleaner answer key, and scored against that tagged evidence rather than by an LLM judge, whose leniency is the weak point of both benchmarks and of our own field study. That is left to future work.

Three more. The cost win is measured at one scale (about 120 messages, ten clusters); union-find makes more and smaller summarizer calls than flat's few large ones, and the crossover point where fixed per-call overhead would erase the saving at much longer conversations is uncharacterized. The evaluation compares against flat summarization only, not against the concurrent structured-eviction systems of §5, so the evidence shows union-find beats the deployed baseline, not that it beats the nearest structured alternative. And storage grows monotonically: the forest only adds nodes and the source store accumulates, so a long-lived deployment eventually needs a cluster-eviction or archival policy, deferred here. The merge threshold, cluster cap, and hot-window size are fixed defaults whose effect on recall is not characterized.

7 What It Unlocks

The forest serializes into a single small store (integer parent pointers, cluster summaries, and TF-IDF vectors fit a SQLite database), and from that four capabilities follow that flat summarization cannot offer:

- Persistence across sessions. Prior conversations reload as pre-merged clusters, not a blank window, with no pinned notes and no "as we discussed last time." A new session inherits what earlier ones understood.
- Multiplayer read-access. Persist the forest across people, not just sessions, and two agents on one repository feed one forest. Agent A files a race condition it hit in the payment flow; agent B, on an unrelated feature, queries the forest and routes around it, the two never communicating. A new contributor's agent inherits the forest on clone. The result is a record the others miss: code is what the software is, git history is what changed, the forest is what is understood.
- Concurrent write. Union-find merges are associative, commutative, and order-independent, so two sessions writing in parallel converge without a referee, the way a CRDT does; when two claims contradict, both are filed with provenance and the reader judges. The session is where you work, the forest where you meet.
- Branching. Forking the forest for a what-if conversation inherits all context, giving version-control semantics for conversations.

Each follows from keeping the sources and their structure instead of a paragraph. Governance is a thin layer over the same store: access modes (private, read-only, shared) and a retention policy on unqueried clusters, with the clustering parameters left as convention. The fuller vision of a shared understanding-layer for a team is developed separately (Union Found); the narrower point is that the data structure measured here already carries the persistence, provenance, and conflict-free merge those capabilities need.

8 Conclusion

Flat context compaction trades provenance, recoverability, and selective retrieval for a paragraph, and stalls the session to do it. Routing the summarizer calls that were happening anyway through a union-find forest

recovers all three by construction, at 0.79x the cost and sub-millisecond latency, and it does so without costing recall: union-find is tied-or-higher in every controlled trial and does not significantly regress in the field, pending the formal non-inferiority test we preregister. The standing result is that the structural and cost wins are free, at no measured cost to what the agent remembers. The open one, which we have preregistered, is whether recall also improves outright.

9 Availability

Code, data, preregistrations, and result logs are archived on Zenodo: the controlled study at 10.5281/zenodo.21215158 (union-find-compaction) and the field study, with its three preregistrations and raw results, at 10.5281/zenodo.21215160 (union-find-compaction-for-gemini-cli). The gemini-cli implementation was submitted as PR #24736. Released under CC BY-SA 4.0.

LLM collaboration disclosure

LLMs enter this work in three roles. Subject of study: the compaction method drives shipped LLMs as its summarizer (Claude Haiku 4.5) and, in the production-mirroring trials, its answerer (Claude Sonnet 4.6), and recall is scored by a blinded LLM judge; the model versions are recorded in the archived result logs (§9). Instrument: independent model families adversarially reviewed the draft and its claims, and the recall judge scores each answer against the uncompressed source, with the mechanical layer (McNemar's test, the harness) holding the verdict. Writing aid: the prose was drafted and revised with Anthropic's Claude (Opus 4.8) from the author's blog posts, experiment logs, and direction; the method, the experiments, the numbers, and the argument are the author's. No LLM decided what to publish.

Funding

This work was conducted independently, with no external, institutional, or commercial funding. All compute and model-API costs were borne by the author.