What Is Context? What a Model Can Hold, and What Happens When It Runs Out of Room

Background to the post of the same name. How much a model can hold at once and how well it uses a long context, what harnesses do when a conversation outgrows the window, what a summary keeps and what it loses, and how one practice carries its rules and its open work through the summary.

Berg Atkinson, Wolfberg LLC · berg@wolfberg.ai

Preprint. Not peer reviewed. Published 28 September 2026.

IN PLAIN TERMS

A language model can only look at so much at once. Everything it works from on a given turn, the instructions, the files, the conversation so far and whatever its tools returned, has to fit in a fixed space called the context window. Windows are large now, but models get less reliable as the input grows, well before the window is full: in a 2026 benchmark, the model that started best finished 96% of a set of agent tasks with 8,000 tokens to read, and 34% of the same tasks with 128,000. Over a long conversation they also lose their way, and once they take a wrong turn they rarely recover.

When a conversation outgrows the window, some tools let the oldest part fall out of view or ask for a new conversation, and others, among them the coding agents from Anthropic, OpenAI and Google, compact it: a model writes a summary of the conversation so far, and the work carries on from the summary. From then on, what the model knows about the earlier conversation is an account written by a model. Summaries drop detail, and they can change meaning: in a 2026 study of current models, compaction kept on average 17% of the instructions users had given for the rest of a session. In the practice this series describes, one summary reversed the meaning of a rule, and another left an agent rebuilding an agreed process from memory instead of reading it.

> What reliably survives a compaction is what the harness loads back in from files. So the defense sits mostly in the harness: rules kept in files that reload on their own, records the machinery writes rather than the agent, a refusal to change the rules until the sources have been read again, and a count of every compaction and whether a reload followed. Anyone using any AI tool can do a smaller version of the same: start fresh for a new task, keep standing instructions where the tool keeps them, ask for a re-read before acting on something from earlier, and assume the early part of a long chat is summarized or gone, whether or not the tool says so.

About this paper. This is a new background paper in the series *How do we use this?*, written to go with the post "What is context?", the third of the four primer posts. It extends background paper 8, *What Is a Harness?* (Atkinson, 2026b), by going further into the first of that paper's six jobs, what the model is shown, which includes what the harness does when the conversation no longer fits. Where this paper relies on the practice's own records, it gives the date each was read. The practice's count of compactions moves every session, and it moved while this paper was being written (§5.3, §6.4).

CONTENTS

1 Introduction

Background paper 8 described the software around a language model, the harness, as six jobs: what the model is shown each turn, what it can look up, what it can touch, what it keeps between turns, when it stops, and whether anything checks the work. A reader of the post that goes with it pointed out that it left out something every long session depends on: what the harness does when the conversation grows too long for the model to hold. That belongs to the first job. Background paper 8 places it with the context manager, the component that decides what the model is shown, whose duties in the survey it follows include compression and summarization (§4.2), and gives it no more than that mention. This paper gives it the room it needs.

The job is easy to mistake for housekeeping, but it changes what the model is working from. Before a compaction, the model reads the conversation itself. After one, it reads a model-written summary of the conversation, and nothing in the model's replies shows which of the two it is working from, even when

the tool tells the user that a compaction happened. Anthropic's guidance to agent builders names the risk in one sentence: “The art of compaction lies in the selection of what to keep versus what to discard, as overly aggressive compaction can result in the loss of subtle but critical context whose importance only becomes apparent later” (Anthropic, 2025a).

This paper makes three claims. First, models use long contexts unevenly and degrade over long conversations, so the practical limit on what a model can use arrives well before the nominal size of its window (§2, §3). Second, compaction, the response the coding agents examined here share, replaces the conversation with a summary; summaries drop detail and can change meaning, and what survives reliably is what the harness loads back in from outside the conversation (§4, §5). Third, the defense is therefore a matter of harness design: rules that reload on their own, a record the agent does not write, a constraint that stops changes to the rules until the sources have been read again, and a count of compactions that makes the gaps visible. The practice's own record shows each of these being added after the one before it failed (§6). Section 7 turns the findings into steps anyone can take with any tool, and §8 sets out the limits.

2 The window, and how models use a long one

A model reads only what is in its context, which Anthropic defines as “the set of tokens included when sampling from a large-language model” (Anthropic, 2025a). Everything the model works from on a turn has to fit there: the system instructions, the user's instructions, any files, the conversation so far and the results of any tools it used. The size of the window is a number on a model's specification sheet. How much of it the model can use well is a different number, and current research puts it well below the first.

Longer is worse, even when the task stays the same. LOCA-bench, a 2026 benchmark, gave seven current models the same fifteen agent tasks, such as gathering exam dates from a course system and an inbox into a sorted spreadsheet, and varied only how much the agent had to read to finish them, from 8,000 to 256,000 tokens. “As the context grows, performance drops sharply even though the underlying task does not change” (Zeng, Huang & He, 2026). Claude 4.5 Opus succeeded on 96% of runs at 8,000 tokens and 34% at 128,000; GPT-5.2 went from 72% to 38.7%, and Gemini 3 Flash from 64% to 21.3% (Figure 1). The authors describe what the failures looked like: as context accumulates, the model often becomes “impatient” and “may end the task prematurely, stop exploring, and mistake partial evidence for a complete review.” A 2025 report from Chroma found the same direction across 18 models current at the time: performance “grows increasingly unreliable as input length grows”, and “varies significantly as input length changes, even on simple tasks” (Hong, Troynikov & Huber, 2025). Anthropic calls the effect context rot and concludes that context “must be treated as a finite resource with diminishing marginal returns” (Anthropic, 2025a).

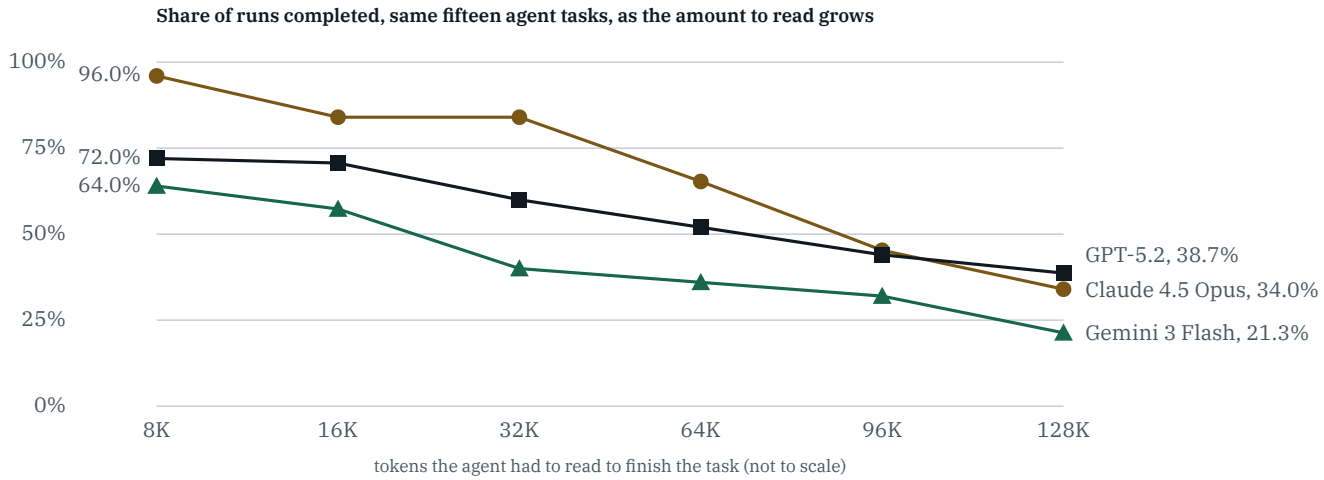

Figure 1. Success on LOCA-bench's fifteen agent tasks as the environment the agent must read grows, for the three frontier models tested; each point is 75 runs (Zeng, Huang & He, 2026, Table 1). The tasks do not change, only the amount to read. At 256,000 tokens, beyond Claude 4.5 Opus's 200,000-token window, all seven models tested scored below 22%. **Measured.**

Where the answer sits matters less than it did, but it still matters. Liu et al. (2023) found that the models of that year used the start and end of a long input best and the middle worst, the effect known as lost in the middle. A 2026 reproduction of that experiment with current open models did not find the pattern: “we do not recover a clear U-shaped curve: accuracy remains comparatively flat across positions, with only a slight upward trend” (Gabín, Perez & Parapar, 2026). Position still matters in harder settings. A 2026 study that placed a reasoning problem inside long filler text found that models “can drop sharply when the target task moves from the end of the context to the middle”: one fell 88 points at 64,000 tokens, and four newer releases, much better with one type of filler, still dropped 16 to 56 points with another. The same study found a cheap remedy. Adding a copy of the task at the end brought middle-position accuracy back to within 4 points of the end-position score, at 8,000 tokens, for all nine models it tested (Zhang et al., 2026).

For a harness, the consequence is that a full window costs something long before it is full, and that what must govern the work belongs where the model will use it, not somewhere in the middle of a long history. The working paper reached the same point from the practice's side: “Being first is a structure” (§5.1 of the working paper, as background paper 8 reports it).

3 Long conversations

A long session is not only a long input. It is a sequence of turns in which the task is often revealed a piece at a time, and in which the model's own earlier answers become part of what it reads. Laban, Hayashi, Zhou and Neville (2025) measured what that does. They took instructions from six kinds of task, among them writing code, database queries and math, and delivered each one two ways: whole, in a single turn, and in pieces over several turns, with a simulated user revealing one piece per turn.

Across 15 models from eight families, the frontier of early 2025 among them, and more than 200,000 simulated conversations, “all the top open- and closed-weight LLMs we test exhibit significantly lower performance in multi-turn conversations than single-turn, with an average drop of 39% across six generation tasks.” (The paper's opening figure states the same comparison as 35%.) Handing a model all the pieces at once, in a single turn, did nearly as well as the original instruction, 95.1% of it on average, so the loss comes from the conversation and not from the way the instructions were cut up.

The drop was mostly not a loss of ability. The authors split performance into two parts: a best case, the 90th-percentile score over repeated runs of the same instruction, and unreliability, the gap between the 90th and the 10th percentiles. In pieces, the best case fell by 16% on average while unreliability rose by 112%, with “performance degrading 50 percent points on average between the best and worst simulated run.” In their words, the drops “are due in large part to increased model unreliability (U), rather than a loss in aptitude (A)” (Figure 2). Their abstract gives the sentence this paper's post borrows: “when LLMs take a wrong turn in a conversation, they get lost and do not recover.”

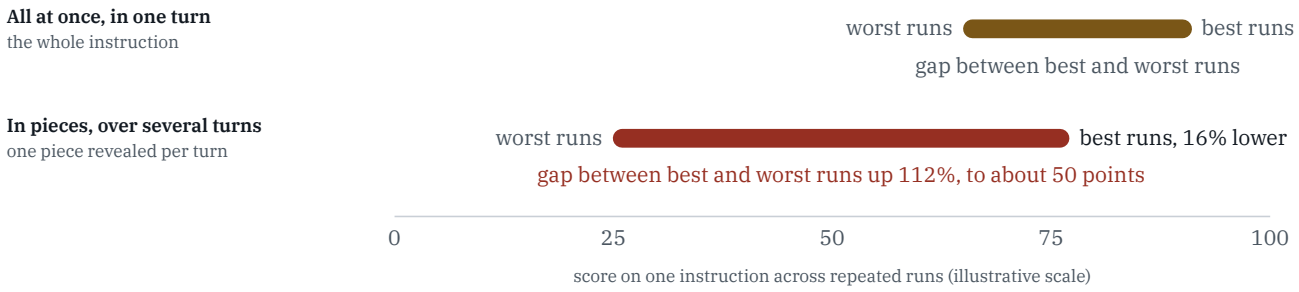

Figure 2. Laban et al.'s (2025, §6.2) decomposition of the multi-turn drop. The best runs, the 90th percentile, fell 16% on average; the gap between the best and the worst runs, the 90th and the 10th percentiles, rose 112%, to about 50 points. The ranges are drawn to those averages from an illustrative starting level; they are not one model's scores. **Schematic, to the paper's reported averages.**

They traced the loss to four behaviors: attempting a full answer before the task was fully stated; leaning on those early attempts once they were wrong, which the authors call answer bloat; attending least to the middle turns of the conversation, a turn-by-turn cousin of Liu et al.'s finding; and long replies that pile up assumptions. The obvious fixes did not help: the two reasoning models tested “deteriorate in similar ways to non-reasoning models”, and lowering the sampling temperature “is ineffective in improving system reliability.” Their advice to users is plain. “If time allows, try again,” because “starting a new conversation that repeats the same information might yield significantly better outcomes than continuing an ongoing conversation”; and “Consolidate before retrying”, gathering the requirements into a single instruction. They add that both “can only offer patched solutions rather than a principled approach.”

The conversations were simulated, the tasks analytical and the language English, and the authors believe the study more likely understates the problem than overstates it (§8). It bears on compaction in two ways. A long session is exactly the setting in which models get lost, so a conversation that reaches

the limit may already carry a wrong turn. And a summary written from inside that conversation may carry the wrong turn forward.

4 What harnesses do at the limit

Anthropic's guidance lists three techniques for work that outlasts one window: “compaction, structured note-taking, and multi-agent architectures” (Anthropic, 2025a). The harness the practice in this series runs on, Claude Code, uses all three. Other coding agents document at least the first: the source code of OpenAI's Codex describes its /compact command as “summarize conversation to prevent hitting the context limit” (OpenAI, n.d.), and Google's Gemini CLI documents /compress, which will “Replace the entire chat context with a summary” (Google, n.d.). Chat apps document less. Their users report the oldest part of a long conversation falling out of view without notice, or a message asking them to start a new conversation; no vendor page read for this paper says which tool does which.

Compaction. “Compaction is the practice of taking a conversation nearing the context window limit, summarizing its contents, and reinitiating a new context window with the summary” (Anthropic, 2025a). In Claude Code it happens automatically as the window fills, or when the user asks: “When a long session compacts, Claude Code summarizes the conversation history to fit the context window” (Anthropic, n.d.). Anthropic's guidance calls one narrow form “One of the safest lightest touch forms of compaction”: clearing the raw results of tool calls made deep in the history, which the model is unlikely to need again (Anthropic, 2025a). A 2025 comparison on software-engineering tasks found that light touch was enough: hiding old tool output halved the cost of keeping everything and solved as many tasks as LLM summaries, and the summaries led to longer runs, “suggesting they mask failure signals that would otherwise prompt earlier termination” (Lindenbauer et al., 2025). LOCA-bench found much the same with current models at 128,000 tokens: turning on compaction moved GPT-5.2 from 38.7% to 36.0% and Gemini 3 Flash from 21.3% to 24.0%, while having the model write code to call its tools, which keeps long intermediate tool output out of the context, raised them to 49.3% and 30.7%. Running Claude 4.5 Opus through Anthropic's own agent SDK lowered its score, from 34.0% to 26.7% (Zeng, Huang & He, 2026).

Notes kept outside the window. “Structured note-taking, or agentic memory, is a technique where the agent regularly writes notes persisted to memory outside of the context window. These notes get pulled back into the context window at later times” (Anthropic, 2025a). MemGPT built the idea into an architecture in 2023, “virtual context management, a technique drawing inspiration from hierarchical memory systems in traditional operating systems”, in which messages pushed out of the window are replaced by a running summary but kept in storage the model can search (Packer et al., 2023). Recent work keeps that shape and measures it on current open models. In a 2026 test, a design that compacts old context into short entries, each carrying an address the agent can use to recall the original, found a planted fact 99.40% of the time on average, against 88.12% for the best of the five other strategies it was compared with, LLM summaries among them (Dang et al., 2026). The test used two open Qwen3 models on small context windows, so it shows the principle rather than a result for today's largest windows: a summary cannot give back what it left out, and an address can.

Separate windows. A task can be split among agents that each work in their own window and hand back only a result. Claude Code's documentation describes a subagent that “handles the research in its own separate context window, so the large file reads stay out of yours. Only the summary and a small metadata trailer come back” (Anthropic, n.d.). What comes back is, again, a summary.

A harness built around the handoff. Anthropic's report on agents that work across many windows is candid about compaction: “compaction isn't sufficient”, because it “doesn't always pass perfectly clear instructions to the next agent” (Anthropic, 2025b). “The core challenge of long-running agents is that they must work in discrete sessions, and each new session begins with no memory of what came before.” Its fix was a harness fix: a first session that sets up the environment, including “a claude-progress.txt file that keeps a log of what agents have done”, and later sessions that make incremental progress and leave that log and the version history in order, so that the next one can “quickly understand the state of work when starting with a fresh context window.”

What survives, by design. Claude Code's documentation says what happens to each type of content when a session compacts (Anthropic, n.d.). The system prompt still applies. The project's root instruction file, CLAUDE.md, with its rules that are not tied to particular paths, the automatic memory, and any plan written in plan mode are “Re-injected from disk.” Up to five recently changed files are re-read, and the skills the user invoked are re-injected up to a cap. Hooks that run at session start with the source “compact” run again and add their output. Everything else lives on in the summary. “Context that hooks added earlier” is “Summarized with the rest of the conversation”, and rules tied to particular paths, which enter the conversation when a matching file is read, are summarized away with it: “If a rule must persist across compaction, drop the paths: frontmatter or move it to the project-root CLAUDE.md.” Figure 3 draws the split. The principle under it is general: what lives in files the harness loads comes back; what was only said in the conversation comes back only as the summary's account of it.

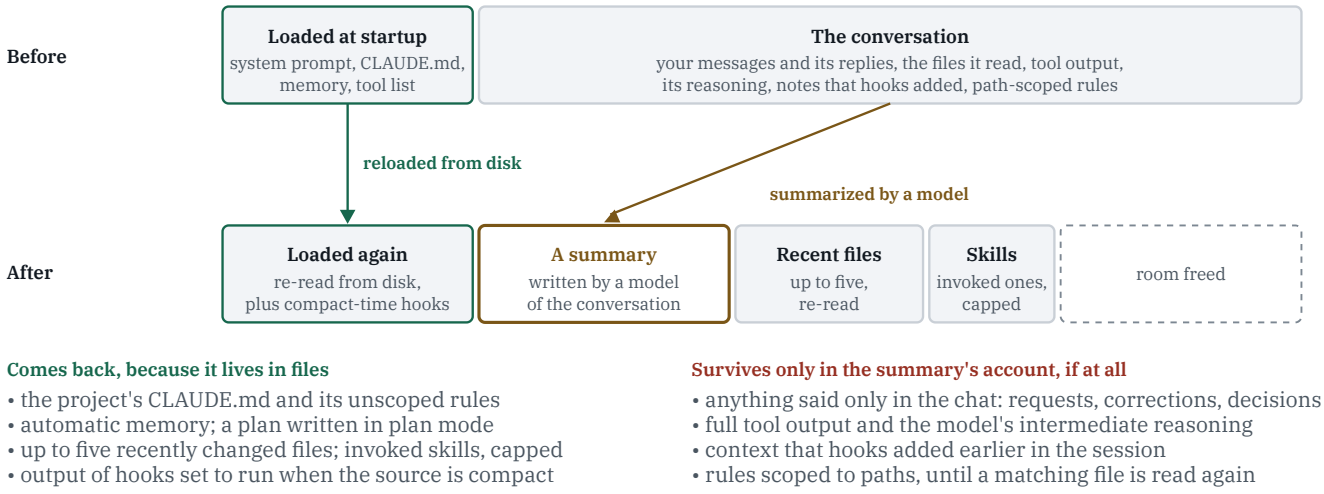

Figure 3. What survives a compaction in Claude Code, after the documentation's own table, "What survives compaction" (Anthropic, n.d., read 27 September 2026). The widths are not to scale. Other tools differ in detail; the split between what reloads from files and what is summarized is the general point. **Schematic, from the vendor's documentation.**

5 What a summary keeps and what it loses

5.1 What the tools say they keep

Claude Code's documentation lists what its summary keeps: "your requests and intent, key technical concepts, files examined or modified with important code snippets, errors and how they were fixed, pending tasks, and current work. It replaces the verbatim conversation: full tool outputs and intermediate reasoning are gone" (Anthropic, n.d.). The summary this paper's own drafting session received when it compacted on 27 September 2026 followed that plan closely, in nine headed sections running from the user's requests and intent to the pending tasks, the current work and a suggested next step. A structured summary of that kind is a good summary. It is still an account: it keeps what its writer judged important at the moment of writing, in the writer's words.

5.2 What summaries get wrong

A 2026 study measured the loss directly, on current models and on the instructions a harness most needs to keep. Wang, Zhang, Lee and Yang (2026) planted what they call session constraints in conversations of about 100,000 tokens, instructions such as "do not delete any emails until I confirm" that "are meant to constrain LLM's behavior for the remainder of a session", and compacted the conversations with eight setups: simple truncation, a token-pruning compressor, and four language models summarizing with either a compaction prompt from Anthropic's API documentation or the prompt of the open-source agent OpenClaw. "Current compactors retain only 17% of injected SCs on average, and most perform worse than running the same task without compaction." Truncation and the

token pruner kept none. The strongest compactor, GPT-5.4-mini, kept as many as 98% on one dataset and as few as 6.7% on another (Figure 4). Asking the compactor explicitly to keep constraints helped, but retention stayed below 40% on the hardest of the three datasets.

The constraints were not beyond the model. In a check where it acted with tools rather than answering multiple-choice questions, the model followed a constraint 31.8% of the time with the compacted context alone, 73.4% with the full uncompacted context, and 99.7% when the constraint was appended after compaction. What worked best was keeping the constraints out of the summary altogether: a small model that reads the user's messages and keeps a running list of constraints beside the compactor recovered 90.3 to 95.6% of them. The study planted each constraint once, at the top of the conversation, and the authors found that constraints stated later, repeated, or phrased strictly survive more often; it did not test Claude models as compactors. Its central finding is the one the practice's record shows below: a summary can keep the task and lose the rule.

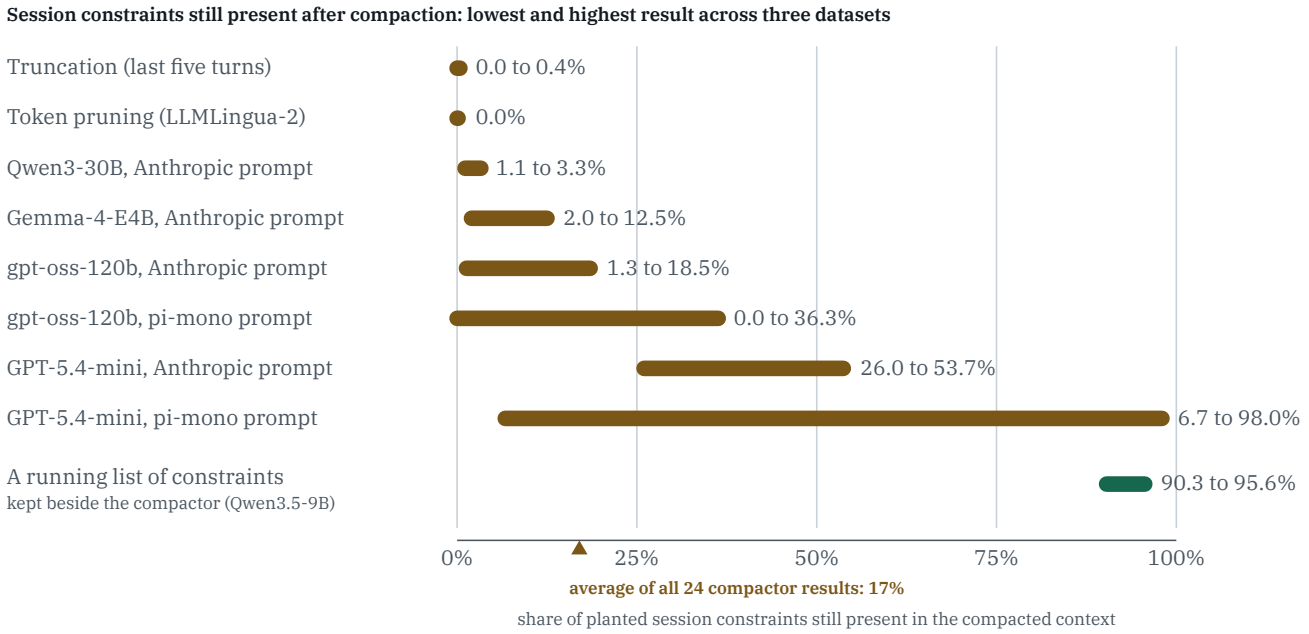

Figure 4. What eight compaction setups kept of a user's session constraints, each planted once in conversations of about 100,000 tokens, with the lowest and highest result across three datasets (Wang et al., 2026, Tables 2 and 4). The Anthropic prompt is the compaction prompt in Anthropic's API documentation; pi-mono is the prompt of the open-source agent OpenClaw. The green bar is a small model keeping a running list of the constraints beside the compactor. GPT-5.4-mini was run on longer contexts and fewer trials than the others. **Measured.**

Losses do not always show up as failures. In a small planning environment of its author's design, removing facts from an agent's context left GPT-5.5's task completion “statistically unchanged (80% → 85%, p = 1.0)” while its “retrieval roughly triples (+42.9 calls)”: the agent went back for what it had lost, and a completion rate alone would not have shown the loss (Liu, 2026). Older work named the two ways a summary can be wrong. Maynez, Narayan, Bohnet and McDonald (2020) separated extrinsic hallucinations, which add something the source does not support, from intrinsic ones, which “use

terms or concepts from the document but misrepresent information from the document." Their summarizers predate today's language models, but the second category is the one the first case below turns on: a summary made of a source's own words that says something the source does not.

5.3 From the logs

The practice's record holds three cases. The third happened while this paper was being written.

A rule reversed. On 26 August 2026 an agent's compacted summary reversed the meaning of one sentence in the practice's own boot rules. Working from the summary, the agent concluded that the rule was wrong, said so, and prepared a "fix" that would have broken a correct line. What stopped it was another rule: any page of the rules must be read whole, in the current session, before it is changed. It is an intrinsic error in Maynez et al.'s sense. The summary was made of the rule's own material and said something the rule did not.

A process rebuilt from memory. On 26 September 2026 the agent writing this series was asked, after a compaction, for a handoff: the document the next session starts from. It wrote one from its memory of the practice's handoff process instead of reading the process, and left out steps the operator had laid down. The operator caught it by asking whether the handoff followed the process that had been laid out. It did not. The error was logged as one the operator caught, with the lesson recorded against it.

A count carried stale. On 27 September 2026, while this paper was being drafted, the same session compacted again. Its summary carried the practice's count of compactions as last read: 82, of which 55 had been followed by a reload. After the reload the count read 83 and 56, because the compaction that produced the summary had itself been counted. The summary also said that the session's open items would appear in the state file the harness writes at each compaction (§6.2); the file counted them but did not list them. Neither error would have mattered much. Both were caught only because the count and the file were read again rather than recalled.

6 From the logs: carrying the work through a compaction

The practice runs long sessions on a closed frontier model family, under a harness of hooks, shared logs and automated checks, with an operator who approves every consequential change (background paper 8, §8). Compactions are routine: the count in §6.4 reads 84 since June 2026. The practice's defense against them was not designed at once. It was built in steps, each added when the one before it was not enough (Figure 5), and it is described here by what each step does.

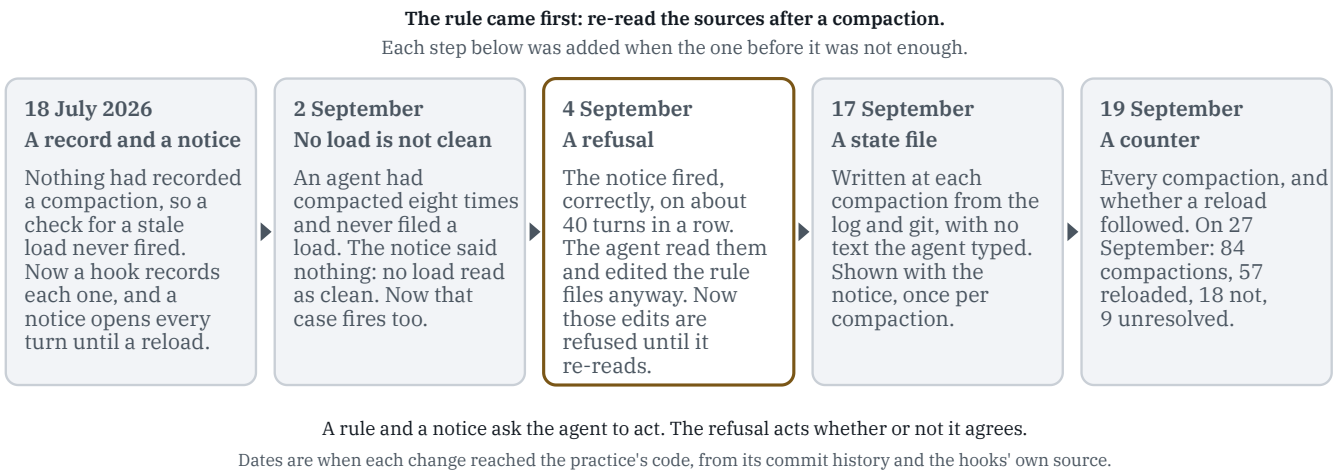

Figure 5. The practice's defense against compaction, in the order it was built. The refusal, outlined, is the only step that does not depend on the agent choosing to act. **Schematic; dates from the practice's record.**

6.1 Rules that reload on their own

The practice keeps the instructions it most needs followed in the files its harness re-injects after every compaction: the automatic memory, an index of more than a hundred short lessons, each pointing to a fuller note of what taught it, and, for sessions opened in its main repository, that repository's instruction file. Its two long rulebooks, of about 117,000 and 80,000 characters, are not loaded that way. The practice's boot procedure requires them to be read whole after every compaction, followed by a receipt that lists what was read and what was not, with a reason for each omission. The receipt is the agent's declaration of what it read. The harness checks it for depth against the size of each source, and can verify nothing more.

6.2 A record the agent did not write

At the moment of each compaction, a hook writes a row to the practice's event log. It has done so since 18 July 2026; seven rows dated earlier were added afterward from other sources (the guard's own source counted no compaction rows at all on 18 July). Since 17 September 2026 the same hook also writes a small state file, capped at 2,048 bytes and “derived only” from the practice's logs and version history. By design it “carries no field the seat types” (the practice calls an agent session a seat), so it can list what is open but cannot describe it: a description would be the agent's own words, and a file written from the agent's own words would be one more summary. It is shown alongside the notice of §6.3, once per compaction, the next time a prompt arrives. The cost of the design showed at this paper's own compaction, where the session's open items, recorded under names the agent chose, were withheld and only counted (§5.3).

6.3 A notice, then a refusal

The rule came first: after a compaction, re-read the sources before acting. It was written into the practice's boot steps, and it was not enough, for reasons the record shows in order.

Nothing recorded a compaction. The practice's load checker could tell a stale load from a fresh one from mid-July, but until 18 July 2026 nothing wrote a compaction to the log, so the check had nothing to fire on. Before the hook existed, an agent compacted in the middle of a session, worked about fifteen turns from the summary, and edited the practice's rules while its load was stale. The operator caught it, not the checker. The fix was the hook of §6.2 and a notice: at the start of every turn, while the agent's latest load predates a compaction, the harness prints the time of the load and the time of the compaction, and says that what the agent holds is “a model-written summary of those surfaces, not the surfaces.”

No load read as clean. On 2 September 2026 the record showed an agent that had compacted eight times without ever filing a load receipt. The notice had printed nothing, because it compared the latest load with the latest compaction, and an agent with no load at all passed as clean. A second notice now fires whenever a compaction exists and no load does.

The notice was read and ignored. On 3 and 4 September 2026 an agent took two compactions after its last load. The notice printed at the start of about forty consecutive turns, and every one was correct. The agent read them and edited the practice's canonical files anyway, including a republish of its state page. It is one episode, in one session, and the design changed because of it. The source code of the fix records the lesson by sorting remedies into three kinds: a rule (“re-load after a compaction”) “went 0-for-40”; a delivery, the notice at the top of every turn, “is strictly better and still lost every turn”; and “Only a CONSTRAINT acts whether or not the seat agrees.” The constraint added on 4 September is a refusal. While an agent's load is stale, the harness refuses its edits to the files that hold the practice's rules and records, and the commands that publish them, and names the load and the compaction in the refusal. Reading is never blocked, and reading again, with a new load receipt, is the only thing that clears it. File edits have no override. For the publishing commands the refusal names one, and asks the agent to say out loud why the change does not depend on what it has not re-read. Everything else, building, testing and ordinary edits, stays the agent's call.

The refusal writes no row to the event log, but the session transcripts record each one. Counted from them on 27 September 2026, it had refused 14 writes in five sessions between 11 and 25 September, five of them by agents those sessions had started. The check matches paths, so two of the 14 were edits to a script of the practice's website that sits in a folder named brain. A fifteenth, on 4 September, names a dummy path and is left out as its builder's test.

6.4 Counting compactions

Since 19 September 2026 the practice has counted its compactions and what followed each. Every compaction falls into one of three states: reloaded, if a load receipt for the same session exists after it; not reloaded, if none exists and the session has since closed; and unresolved, if none exists yet and the session is still open. The counter declines to call the last a failure, because “calling it a failure now would be counting a race as a result.” Read on 27 September 2026 at 16:53 UTC, over the live log and its archive, it counted 84 compactions: 57 reloaded, 18 not reloaded, and 9 unresolved, so 18 of the 75 settled compactions, 24.0%, were never followed by a reload (Figure 6). At the working paper's last revision it read 70, 43, 18 and 9 (§10.9 of the working paper). Since then the count of compactions and

the count of reloaded ones have both risen by fourteen, and the other two counts have not moved. One of the fourteen is the compaction of §5.3.

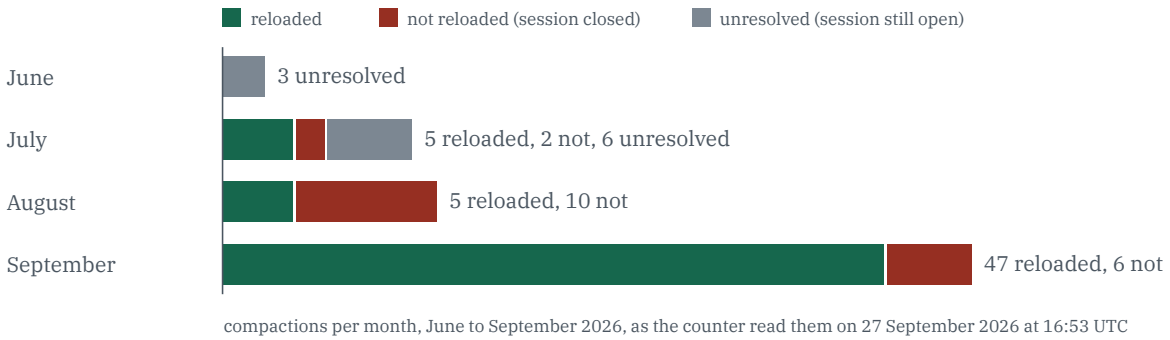

Figure 6. The practice's compaction counter by month. “Reloaded” means only that a load receipt came later in the same session; 20 of the 57 reloaded compactions were followed by at least one more compaction before that receipt came (see text). The nine unresolved all date from June and July. **Measured, read 27 September 2026.**

The counter is generous in one direction, and its reading should be taken with that in mind. “Reloaded” means only that a load receipt came later in the same session, not that it came promptly. Of the 57, 37 were reloaded before the session compacted again; the other 20 went through at least one more compaction first, and one receipt cleared eight at once. The monthly split still shows a direction: in August, 5 of 15 settled compactions were reloaded, and in September 47 of 53. The nine unresolved all date from June and July, from sessions that never recorded a close. The counter counts receipts, which are the agent's own declaration of what it read, and it cannot say how well the reading was done. That makes the 18 not reloaded a floor rather than an estimate: a receipt can be filed without a careful reading, but a missing one cannot be faked. It is not a catch rate, and nothing in it measures whether a compaction caused an error.

7 What anyone can do

The practice's defenses are specific to its harness, but the findings behind them apply to anyone who uses an AI tool for more than a short exchange. Table 1 lists the steps, why each helps, and what each cannot do.

Table 1. Working with a context that runs out. “Cannot” is the honest limit: what the step leaves untouched.

Do this	Why it helps	What it cannot do
Start a fresh conversation for a new task. If one has gone wrong, start again with all the requirements in one message.	Models do much worse when a task arrives in pieces over many turns and rarely recover from a wrong turn; the same requirements given at once did almost as well as the original instruction (Laban et al., 2025).	Carry over what the old conversation learned, unless it goes into the new first message

Do this	Why it helps	What it cannot do
Put instructions that must hold where the tool keeps standing instructions (project instructions, custom instructions, a CLAUDE.md file), not only in the chat.	What the harness reloads from files survives a compaction; what was said in the chat survives only in the summary (Anthropic, n.d.). In a 2026 test, a constraint appended after compaction was followed 99.7% of the time, against 31.8% when it had to survive in the summary (Wang et al., 2026).	Make a long instruction file well read; long contexts are used unevenly (§2)
Keep the source where the work is: in files, a progress log and the version history, not only in the conversation.	The summary is then not the only copy. A design that kept an address for everything it compacted recalled a planted fact 99.40% of the time, against 88.12% for the best alternative tested (Dang et al., 2026), and long-running agents pick up from a progress file and the version history (Anthropic, 2025b).	Make the agent read it; that takes a habit, a rule or a constraint (§6)
Before acting on something from earlier in a long session, ask for the source to be read again.	Models get less reliable as their input grows (Zeng, Huang & He, 2026); a copy of the task placed at the end restored accuracy lost in the middle of long contexts (Zhang et al., 2026); and compaction keeps only a small share of standing constraints (Wang et al., 2026).	Catch an error in the source itself
In any long session, assume the early part is summarized or gone, whether or not the tool says so, and say again what matters.	After a compaction, what the model knows of the earlier conversation is a model-written account of it (Anthropic, 2025a; n.d.), and a tool that drops the oldest turns keeps no account of them at all; not every tool says which it did (§4).	Bring back detail the summary dropped, unless the source still exists
If you build on these tools: count compactions and what follows them, and make re-reading a precondition for changing what governs the work, not a reminder.	In the practice, a rule and then a notice at the top of every turn both failed; a refusal is what changed the act (§6.3).	Show how well the re-reading was done; a receipt records what was read, not how

The post that goes with this paper gives the first, second, fourth and fifth of these in short form. The third and the last are for anyone building on these tools. For someone who sets up these tools for a team, the same steps become defaults: standing instructions in the shared project or workspace, and a new conversation for each new task as the norm rather than the exception.

8 Limits

The research ages quickly. Most of the evidence here is from 2025 and 2026, and the models it tests are already being replaced, so a practitioner should expect the sizes of these effects to differ by model. Two older studies stay for what they introduced rather than for their numbers: Liu et al. (2023), which named lost in the middle, an effect a 2026 reproduction with current open models did not find in its original form; and Maynez et al. (2020), whose distinction between intrinsic and extrinsic errors still describes how a summary goes wrong. Laban et al. tested the frontier models of early 2025. Most of the 2026 studies are preprints and most test open models; of the studies cited, only LOCA-bench tests current models from Anthropic, OpenAI and Google side by side.

Laban et al.'s conversations were simulated. The authors state that their simulations “are not representative of natural human-AI conversation”, that the tasks were analytical, and that the work was on English text only. They argue that the degradation they measured “is most likely an underestimate”.

The practice's evidence is one practice's. It comes from one operator, one model family and one harness, and from instruments the practice built and reads itself. The three cases of §5.3 were chosen because they illustrate the argument, not sampled. The forty-warnings episode of §6.3 is one agent in one session, and the count of forty is its builder's own. The counter counts what the hook recorded, and seven early rows came from other sources.

The practice's defenses have gaps of their own. The refusal writes no row to the event log when it fires, so its count in §6.3 comes from reading session transcripts, which is slower and easier to get wrong than a log: a first pass that allowed for only one line-ending style counted none. The hook that runs it sees the harness's own file and shell tools but not the connectors through which agents write to the practice's pages in Notion, its system of record, so a stale agent's edit there is not refused. The state file cannot say what an open item is. And the handoff error of §5.3 happened with every one of these defenses in place, because a handoff document is not one of the files the refusal protects. What caught it was the operator.

What would change the argument. Evidence that current compaction summaries preserve the meaning of their sources, measured at scale against the sources, would weaken §5. Evidence that models with the longest windows no longer lose accuracy as the input grows, or over many turns, would weaken §2 and §3. And evidence that a notice at the start of every turn changes agents' behavior as reliably as a refusal would weaken the design argument of §6.3, which rests on one practice's record.

How to cite this part. Atkinson, B. (2026). *What Is Context? What a Model Can Hold, and What Happens When It Runs Out of Room.* Background paper 9 to *How do we use this?* Working paper, Wolfberg LLC.

Disclosure. Drafting and literature synthesis were assisted by AI models (Claude, Anthropic) working under the author's direction; the author is responsible for the content. The works in the References were read in full, by AI reading agents working under the author's direction, and every quotation and number

this paper takes from them was then checked by the drafting model against the saved full text of its source. Pages from laboratories and vendors were read at the linked pages, Anthropic's two engineering posts and Chroma's report on 24 September 2026, and the Claude Code documentation, the Codex source file and the Gemini CLI documentation on 27 September 2026, and are current as of then, not permanent. The practice's records were read at their source on 26 and 27 September 2026: its event log, its rulebook, and the source code and commit history of its hooks. Every arXiv identifier below was resolved against arXiv on 27 September 2026 with its title and first author matched.

Competing interests. The author owns Wolfberg LLC, the practice studied. The practice's working model is from Anthropic's Claude family, the harness described in §4 and §6 is Anthropic's Claude Code, the drafting assistance used the same family, and several of the sources are Anthropic's own publications and documentation.

Data availability. The practice's logs contain client work and personal records. They are private and are not offered for sale or sharing. Figures from the practice are reported as of the dates given with them.

REFERENCES

Atkinson, B. (2026a). *How Do We Use This? Findings from running an AI team on real work for six months, and a test that could prove them wrong.* Working paper, Wolfberg LLC. wolfberg.ai/papers/how-do-we-use-this

Atkinson, B. (2026b). *What Is a Harness? The Software Around the Model, and Why It Decides So Much.* Background paper 8 to *How do we use this?* Wolfberg LLC. wolfberg.ai/papers/how-do-we-use-this/part-8

Dang, T., Ichikawa, Y., Fatima, S., & Shirahata, K. (2026). *Addressable recall compaction for long context-window control in AI agents.* arXiv:2607.25066. arxiv.org/abs/2607.25066

Gabín, J., Perez, A., & Parapar, J. (2026). *Lost in the evidence? Reproducing document position and context size effects in RAG.* In *Proceedings of SIGIR '26*. arXiv:2605.27105. arxiv.org/abs/2605.27105

Laban, P., Hayashi, H., Zhou, Y., & Neville, J. (2025). *LLMs get lost in multi-turn conversation.* arXiv:2505.06120. arxiv.org/abs/2505.06120

Lindenbauer, T., Slinko, I., Felder, L., Bogomolov, E., & Zharov, Y. (2025). *The complexity trap: Simple observation masking is as efficient as LLM summarization for agent context management.* arXiv:2508.21433. arxiv.org/abs/2508.21433

Liu, N. F., Lin, K., Hewitt, J., et al. (2023). *Lost in the middle: How language models use long contexts.* arXiv:2307.03172. arxiv.org/abs/2307.03172

Liu, S. (2026). *What does context compression cost an agent? Interaction costs unrevealed by task-completion metrics.* arXiv:2608.16370. arxiv.org/abs/2608.16370

Maynez, J., Narayan, S., Bohnet, B., & McDonald, R. (2020). *On faithfulness and factuality in abstractive summarization.* arXiv:2005.00661. arxiv.org/abs/2005.00661

Packer, C., Wooders, S., Lin, K., et al. (2023). *MemGPT: Towards LLMs as operating systems.* arXiv:2310.08560. arxiv.org/abs/2310.08560

Wang, Z., Zhang, Y., Lee, D., & Yang, Y. (2026). *Lost in compaction: Evaluating side-constraint loss under context compaction.* arXiv:2608.11242. arxiv.org/abs/2608.11242

Zeng, W., Huang, Y., & He, J. (2026). *LOCA-bench: Benchmarking language agents under controllable and extreme context growth.* arXiv:2602.07962. arxiv.org/abs/2602.07962

Zhang, C., Cui, H., Huang, X., & Sang, J. (2026). *Positional failures in long-context LLMs: A blind spot in reasoning benchmarks.* arXiv:2605.23170. arxiv.org/abs/2605.23170

LABORATORY AND VENDOR PAGES (READ AT THE LINKED PAGES, ON THE DATES GIVEN IN THE DISCLOSURE)

Anthropic. (2025a, September 29). *Effective context engineering for AI agents.* anthropic.com/engineering/effective-context-engineering-for-ai-agents

Anthropic. (2025b, November 26). *Effective harnesses for long-running agents.* anthropic.com/engineering/effective-harnesses-for-long-running-agents

Anthropic. (n.d.). *Explore the context window.* Claude Code documentation. Read 27 September 2026. code.claude.com/docs/en/context-window

Google. (n.d.). *CLI commands: /compress.* Gemini CLI documentation, docs/reference/commands.md. Read 27 September 2026. github.com/google-gemini/gemini-cli

Hong, K., Troynikov, A., & Huber, J. (2025, July). *Context rot: How increasing input tokens impacts LLM performance.* Chroma technical report. trychroma.com/research/context-rot

OpenAI. (n.d.). *slash_command.rs: the /compact command.* Codex source code, codex-rs/tui/src/slash_command.rs. Read 27 September 2026. github.com/openai/codex