AGENT ARCHITECTURE · CONTEXT ENGINEERING

Keeping the global agent lean: memory compression and sub-agents

LLM agents re-read an ever-growing context of thoughts, tool calls and tool results. This sheet maps two ways to keep the global agent's window bounded: compressing what is already inside (compaction, tool-result clearing, external memory, prompt compression) and isolating exploration in sub-agents that return condensed reports. It includes a token model and trade-offs.

Advanced · 12 min · context window · compaction · summarization · sub-agents

Sheet
by Simon-Pierre Boucher

sheet.spboucher.ai/s/keeping-the-global-agent-lean-memory-compression-and-sub-agents
November 17, 2026

An LLM agent re-reads its whole context at every step: the system prompt, the task, and every thought, tool call and tool result so far [1]. On long tasks this window, which is the global agent's working memory, fills with stale observations. Two families of techniques keep it bounded. **Compression** rewrites what is already in the window into fewer tokens. **Sub-agents** stop the noise from entering it at all: they run exploration in separate contexts and return only a condensed result [7].

Sources: 1 7

01 Why the global context fills up

WARNING

A full window is not only expensive

Models do not use long contexts uniformly. Liu et al. found that accuracy on multi-document question answering and key–value retrieval was highest when the relevant information sat at the beginning or the end of the input, and dropped when it sat in the middle [2]. In an agent loop, the facts that matter (the original goal, an early constraint) drift into exactly that middle as observations pile up. And once C_t reaches the window limit W, the loop cannot continue at all.

Sources: 2

APPEND-ONLY AGENT LOOP

$$C_t = C_0 + \sum_{i=1}^{t} (a_i + o_i)\,, \qquad \sum_{t=1}^{T} C_t = T\,C_0 + (a + o)\,\frac{T(T+1)}{2}$$

f_x Result derived in this sheet

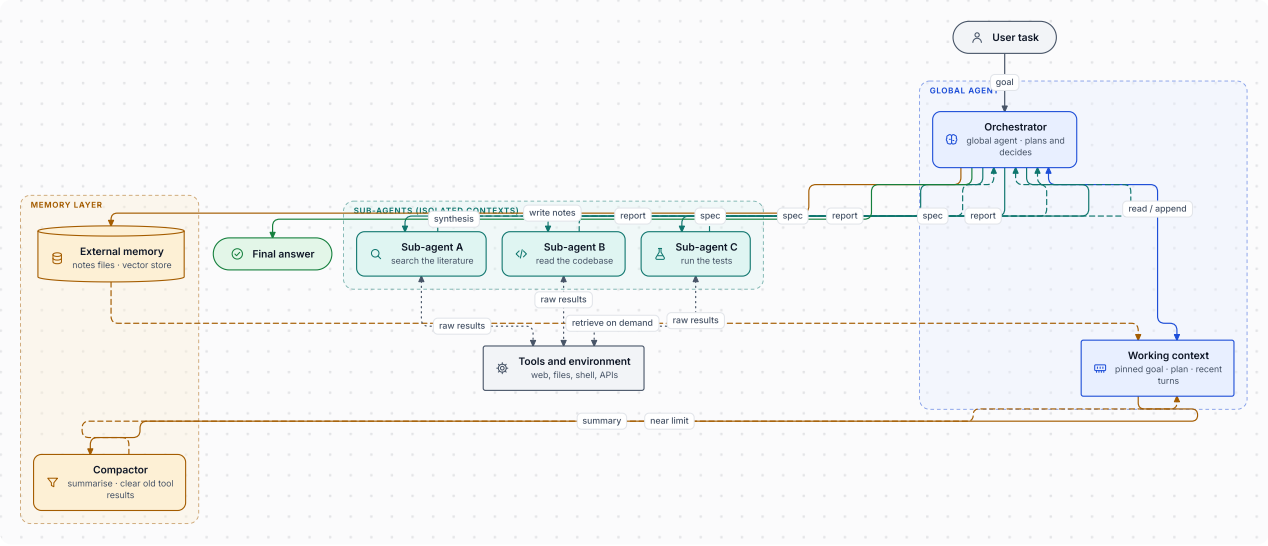
The first identity is bookkeeping for a loop that only appends, such as a ReAct-style loop. The second assumes a constant amount added per step ($a_i = a$, $o_i = o$). Under that assumption the context grows linearly in T, and the total number of prompt tokens read over the run grows quadratically. Tool results (file contents, web pages, logs) usually make o much larger than a. Prompt caching lowers the price of re-reading a stable prefix. It does not shrink the window, and it does not reduce the number of tokens the model must attend over.

C_t	Tokens in the agent's context at the start of step t + 1	tokens
C_0	Initial context: system prompt, tool definitions, task statement	tokens
a_i	Tokens generated at step i (reasoning and the tool call)	tokens
o_i	Tokens of the tool result (observation) returned at step i	tokens
a, o	Constant per-step averages of a_i and o_i (simplifying assumption)	tokens
T	Number of steps in the run	

Sources: 1

02 Two levers: compress the window, or keep noise out of it

Orchestrator with a memory layer and isolated sub-agents



Global agent context · Compression and external memory · Delegation: spec out, condensed report back

Raw tool traffic, never enters the global context

1 **Task arrives**
— The orchestrator loads the goal into its working context and pins it there.

2 **Delegate**
— Separable subtasks go out as self-contained specs, in parallel when they are independent.

3 **Explore in isolation**
— Each sub-agent runs its own loop. Bulky tool output fills its own window, not the orchestrator's.

4 **Return condensed reports**
— Only findings, evidence pointers and open questions come back.

5 **Compact and persist**
— Near the threshold, old turns are summarised and old tool results cleared. Decisions are written to external notes and retrieved when needed.

6 **Synthesise**
— The orchestrator merges the reports and produces the answer.

Raw tool traffic (dotted) stays inside the sub-agents' contexts. Only task specs and condensed reports (teal) cross into the global context, and the compactor and external memory (amber) bound what remains.

f THRESHOLD-TRIGGERED COMPACTION

$$C \leftarrow C_0 + s + r \quad \text{whenever } C > \theta W, \qquad \rho = \frac{C - C_0 - r}{s}$$

f_x **Result derived in this sheet**

Between compactions the context grows as in the append-only model. After each compaction it falls back to about $C_0 + s + r$, so the context follows a sawtooth bounded by θW. Choosing $\theta < 1$ leaves room for the summarisation call itself. A larger compression ratio ρ frees more space and loses more detail. Clearing tool results first is the cheapest reduction, and it can be undone if the raw output is stored.

W	Context window of the model	tokens
θ	Trigger fraction of the window, 0 < θ < 1	
s	Length of the summary that replaces older turns	tokens
r	Recent turns kept verbatim after compaction	tokens
ρ	Compression ratio of the summarised segment	

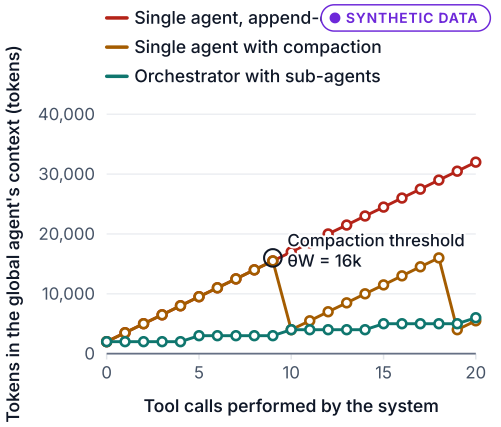
03 Compression: rewrite the window into fewer tokens

Compression techniques differ mainly in what they keep in the window and whether what they drop can be recovered.

TECHNIQUE	KEEPS IN THE WINDOW	DROPS	MAIN RISK
Rolling summarisation (compaction)	A model-written summary of older turns, plus the most recent turns verbatim [7]	Verbatim history	Details the summariser judged unimportant are lost for good; summaries of summaries drift
Tool-result clearing	The tool call and a placeholder or pointer to the stored output [7]	Bulky old observations	The agent must re-run or re-read the tool if it needs the data again
External memory with paging (MemGPT-style)	A small main context; the agent itself moves information in and out of external storage [3]	Nothing permanently: data lives outside the window	Retrieval misses; extra calls spent managing memory
Structured notes / scratchpad files	Agent-written plan, decisions and open questions, re-read when needed [7]	The raw trace	Notes are only as good as the agent's discipline in writing them
Memory stream with reflection	Memories retrieved by recency, importance and relevance, plus higher-level reflections [4]	Low-scoring memories (they stay in the store)	Reflections can encode wrong inferences that later steps trust
Token-level prompt compression (LLMLingua)	Tokens that a small language model judges informative [5]	Low-information tokens	Lossy for exact strings such as code, identifiers and numbers

Sources: 7 3 4 5

Global context size under the three strategies



Synthetic, computed from the token model above: $C_0 = 2000$ and $a + o = 1500$ tokens per tool call. Compaction triggers at $\theta W = 16000$ and falls back to 4,000 tokens. The orchestrator delegates blocks of 5 tool calls with $d_k + r_k = 1000$. Each sub-agent's own context peaks near 9,000 tokens before being discarded.

04 Sub-agents: context firewalls for exploration

ƒ ORCHESTRATOR VERSUS SUB-AGENT CONTEXT

$$C^{\text{orch}} = C_0 + \sum_{k=1}^{K} (d_k + r_k), \qquad C_k^{\text{sub}} = C_0' + d_k + \sum_{i=1}^{n_k} (a_{k,i} + o_{k,i})$$

f_x Result derived in this sheet

The orchestrator pays $d_k + r_k$ tokens per subtask, however many steps n_k the sub-agent took. The cost of exploration lives in a context that is discarded once the report is returned. The total spend of the system is the sum over all contexts. Delegation therefore lowers the peak load on any single window, but it usually raises the total number of tokens processed.

C^{orch}	Orchestrator context after K delegated subtasks (its own reasoning turns omitted)	tokens
C_k^{sub}	Context of sub-agent k at the end of its subtask	tokens
K	Number of delegated subtasks	
d_k	Task specification sent to sub-agent k	tokens
r_k	Condensed report returned by sub-agent k	tokens
C_0'	Sub-agent system prompt and tool definitions	tokens
n_k	Steps taken by sub-agent k	
$a_{k,i}, o_{k,i}$	Generated tokens and observation tokens at step i of sub-agent k	tokens

Decompose
Split the goal into subtasks that share little state, and run them in parallel when they are independent.

Write a self-contained spec
State the objective, scope boundaries, allowed tools, output format and a stopping criterion. The sub-agent sees none of the orchestrator's history, so d_k must carry everything it needs.

Explore in isolation
The sub-agent runs its own loop. Its $\sum_i o_{k,i}$ of raw output stays in its own window.

Return a condensed report
Return findings, evidence pointers (paths, URLs, line numbers), confidence and open questions, within a budget of r_k tokens.

Integrate and verify
Merge the reports, resolve conflicts, spot-check the pointers, and record decisions in external notes.

Sources: 6 7

- Keep the goal, the constraints and the current plan in a pinned prefix that compaction never rewrites.
- Clear old tool results before summarising anything else, and keep a pointer to the stored raw output.
- Trigger compaction at θW well below the window, leaving room for the summarisation call.
- Give every sub-agent a self-contained spec d_k and an explicit report budget r_k.
- Ask for evidence pointers in every report, and verify a sample of them.
- Evaluate task success and total tokens, not only the peak size of the global context.

Compression and delegation attack the same problem at different points in the loop.

	Compression in one context (LEVER 1)	Delegation to sub-agents (LEVER 2)
WHERE THE NOISE GOES	Enters the window and is removed later	Never enters the global window
GLOBAL CONTEXT GROWTH	Sawtooth bounded by θW	$d_k + r_k$ per subtask, independent of n_k
TOTAL TOKENS	Lower: one context, plus the summarisation calls	Higher: Anthropic reports that its multi-agent research system used about 15× the tokens of a chat interaction [6]
LATENCY	Sequential; each compaction adds a model call	Independent subtasks can run in parallel
WHO DECIDES WHAT IS LOST	The summariser	The sub-agent writing the report; the orchestrator never sees the rest
COORDINATION COST	None	Spec quality, duplicated work, conflicting findings
BEST FIT	Long sequential work on tightly coupled state	Breadth-first work with separable subtasks; less suited to tasks with heavy shared context and dependencies, such as most coding work [6]

VERDICT

The two levers compose. An orchestrator can delegate exploration to sub-agents and still compact its own history on very long runs. Delegate when subtasks are separable; compress when the state is tightly shared.

Sources: 6

Summary drift

Repeated compaction summarises summaries, so early constraints get paraphrased away. **Mitigation:** pin the goal and constraints in a section that compaction never touches, and keep a verbatim decision log in external notes.

Irrecoverable exact details

Identifiers, paths and numbers are the first things a summary loses. **Mitigation:** store raw tool outputs outside the window and keep pointers in context. Clearing with a pointer preserves recoverability; summarising does not.

Telephone effect in delegation

A report is a lossy compression, and it can be wrong. The orchestrator cannot check what it never saw. **Mitigation:** require evidence pointers in every report, and spot-check them before acting on a finding.

Under-specified delegation

Vague specs lead sub-agents to duplicate each other's work or misread the scope [6]. **Mitigation:** give each spec an explicit objective, boundaries, output format and stopping criterion.

Cost explosion

Every sub-agent re-reads its own prompt and history, so total tokens scale with the number of sub-agents and their steps. **Mitigation:** scale the number of sub-agents and their step budgets to the difficulty of the task.

Sources: 6

QUESTION

An orchestrator delegates $K = 4$ subtasks. Each sub-agent makes 30 tool calls averaging 2,000 observation tokens each. Specs are $d_k = 300$ tokens and reports are $r_k = 700$ tokens. By how many tokens does the orchestrator's context grow?

A. 4,000 tokens

B. 240,000 tokens

C. 244,000 tokens

D. 1,000 tokens

SHOW THE ANSWER

A — Correct. $\sum_k (d_k + r_k) = 4 \times (300 + 700) = 4000$. The sub-agents' observations never enter the orchestrator's window.

B — Incorrect. That is $4 \times 30 \times 2000$, the observation tokens spread across the four sub-agent contexts, which are discarded after each report.

C — Incorrect. This adds the sub-agents' observations to the specs and reports. That is the growth you would get without delegation, plus overhead.

D — Incorrect. That is the cost of a single subtask; there are $K = 4$ of them.

References

[1] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, Yuan Cao (2023). *ReAct: Synergizing Reasoning and Acting in Language Models*. International Conference on Learning Representations (ICLR). arxiv.org/abs/2210.03629 UNVERIFIED

[2] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, Percy Liang (2024). *Lost in the Middle: How Language Models Use Long Contexts*. Transactions of the Association for Computational Linguistics. arxiv.org/abs/2307.03172 UNVERIFIED

[3] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, Joseph E. Gonzalez (2023). *MemGPT: Towards LLMs as Operating Systems*. arXiv. arxiv.org/abs/2310.08560 UNVERIFIED

[4] Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, Michael S. Bernstein (2023). *Generative Agents: Interactive Simulacra of Human Behavior*. Proceedings of the ACM Symposium on User Interface Software and Technology (UIST). arxiv.org/abs/2304.03442 UNVERIFIED

[5] Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, Lili Qiu (2023). *LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models*. Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP). arxiv.org/abs/2310.05736 UNVERIFIED

[6] (2025). *How we built our multi-agent research system*. Anthropic Engineering. www.anthropic.com/engineering/built-multi-agent-research-... UNVERIFIED Engineering blog post; check the token-usage figure and the remarks on task fit against the published text.

[7] (2025). *Effective context engineering for AI agents*. Anthropic Engineering. www.anthropic.com/engineering/effective-context-engineeri... UNVERIFIED
Engineering blog post describing compaction, tool-result clearing, structured note-taking and sub-agent architectures.