PACT: Harness-Side, Cache-Friendly Context Management for Long-Horizon Agents

Pengcheng Xu et al.
University of California, Irvine

November 2026

Abstract

Long-horizon agents outgrow their context window. Existing remedies summarise or drop history, losing evidence, or let the agent manage its own context, which spends turns and forces prompt recomputation under prefix caching. We present PACT, a harness-side policy that spends no agent turn on context management. When the transcript crosses a threshold, PACT moves tool results verbatim into files in the agent's workspace and leaves short stubs that are never edited again, so the prompt changes rarely and the prefix cache stays valid. A variant, commit-with-recall, answers from the archive near the end of a token budget. On held-out BrowseComp-Plus queries, pre-registered tests find PACT non-inferior to the self-managed VISTA with 51% fewer uncached prompt tokens. At VISTA's largest budget it is more accurate over two pooled held-out sets (0.53 vs. 0.48; one set fell short) at half the cost, and at the default budget with commit-with-recall (0.54 vs. 0.43). Non-inferiority and the saving hold on three more backbones, up to 550B parameters, and on GAIA's hidden test set. On LOCA-bench, PACT is not detectably less accurate than VISTA at three data scales while the server recomputes 28–36% fewer prompt tokens, although a cache-blind count ranks it as more expensive.

Code: `https://anonymous.4open.science/r/pact-acl26`

1 Introduction

Language agents that search, browse or analyse data call tools for dozens of steps, and their transcripts soon exceed the context window (Zeng et al., 2026; Chen et al., 2026). From then on, something must leave the context, and every current remedy pays for it. Truncation and summarisation lose evidence (Lindenbauer et al., 2025; Wu et al., 2026a). Policies that let the agent manage its own context spend the agent's turns on it (Xu et al., 2026a; Zhang et al., 2026b; Sun et al., 2026). Any policy that edits the prompt also pays at the server, because a prefix cache can reuse only what precedes the first changed token (Kwon et al., 2023). An edit early in a long prompt therefore makes the server compute everything after it again.

A recent self-managed method, VISTA (Xu et al., 2026a), shows the agent a dashboard of its context and gives it tools to archive and restore blocks. On LOCA-bench these tools take 7–16.5% of VISTA's tool turns. Because the dashboard is re-sent as the last message of every prompt, VISTA also changes its prompt on 26.3 of its 27.3 steps at the largest scale. Evaluations that report total or peak tokens (Kang et al., 2026; Zhang et al., 2026b; Sun et al., 2026) cannot see this second cost, and

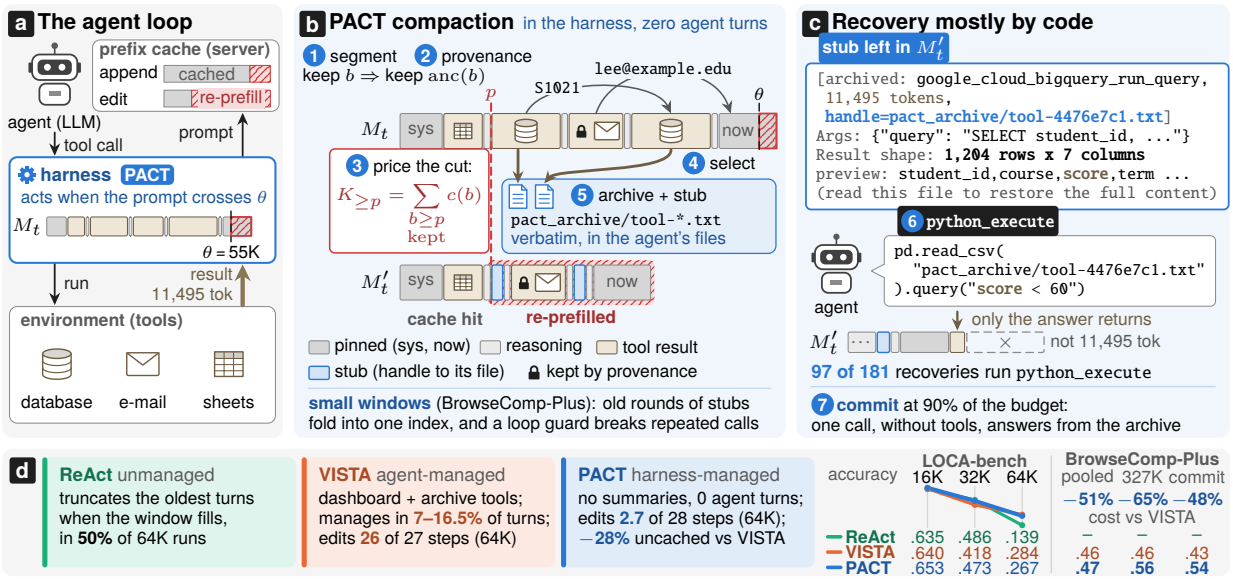

Figure 1. PACT manages context in the harness, archives instead of summarising, and edits the prompt rarely. (a) The harness sits between the agent and the server and compacts the transcript when the prompt crosses the threshold θ (55,000 tokens under LOCA-bench's 60,000-token limit), so the agent spends no turn on it. (b) Compaction segments the transcript into blocks (1), keeps the provenance ancestors of every kept block (2), prices each cut position p by the kept tokens $K_{\geq p}$ the server must prefill again (3), selects blocks to evict (4), and writes them verbatim to files, leaving stubs that are never edited again (5). (c) Agents recover evidence mostly by running code on an archived file, so only its printed output enters the context (6). Near the end of a token budget, one call without tools answers from evidence recalled from the archive (7). (d) The three policies, LOCA-bench accuracy by data scale (32K pooled over two runs), and held-out BrowseComp-Plus accuracy with PACT's change in uncached prompt tokens against VISTA (three sets pooled, the largest budget of VISTA's sweep, and commit-with-recall; §5.1).

VISTA and TokenPilot (Xu et al., 2026b) price cache hits per task at API rates. We instead count, for every method and every step on one server, the prompt tokens the server computes.

Our approach rests on two observations. First, if nothing removed from the context is paraphrased and all of it stays readable, deciding what to remove needs no judgment from the agent, because a wrong removal costs one read instead of a lost fact. Second, under a prefix cache the cost of context management is governed by how often and how early the prompt changes (§3.1). PACT turns these observations into a policy that manages context in the harness, archives instead of summarising, and edits the prompt rarely (Figure 1). It spends no agent turn. When the transcript crosses a threshold, it moves tool results and reasoning verbatim into files in the agent's workspace and leaves stubs that name the call, the shape of the result and a preview. Stubs are never edited again, so between two compactions the prompt only grows and the cache stays valid. Agents read archived evidence back mostly by running code on the file, which returns only what the code prints. When a token budget is nearly spent, *commit-with-recall* rebuilds the answering context from the archive and asks once, without tools, for the answer.

We compare PACT with VISTA run from its authors' code and charge every policy for its own model calls. On BrowseComp-Plus (Chen et al., 2026), in pre-registered tests on held-out queries, PACT is non-inferior to VISTA over 700 queries with 51% fewer uncached prompt tokens, and more accurate than the VISTA authors' three ReAct baselines. At the largest budget of VISTA's sweep it

is more accurate than VISTA on one held-out set (0.56 vs. 0.46); a pre-registered replication on a second set was not met (0.51 vs. 0.49), and over both sets it is more accurate (0.53 vs. 0.48) at under half the serving cost. With commit-with-recall it is also more accurate at the default budget (0.54 vs. 0.43), and no run ends out of budget. Non-inferiority and the saving hold on gpt-oss-20b, Claude Haiku 5.5 and the 550B-parameter Nemotron-3-Ultra. On LOCA-bench (Zeng et al., 2026), PACT is not detectably less accurate than VISTA at any of three data scales while the server recomputes 28–36% fewer prompt tokens. A cache-blind count reverses this ranking, because PACT sends more tokens but rarely edits its prompt (§5.4).

Running every method at three scales of the same tasks also shows when context management helps at all. It helps once the unmanaged agent's peak prompt exceeds the context limit, where it adds about 15 points, and has no detectable effect below that. A method evaluated at a single operating point can therefore look either useless or essential.

Our contributions are as follows.

- **Method.** PACT, a training-free, harness-side context policy that archives verbatim, edits the prompt rarely, and answers from its archive near the end of a budget (§3).
- **Evidence.** A pre-registered comparison on held-out queries, GAIA's hidden test set and four backbones with VISTA, run from its authors' code. PACT is non-inferior at about half the serving cost, and more accurate at a larger budget or with commit-with-recall (§5.1, §5.3).
- **Measurement.** A per-step account of every method's cache cost, checked against the server, under which the cache-blind ranking reverses (§5.4), and a within-task finding that the benefit of context management depends on where the unmanaged agent's prompt sits against the limit (§5.5).

2 Related Work

Context management policies. Fixed rules clear old tool results or reasoning (Zeng et al., 2026) or evict agent-annotated episodes by priority (Semenov and Dorofeev, 2026). Summarisation rewrites history (Wu et al., 2026a), with guidelines that can be optimised from failures (Kang et al., 2026), and observation masking replaces old observations with placeholders, matching summarisation's solve rate at about half an unmanaged agent's cost (Lindenbauer et al., 2025). Agent-managed policies expose context state and tools to the model (Xu et al., 2026a; Lin et al., 2026; Li et al., 2026a; Shao et al., 2026), keep task state outside the agent's context (Ma et al., 2026), or train the agent's editing decisions (Zhang et al., 2026b; Pan et al., 2026; Li et al., 2026b; Wu et al., 2026b; Sun et al., 2026); a separate manager can also be adapted to the agent (Yi et al., 2026). MemAct argues that editing belongs in the agent's policy, optimised end to end (Zhang et al., 2026b). PACT gives up that optimisation in exchange for a policy that needs no training, works with any backbone and spends no agent turn deciding what to remove. Its agents do spend turns reading evidence back (181 reads in 75 episodes at 64K).

Prefix caching and prompt edits. In serving stacks built on vLLM's paged KV cache (Kwon et al., 2023), prefix caching lets a request reuse the cache up to its first changed token. In a hybrid model, whose recurrent layers resume only from a saved state, reuse stops at the last such state (Pan et al., 2025). Prompt caching substantially cuts the API cost of agents when the prefix stays stable, and dynamic content in the prompt breaks it (Lumer et al., 2026). Harness-side methods keep removed content recoverable and time their edits around the cache (Xu et al., 2026b; Hao et al., 2026; Yao et al.,

2026; Su et al., 2026). CliffCompaction compacts rarely and never compacts a compaction again, but it truncates or drops old content, which can then be recovered only by running the tool again (Nguyen et al., 2026). Runtime context assembly can lower token counts while also lowering the cache-hit ratio (Xu et al., 2026c), a trade-off that a per-step cache account makes visible. KV-cache compaction (Liu et al., 2026b), parallel summarisation (Cim et al., 2026), tool-schema compression (Sakizli, 2026) and tool-output pruning (Lodha et al., 2026) reduce the same bill elsewhere (see Zhou et al., 2026, for a survey).

Budget-aware answering. Budget notices tell the agent how much budget remains or force an answer from the current context (Wang et al., 2024; Liu et al., 2026a; Fang et al., 2026). Commit-with-recall instead rebuilds the answering context from the verbatim archive, so the answer can use evidence that compaction removed from view.

Positioning. Placeholders (Lindenbauer et al., 2025), recoverable archives (Xu et al., 2026a,b), infrequent compaction (Nguyen et al., 2026) and loop detectors (Wang et al., 2025) each appear in prior work. PACT combines them in the harness with provenance-aware eviction and recall at answer time. It asks an open question: does a policy that spends no agent turn and rarely edits its prompt give up accuracy against the self-managed state of the art, run from its authors' code, when every policy is charged for its own calls on one server?

3 Method

3.1 Problem Formulation and Cost Model

Agent loop. An agent solves a task q in steps $t = 1, \dots, T$. At step t the harness sends the prompt $P_t = [S; \mathcal{T}; q; M_t]$, where S is the system prompt, $\mathcal{T}$ the tool schemas and M_t the transcript of earlier steps. The model returns an action a_t (reasoning, then a tool call or a final answer) and the tool returns an observation o_t. A context policy π may rewrite the transcript between steps, $M_{t+1} = \pi(M_t \circ a_t \circ o_t)$, where $\circ$ denotes concatenation and an unmanaged agent uses the identity. Every prompt must fit a context limit, $|P_t| \le B$ tokens, and an episode has at most H steps. On BrowseComp-Plus a run also has a token budget G on the new content it consumes, the tokens the model generates plus the tool results it ingests; the agent never sees it.

Transcript blocks. The transcript is a sequence of blocks $M_t = (b_1, \dots, b_n)$, the smallest spans that can be removed while the chat format stays valid, such as the payload of one tool result or one span of reasoning. Block b has $c(b)$ tokens and starts at position $\mathrm{pos}(b)$ of the prompt. The system prompt, the task and the last turn are *pinned*. An evicted block is replaced by a *stub* $\sigma(b)$, and a block is *prunable* if it is not pinned and evicting it frees at least 200 tokens, $c(b) - c(\sigma(b)) \ge 200$.

Prefix-cache cost. A server with a prefix cache keeps the key-value state of the previous request together with its completion, $Q_{t-1} = P_{t-1} \circ a_{t-1}$. Let p_t be the first position at which P_t differs from Q_{t-1}, or $|Q_{t-1}|$ if Q_{t-1} is a prefix of P_t. Under an attention-only cache the server reuses the first p_t tokens and computes

$$U_t = |P_t| - p_t = N_t + K_t \tag{1}$$

uncached tokens, where N_t counts the tokens that are new at step t (the observation, any stub or status message written then) and K_t the tokens after p_t that were already cached and must be prefilled again. A step is an *append* if $p_t = |Q_{t-1}|$, so $K_t = 0$, and an *edit* otherwise. On a hybrid model whose recurrent layers resume only from states saved at some positions, the server restarts

from the last saved state before p_t and computes more (Appendix N). The cost of a policy is $\sum_t U_t$ with its own model calls included, and its *cost per correct answer* is that sum over all tasks divided by the number of tasks solved.

Edit frequency bounds the cost. New content is computed once under any policy, so a policy controls cost through $\sum_t K_t$, which only edits incur.

Proposition 1. *Let a policy leave the prompt unchanged unless appending a step would take it above θ tokens, and let every change leave at most $\kappa < \theta$ tokens. If an episode appends $\Lambda = \sum_t (|a_t| + |o_t|)$ tokens, its number C of compactions, which bounds its edits, and its re-prefilled tokens satisfy*

$$C \le 1 + \frac{\Lambda}{\theta - \kappa}, \qquad \sum_t K_t \le \kappa\, C. \tag{2}$$

Proof sketch. Steps without a compaction only append. After a compaction the prompt holds at most κ tokens, so the next one needs more than $\theta - \kappa$ appended tokens, and a compaction re-prefills only tokens it keeps (full proof in Appendix B). □

Hence $\sum_t U_t \le \sum_t N_t + \kappa(1 + \Lambda/(\theta - \kappa))$ whatever the number of steps, while a policy that edits every step incurs at least d re-prefilled tokens per step for the d cached tokens kept after each first change (Appendix B). PACT changes the prompt only at compactions, since a stub frees too little to be evicted again, so the bound applies whenever a compaction brings the prompt below θ.

3.2 Harness-Side Compaction

PACT runs after the tool results of a step arrive, when $|M_t| + |\mathcal{T}|$ exceeds the threshold θ (Algorithm 1).

Provenance closure. A lexical extractor maps each block to the identifiers it mentions, such as record ids, paths and e-mail addresses. A block *defines* an identifier it mentions first and *references* identifiers defined earlier, so every edge of the induced graph points backwards and the graph is acyclic. An evicted set S must be closed, which means that with each block it contains every prunable block that references it within three hops. Equivalently, a kept block keeps every block it references within three hops, so no kept block points at evidence that left the context, and evicted evidence stays in the archive.

Cut selection. An edit's first change is at $p(S) = \min_{b \in S} \mathrm{pos}(b)$, and it re-prefills the kept tokens after it, $K_{\ge p}(S) = \sum_{b \notin S,\ \mathrm{pos}(b) \ge p(S)} c(b)$. A wrongly evicted block costs about $r(b) = c(b) + \rho$ tokens to recover (one read, with $\rho = 400$ for the call), and $\hat{n}(b) \in [0, 1]$ forecasts whether it will be needed soon. PACT chooses

$$\begin{aligned} S^\star \in \arg\min_{S \text{ closed}} \ & \Gamma\, K_{\ge p}(S) + \textstyle\sum_{b \in S} \hat{n}(b)\, r(b) \\ \text{s.t.} \ & |P'_t| - |\mathcal{T}| \le \tau, \end{aligned} \tag{3}$$

where $\Gamma = 0.25$ prices a re-prefilled token against a recovered one, P'_t is the compacted prompt, and $\tau = 0.45(\theta - |\mathcal{T}|)$ leaves room for several steps. The forecast samples $k = 4$ short continuations of the agent that each predict its next four actions; $\hat{n}(b)$ is the fraction that mention an identifier b defines, propagated through the provenance graph for up to three hops. The search is greedy. For each of the 64 latest prunable positions p, it fills S from blocks at or after p in increasing $\hat{n}(b)\, r(b)/c(b)$, each with its closure, until the constraint holds, and keeps the cheapest candidate. When no candidate meets τ, it evicts every prunable block. This fallback is common at 64K, where

Algorithm 1 One step of PACT (harness side; the agent is not consulted). LOCA-bench settings: θ=55K, Γ=0.25, ρ=400, α=0.10.

Require: transcript M_t, tool schemas $\mathcal{T}$, archive A
answer a call that repeats a read-only call archived in A from A
if $|M_t| + |\mathcal{T}| \le \theta$ **then return** M_t
$\mathcal{U} \leftarrow$ prunable blocks outside the last turn
$\hat{n} \leftarrow$ forecast from k sampled continuations
$\tau \leftarrow 0.45\,(\theta - |\mathcal{T}|)$
for each of the 64 latest prunable positions p **do**
    $S_p \leftarrow$ greedy closed set from blocks at or after p until $|P'_t| - |\mathcal{T}| \le \tau$
    $\mathrm{cost}(p) \leftarrow \Gamma K_{\ge p}(S_p) + \sum_{b \in S_p} \hat{n}(b)\, r(b)$
end for
$S \leftarrow S_{p^\star}$ with $p^\star = \arg\min_p \mathrm{cost}(p)$; if no S_p meets τ, $S \leftarrow \mathcal{U}$
keep blocks whose $\hat{n}$ exceeds the risk-controlled threshold at level α, if B allows
write each $b \in S$ verbatim to A and replace it by its stub $\sigma(b)$
reselect the tool schemas shown if the cut keeps cached at most a third of what it re-prefills
truncate the oldest turns if $|M'_t| + |\mathcal{T}| > B$ ▷ lossy fallback
return M'_t

one tool result can fill much of the window. The median compaction there re-prefills 34,839 tokens, more than τ itself, and never cuts at the start of the transcript.

Verbatim archive and stubs. Each evicted block is written verbatim to a file in `pact_archive/` in the agent's workspace and replaced by a stub naming the call, its arguments, the shape of the result and a 600-character preview (Figure 1c). A wrong eviction costs one file read or one code query. Two rules keep the harness cheap. A read-only call that repeats an archived call is answered from the archive, and when a cut would leave cached at most a third of what it re-prefills, PACT also reselects which tool schemas are shown, since that then costs little more (56% of compactions at 64K). The scheme is not strictly lossless. When evicting every prunable block cannot bring the prompt under B, the harness truncates the oldest turns. This happens in 7% of PACT's episodes at 64K, against 50% of the unmanaged agent's.

3.3 Commit-with-Recall

Under the benchmark's rule, a run that spends its token budget before answering is wrong. Once the meter reaches $0.9\,G$, PACT therefore ends the research phase itself. It splits every tool result of the run, all of which it keeps verbatim, into overlapping 1,200-character passages, ranks them by BM25 against the question and the agent's latest notes, drops exact duplicates, and fills the window with the task, the notes and the best passages. One call without tools then asks the model to commit to its best answer, which lands inside the budget. If the budget runs out first, the closing call that the client makes receives the same recalled context instead of what survived compaction.

3.4 Small-Window Extensions

Two mechanisms, developed on a separate set of development queries and off on LOCA-bench, handle windows that one tool result can nearly fill and greedy decoding. *Folding.* When stubs alone keep the prompt over the threshold, the oldest rounds whose results are all stubs are replaced, in one batch, by an index with one line per call (its arguments and archive path), down to 60% of the window. *Loop guard.* Under greedy decoding an agent that returns to an earlier state tends to repeat

itself, so when a call returns the same result a third time within eight calls, the harness appends a one-line notice and samples the next step once at temperature 0.7. On LOCA-bench PACT runs Algorithm 1 in full. On BrowseComp-Plus it drops the forecast ($\hat{n} \equiv 0$) and adds folding and the loop guard, and commit-with-recall is a separate, pre-registered variant.

4 Experimental Setup

Benchmarks. *BrowseComp-Plus* (Chen et al., 2026) asks questions that require many searches over a fixed corpus of 100K web documents. We use the VISTA authors' client and dense retriever with a 32,768-token window, their default budget $G = 163{,}840$ and greedy decoding; θ is the same fraction of the window as on LOCA-bench. Folding and the loop guard were developed on 30 queries, with 100 more for diagnosis, none of them in a held-out set. The benchmark's judge grades the answers. Its *strict* rule counts a run that exhausts the budget as wrong, and the VISTA authors' *lenient* rule grades the answer the client forces at that point. *LOCA-bench* (Zeng et al., 2026) ships 75 task configurations (15 families, five seeds each) at three environment scales, 16K, 32K and 64K, that differ only in how much data the environment holds, so comparisons across scales are paired by task. Every method there has $B = 60{,}000$, $H = 35$ and the same sandboxed tool servers, and PACT compacts at $\theta = 55{,}000$ (Appendix P).

Models. The main backbone is Qwen3.8-27B served by vLLM with prefix caching, at temperature 1.0 on LOCA-bench. On BrowseComp-Plus we add gpt-oss-20b, served the same way, Nemotron-3-Ultra (550B parameters, 55B active) and Claude Haiku 5.5, reached through hosted APIs that report no cache counts, so their cost is counted in total tokens (Appendix A).

Baselines. VISTA (Xu et al., 2026a) runs from its authors' repository with the dashboard their paper reports. On LOCA-bench we add two unmanaged agents built on the benchmark's own ReAct runner (Yao et al., 2023). *ReAct* (*ours*) drops the oldest turns when a prompt would overflow, and *ReAct* (*authors'*), the VISTA authors' launcher, ends an episode when its transcript overflows. On BrowseComp-Plus we add the VISTA authors' three ReAct baselines, which truncate, clear or mask old tool results. No baseline is reimplemented. Our changes to the authors' code add a sandbox, the cost instrument, a longer request timeout and the step limit, and leave their policies untouched (Appendix A).

Cost metrics. The server reports how many prompt tokens of each request it served from its cache, and the rest are the uncached tokens U_t. Our instrument computes the same quantity from the prompts themselves under an attention-only cache, which does not depend on what else the server is running; we call its count *re-prefilled tokens*. Both counts include each policy's own model calls.

Statistical analysis. Comparisons are paired on the tasks both methods completed. We use McNemar's exact test for correctness, a paired t test for token counts and bootstrap intervals over tasks for ratios. A method is non-inferior at a 0.10 margin if the one-sided 95% Wald lower bound of the paired accuracy difference exceeds -0.10. For nulls we give the minimum detectable effect (MDE, 80% power). Every BrowseComp-Plus test below, and the second LOCA-bench run at 32K, was fixed in a written plan (queries, analysis script and decision rule) before its data existed, with Holm's correction over each family of tests. The appendix gives the full statistics.[1]

[1] Code, configurations and the time-stamped plans: `https://anonymous.4open.science/r/pact-acl26`.

Method	Acc.	Uncached	Unc./correct	Cycles
PACT	0.51	211K	0.41M	25/93
VISTA (authors')	0.48	392K	0.82M (1.97×)	45/99
ReAct, truncate (authors')	0.39	252K	0.66M (1.58×)	50/114
ReAct, clear (authors')	0.40	472K	1.18M (2.85×)	62/116
ReAct, mask (authors')	0.40	464K	1.17M (2.84×)	65/117

Table 1. Second held-out BrowseComp-Plus set (200 queries, strict rule). Uncached prompt tokens per query (server count, each policy's own calls charged), uncached tokens per correct answer with its ratio to PACT (the 95% bootstrap interval excludes parity for every baseline), and budget-exhausted runs that end in an exact cycle of repeated calls.

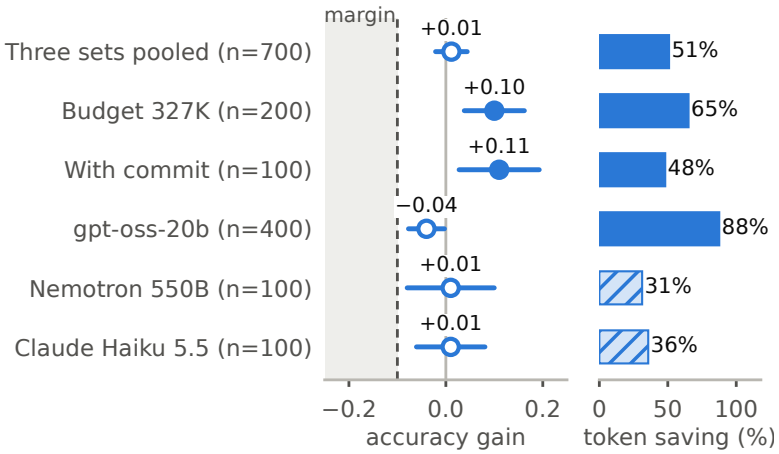

Figure 2. PACT against VISTA on held-out BrowseComp-Plus. Each row is a pre-registered comparison: the three held-out sets pooled, the second set at the largest budget of VISTA's sweep, the first set with commit-with-recall, and three more backbones. Left, the paired accuracy difference (strict rule) with its 95% interval; filled markers exclude zero, and the dashed line is the non-inferiority margin. Right, PACT's saving in uncached prompt tokens per query (server count; $p < 0.001$ in every row). Hatched: total tokens, since the hosted APIs report no cache counts.

5 Results

5.1 Results on BrowseComp-Plus

We ran three held-out sets in sequence. The first (100 queries) compared PACT with VISTA once, after which the configuration was frozen (Appendix J). For the second (200 queries), run with all methods on one server (Table 1), three tests were fixed in advance. They asked for non-inferiority to VISTA, superiority over each of the VISTA authors' ReAct baselines, and a lower cost, and all three were met. PACT is non-inferior to VISTA (0.51 vs. 0.48) while the server recomputes 46% fewer prompt tokens, and it is more accurate than the truncate, clear and mask baselines (each $p < 0.001$ after Holm's correction). The third set (400 queries) tested whether PACT is also more accurate than VISTA under the lenient rule. It is not (0.510 vs. 0.498), but the secondary tests reproduced non-inferiority under the strict rule (0.455 vs. 0.450) and the saving (169K vs. 374K uncached tokens per query). Pooled over the three sets, PACT is non-inferior to VISTA (0.474 vs. 0.463) with 51% fewer uncached tokens (Figure 2).

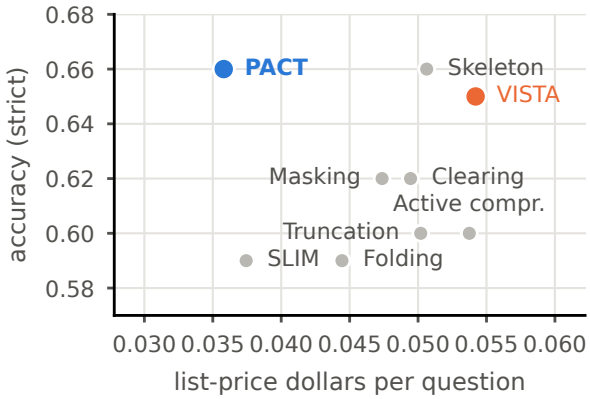

Figure 3. Accuracy against cost per question, Claude Haiku 5.5 (Appendix K).

Other backbones. Pre-registered tests on never-run queries find PACT non-inferior to VISTA on Nemotron-3-Ultra (100 queries; 0.45 vs. 0.44) with 31% fewer tokens, on Claude Haiku 5.5 (100 queries; 0.66 vs. 0.65) with 36% fewer, and on gpt-oss-20b (400 queries; 0.29 vs. 0.33), where VISTA needs 7.06× PACT's uncached tokens per correct answer. On gpt-oss VISTA is four points ahead (two-sided $p = 0.048$), and a first test without VISTA's empty-turn retry missed the margin (Appendix J). On Claude Haiku 5.5 PACT is also non-inferior to seven published context managers, at the lowest cost (Figure 3; Appendix K). An extension with a reasoning ledger, a summary at each compaction and a continuation note (Appendix L) is non-inferior to ACM's mechanism (Li et al., 2026b) in one harness on three of four backbones, at 0.55× its tokens on Claude Haiku; official ACM in its own runner is more accurate on Claude Haiku, but run inside that runner PACT is non-inferior to it on Claude Haiku at 0.80× its tokens and more accurate on MiniMax (0.660 vs. 0.540). In our harness PACT is non-inferior to official ACM on MiniMax at 0.67× its tokens (0.610 vs. 0.540) and more accurate on Nemotron-3-Ultra (0.660 vs. 0.570) at 2.1× its tokens.

GAIA. On the 301 hidden test tasks of GAIA (Mialon et al., 2024), VISTA's third benchmark, a pre-registered test finds PACT non-inferior to VISTA (0.346 vs. 0.355, lower bound −0.074) with 52% fewer uncached tokens per task ($p < 0.001$; Appendix M). With Claude Haiku 5.5, PACT scores 0.372 vs. 0.346 with 25% fewer tokens.

5.2 Effect of the Token Budget

Larger token budgets. Because the agent never sees the budget, a run at a larger budget nearly contains the run at every smaller one (Figure 4). Two Holm-corrected tests were fixed before any run at 327,680, the largest budget of the VISTA authors' sweep, and both were met. At this budget PACT is more accurate than VISTA (0.56 vs. 0.46) while the server recomputes 283K prompt tokens per query against 812K. It is also more accurate than VISTA run at the default budget, while cheaper by both counts. A replication on the third held-out set (398 queries), pre-registered before any run, was not met: 0.51 vs. 0.49 (40:33, $p = 0.24$). The saving replicated: 660K against 1,389K recomputed prompt tokens per query, and VISTA spends 2.2× PACT's tokens per correct answer. Pooled over both sets (596 queries), PACT is more accurate (0.53 vs. 0.48, 71:44, $p = 0.007$). Additional budget raises PACT's accuracy at a low marginal cost; VISTA's accuracy plateaued on the second set and rose less on the third, where the difference in gains is not significant.

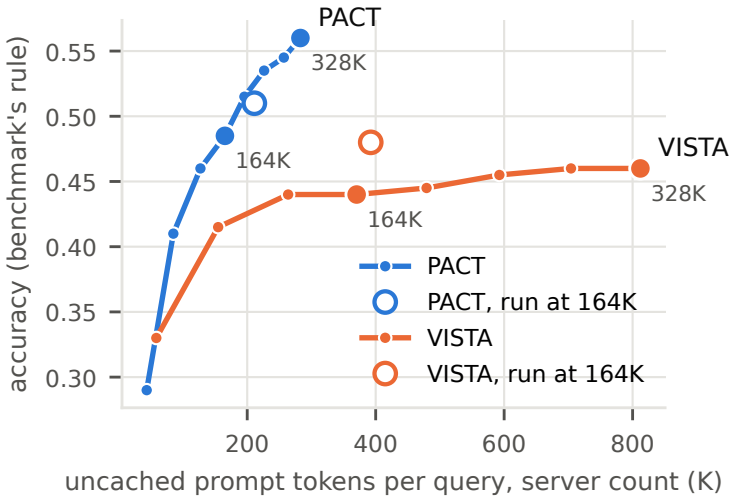

Figure 4. PACT converts additional budget into accuracy at a low cost. Accuracy (strict rule) against uncached prompt tokens per query as the token budget grows to 327,680, read off the runs at that budget (200 held-out queries). Hollow markers are the runs made at 163,840, which agree with the read-off on 82–83% of queries (Appendix N).

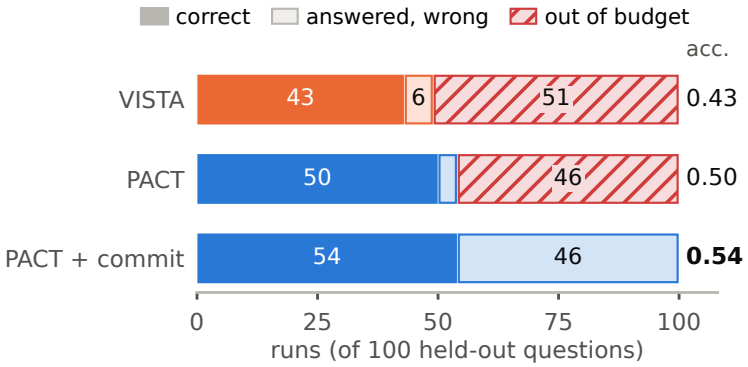

Figure 5. Commit-with-recall turns budget exhaustion into answers. How the 100 runs of the first held-out set end, rerun side by side, with accuracy (strict rule) on the right.

Answering within the budget. On the first held-out set, about half of VISTA's and PACT's runs ended without an answer because the budget ran out (Figure 5). Pre-registered tests on that set, rerun with the three methods side by side, found commit-with-recall more accurate than VISTA (0.54 vs. 0.43) at half its uncached tokens, with no run out of budget. Its gain over PACT alone is not significant. The set had been used once before, and VISTA scored 0.48 there against 0.43 in the rerun, so part of the gap is VISTA's run-to-run variation (Appendix J).

5.3 Results on LOCA-bench

All four methods run the same 75 tasks at three scales (Figure 6). PACT is not detectably less accurate than VISTA at any scale, with 0.653 vs. 0.640 at 16K, 0.473 vs. 0.418 at 32K (pooled over two runs) and 0.267 vs. 0.284 at 64K (MDE 0.18). With 32K pooled, neither managed policy is detectably behind the best method at any scale, while both unmanaged agents are at 64K (Table 7). There PACT doubles the accuracy of our ReAct (0.278 vs. 0.139 on the 72 tasks both completed). Where

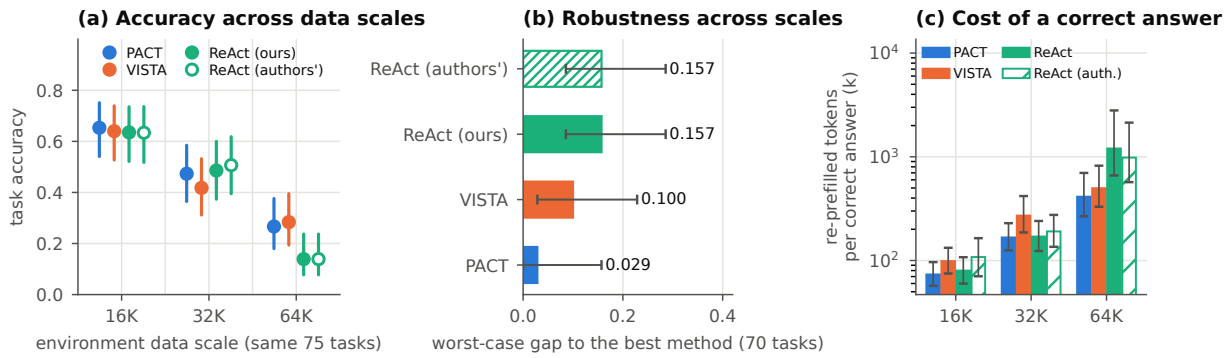

Figure 6. LOCA-bench across data scales (same 75 tasks, $B = 60{,}000$). (a) Accuracy with 95% Wilson intervals, 32K pooled over two runs. (b) Each method's largest accuracy gap to the best method over the three scales, with 95% bootstrap intervals over the 70 tasks every method ran. (c) Re-prefilled tokens per correct answer (log axis, 95% bootstrap intervals; 32K first run; Table 3).

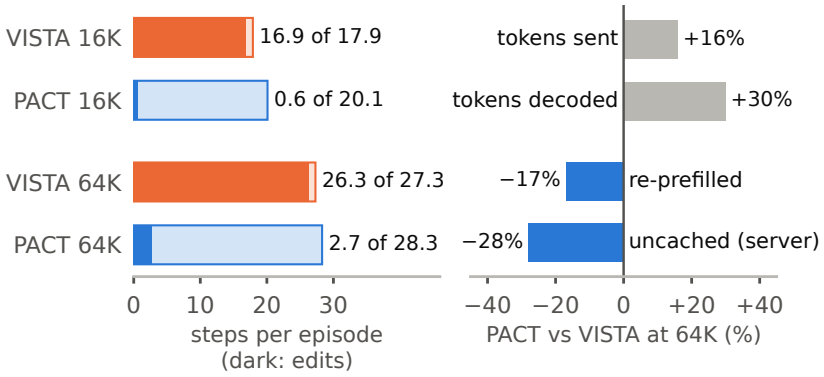

Figure 7. PACT sends more tokens but edits its prompt rarely. Left, steps per LOCA-bench episode, dark where the step edits the prompt. Right, PACT's difference from VISTA at 64K on two cache-blind counts (grey) and two counts of what the server computes (blue); at 16K they are +20, +24, −23 and −36%.

PACT and VISTA shared the server, it recomputes 36% fewer prompt tokens for PACT at 16K and 28% fewer at 64K (Table 2). Per correct answer, PACT has the lowest re-prefill at every scale, and it is significantly cheaper than VISTA at 16K (1.34×) and than both unmanaged agents at 64K.

5.4 Source of the Cost Reduction

By a cache-blind count PACT is the more expensive policy. At 64K it sends 16% more tokens than VISTA in total ($p = 0.005$) and decodes 30% more, yet the server computes 28% fewer of its prompt tokens (Figure 7). The instrument explains this *counting reversal*. VISTA re-sends its dashboard as the last message of every prompt, so each new prompt departs from the cached one where the old dashboard stood, and 26.3 of its 27.3 steps are edits. PACT edits only at compactions, on 2.7 of 28.3 steps ($p < 0.001$), as Proposition 1 predicts, and discards 34% less cache in all. On the hybrid backbone each edit also restarts the server from an earlier saved state, and at PACT's cuts on BrowseComp-Plus the server recomputed about 2.0× what the instrument charges (Appendix N), consistent with the larger server-side saving. Against the unmanaged agent the number of edits is the same, but each large edit of PACT's discards 27,538 cached tokens against 49,255 for truncation, which cuts at the front. The extra decoding would erase the saving only if a decoded token cost

more than 24 prefilled ones. Appendix O traces example tasks.

5.5 When Context Management Helps

In a pilot run, the unmanaged agent's median peak prompt reaches 0.66, 0.98 and 1.07 of the limit at 16K, 32K and 64K (Table 8). VISTA's paired accuracy gap to that agent is $+0.000$, -0.063 (two runs pooled) and $+0.155$, and PACT's is $+0.027$, $+0.007$ and $+0.139$. Neither policy gains below the limit, and both gain beyond it. The change in VISTA's gap from 32K to 64K is significant within task ($+0.221$, $p = 0.011$).

5.6 Variance and Ablations

Run-to-run variation. With identical inputs up to the first reply, 24 of 67 tasks (36%) changed outcome between two agents, so 75 tasks detect only differences of about 17 points. Every LOCA-bench number follows tested fixes for four harness defects and two analysis errors (Appendix P).

6 Conclusion

PACT manages context in the harness, archives evidence verbatim instead of summarising it, and edits the prompt rarely. On held-out BrowseComp-Plus queries and GAIA's hidden test set it is non-inferior to VISTA at about half the serving cost, and more accurate at a larger budget or with commit-with-recall. On LOCA-bench it is not detectably less accurate while the server recomputes 28–36% fewer prompt tokens. Policies should be reported with their operating point, server cost and repeated runs.

Limitations

Our evaluation covers three benchmarks and four backbones, so the results may not carry over to other agents, tools or serving stacks. Several comparisons rest on one held-out set or a modest number of tasks and can detect only large differences in accuracy. PACT also takes more steps and decodes more tokens than the baselines, and its cost model idealises the prefix cache that real servers implement. The benchmarks use somewhat different configurations of the method, and only the extension's components are attributed (Appendix L). Official ACM in its own runner is more accurate than PACT in ours on Claude Haiku 5.5; inside ACM's runner PACT matched it on Haiku and was more accurate on MiniMax-M3.1-Flash, and in ours it was non-inferior on MiniMax, each on one question set that informed our diagnoses.

Ethical Considerations

LOCA-bench's tasks use synthetic data, and BrowseComp-Plus uses a fixed, public web corpus; we use both, and the VISTA code, as released for research evaluation. We found that a code tool could read LOCA-bench's answer keys; our runs confine it, and we disclose the residual path we left open (Appendix P).

Acknowledgements

We used AI assistants (large language models) to write and debug experiment and analysis code and to draft and edit the text. The authors reviewed the code, results and text and take full responsibility for them.

References

Anastasios Nikolas Angelopoulos, Stephen Bates, Adam Fisch, Lihua Lei, and Tal Schuster. 2024. Conformal risk control. In *The Twelfth International Conference on Learning Representations*.

Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Sahel Sharifymoghaddam, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, Yanxi Li, Haoran Hong, Xinyu Shi, Xuye Liu, Hosna Oyarhoseini, Nandan Thakur, Crystina Zhang, Luyu Gao, and 2 others. 2026. BrowseComp-Plus: A fair and disentangled evaluation benchmark for deep search agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics* (*Volume 1: Long Papers*), pages 22349–22370. Association for Computational Linguistics.

Musa Cim, Burak Topcu, Chita Das, and Mahmut Taylan Kandemir. 2026. Parallel context compaction for long-horizon LLM agent serving. *arXiv preprint arXiv:2605.23296*.

Zhengru Fang, Senkang Forest Hu, Zhonghao Chang, Yu Guo, Yihang Tao, Hongyao Liu, Mengzhe Ruan, Jun Huang, and Yuguang Fang. 2026. Inference-time budget control for LLM search agents. *arXiv preprint arXiv:2605.05701*.

Xubin Hao, Hongjin Meng, Xin Yin, Jiawei Zhu, and Chenpeng Cao. 2026. Self-GC: Self-governing context for long-horizon LLM agents. *arXiv preprint arXiv:2607.00692*.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. 2026. ACON: Optimizing context compression for long-horizon LLM agents. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, pages 55321–55375. PMLR.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, pages 611–626. ACM.

Tianjian Li, Jingyu Zhang, William Jurayj, Xi Wang, Chuanyang Jin, Mehrdad Farajtabar, Eric Nalisnick, and Daniel Khashabi. 2026a. Self-compacting language model agents. *arXiv preprint arXiv:2606.23525*.

Xiaochuan Li, Ryan Ming, Meng Chu, Shuai Shao, Rong Jin, and Chenyan Xiong. 2026b. ACM: Agentic context management for long horizon tasks. *arXiv preprint arXiv:2607.23809*.

Yin Lin, Elaine Ang, Erkang Zhu, Bolin Ding, and Jingren Zhou. 2026. Context as an environment: Programmatic context management for long-horizon agents. *arXiv preprint arXiv:2608.21690*.

Tobias Lindenbauer, Igor Slinko, Ludwig Felder, Egor Bogomolov, and Yaroslav Zharov. 2025. The complexity trap: Simple observation masking is as efficient as LLM summarization for agent context management. *arXiv preprint arXiv:2508.21433*.

Tengxiao Liu, Zifeng Wang, Jin Miao, I-Hung Hsu, Jun Yan, Jiefeng Chen, Rujun Han, Fangyuan Xu, Yanfei Chen, Ke Jiang, Samira Daruki, Yi Liang, William Yang Wang, Tomas Pfister, and Chen-Yu Lee. 2026a. Budget-aware tool use enables effective agent scaling. In *Conference on Language Modeling* (*COLM*).

Yujian Liu, Jiabao Ji, Li An, Rohit Jain, Gungor Polatkan, Siyu Zhu, and Shiyu Chang. 2026b. Practical online KV cache compaction for LLM agents: An empirical study. *arXiv preprint arXiv:2608.00902*.

Abhilasha Lodha, Mahsa Pahlavikhah Varnosfaderani, Abir Chakraborty, and Abhinav Mithal. 2026. Less context, better agents: Efficient context engineering for long-horizon tool-using LLM agents. *arXiv preprint arXiv:2606.10209*.

Elias Lumer, Faheem Nizar, Akshaya Jangiti, Kevin Frank, Anmol Gulati, Mandar Phadate, and Vamse Kumar Subbiah. 2026. Don't break the cache: An evaluation of prompt caching for long-horizon agentic tasks. *arXiv preprint arXiv:2601.06007*.

Ziyu Ma, Hailang Huang, Shun Zou, Yong Wang, Shidong Yang, Yiming Hu, Fei Wei, and XiangXiang Chu. 2026. LongHorizon-Harness: Advancing long-horizon agents for real-world tasks. *arXiv preprint arXiv:2608.01964*.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2024. GAIA: a benchmark for general AI assistants. In *The Twelfth International Conference on Learning Representations*.

Trang Nguyen, Eulrang Cho, Bingqing Chen, and Tim Dettmers. 2026. CliffCompaction: Cost-efficient compaction for long-horizon coding agents. *arXiv preprint arXiv:2609.26779*.

Rui Pan, Zhuang Wang, Zhen Jia, Can Karakus, Luca Zancato, Tri Dao, Yida Wang, and Ravi Netravali. 2025. Marconi: Prefix caching for the era of hybrid LLMs. In *Proceedings of Machine Learning and Systems*, volume 7. MLSys.

Zhuoshi Pan, Qizhi Pei, Junru Lu, Honglin Lin, H. Vicky Zhao, Di Yin, and Xing Sun. 2026. ContextPilot: Teaching agents for proactive context management via fine-grained RL. *arXiv preprint arXiv:2608.28476*. Accepted to EMNLP 2026 (Main Track).

Furkan Sakizli. 2026. TSCG: Deterministic tool-schema compilation for agentic LLM deployments. *arXiv preprint arXiv:2605.04107*.

Andrew Semenov and Svyatoslav Dorofeev. 2026. Beyond compaction: Structured context eviction for long-horizon agents. *arXiv preprint arXiv:2606.11213*.

Rulin Shao, Shannon Zejiang Shen, Junjie Oscar Yin, Yuetai Li, Minheng Wang, Hamish Ivison, Radha Poovendran, Nathan Lambert, Teng Xiao, Mike Lewis, Wen-tau Yih, Luke Zettlemoyer, and Pang Wei Koh. 2026. Context language models. *arXiv preprint arXiv:2609.37725*.

Yupeng Su, Jiayi Tian, Zheng Zhang, and Souvik Kundu. 2026. ReFold: Training-free reversible inter-turn context folding for long-horizon agents. *arXiv preprint arXiv:2610.07863*.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. 2026. Scaling long-horizon agent via context folding. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, pages 116894–116910. PMLR.

Nikhil Verma. 2026. Active context compression: Autonomous memory management in LLM agents. *arXiv preprint arXiv:2601.07190*.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, and 5 others. 2025. OpenHands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. 2024. MINT: Evaluating LLMs in multi-turn interaction with tools and language feedback. In *The Twelfth International Conference on Learning Representations*.

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Dingchu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, Minhao Cheng, Shuai Wang, Hong Cheng, and Jingren Zhou. 2026a. ReSum: Unlocking long-horizon search intelligence via context summarization. *arXiv preprint arXiv:2509.13313*.

Yong Wu, Yanzhao Zheng, Tianze Xu, Zhentao Zhang, Yuanqiang Yu, Jihuai Zhu, Chao Ma, Binbin Lin, Baohua Dong, Hangcheng Zhu, Ruohui Huang, and Gang Yu. 2026b. ContextBudget: Budget-aware context management for long-horizon search agents. In *Conference on Language Modeling (COLM)*.

Binyan Xu, Haitao Li, and Kehuan Zhang. 2026a. LLM agents are latent context managers: Eliciting self-managed context via state proprioception. *arXiv preprint arXiv:2606.30005*.

Buqiang Xu, Zirui Xue, Dianmou Chen, Chenyang Fu, Chiyu Wu, Caiying Huang, Chen Jiang, Jizhan Fang, Xinle Deng, Yijun Chen, Yunzhi Yao, Xuehai Wang, Jin Shang, Gong Yu, and Ningyu Zhang. 2026b. TokenPilot: Cache-efficient context management for LLM agents. *arXiv preprint arXiv:2606.17016*. Accepted to Findings of EMNLP 2026.

Peng Xu, Zuyu Zhang, Yuze Sun, Feng Tian, Long Wang, and Chen Zhang. 2026c. ContextPipe: Database-inspired context assembly for long-horizon agents. *arXiv preprint arXiv:2609.00749*.

Junjie Yao, Zhangchen Zhou, and Zhi-Qin John Xu. 2026. CADOC: Cache-aware dynamic object context for long-horizon agents. *arXiv preprint arXiv:2609.37012*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Howard Yen, Yoonsang Lee, Ashwin Paranjape, Mengzhou Xia, Thejas Venkatesh, Jack Hessel, Danqi Chen, and Yuhao Zhang. 2026. Lost in the maze: Overcoming context limitations in long-horizon agentic search. *arXiv preprint arXiv:2510.18939*.

Lu Yi, Runlin Lei, Liuyi Yao, Yuexiang Xie, Yuyang Li, Wenhao Zhang, Zhewei Wei, Yaliang Li, and Jian-Yun Nie. 2026. Learning agent-compatible context management for long-horizon tasks. *arXiv preprint arXiv:2605.30785*.

Weihao Zeng, Yuzhen Huang, and Junxian He. 2026. LOCA-bench: Benchmarking language agents under controllable and extreme context growth. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, pages 153347–153363. PMLR.

Haoxiang Zhang, Qixin Xu, Zhuofeng Li, Lei Zhang, Pengcheng Jiang, Yu Zhang, and Julian McAuley. 2026a. Masking stale observations helps search agents – until it doesn't: A regime map and its mechanism. *arXiv preprint arXiv:2606.00408*.

Yuxiang Zhang, Jiangming Shu, Ye Ma, Xueyuan Lin, Shangxi Wu, and Jitao Sang. 2026b. Memory as action: Autonomous context curation for long-horizon agentic tasks. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 19149–19164. Association for Computational Linguistics.

Chenyu Zhou, Huacan Chai, Wenteng Chen, Zihan Guo, Rong Shan, Yuanyi Song, Tianyi Xu, Yingxuan Yang, Aofan Yu, Weiming Zhang, Congming Zheng, Jiachen Zhu, Zeyu Zheng, Zhuosheng Zhang, Xingyu Lou, Changwang Zhang, Zhihui Fu, Jun Wang, Weiwen Liu, and 2 others. 2026. Externalization in LLM agents: A unified review of memory, skills, protocols and harness engineering. *arXiv preprint arXiv:2604.08224*.

A Baselines, Configurations, and Compute

Baseline implementations. Both unmanaged LOCA-bench agents are the benchmark's own ReAct runner: ours from the benchmark's repository (`hkust-nlp/LOCA-bench`, commit `8b6fac4`), the authors' from the copy vendored in the VISTA repository (`binyxu/VISTA`, commit `647a93d`), run through VISTA's own command line with its own strategies. Our changes to the VISTA codebase are 60 lines, and none touches its policies: the sandbox that confines the code tools, the cost instrument, a request timeout that no longer truncates long prompts, and the step limit its runner lacks (both limits admit the same 35 model calls). Our own codebase adds 414 lines over five files: PACT, the same sandbox, instrument and timeout, the fixes of Appendix P, and three pieces copied verbatim from the VISTA runner so that the two codebases cannot differ where they should agree, namely two tool-schema sanitisers and its token estimator. The two unmanaged agents differ in one setting. Ours keeps the runner's own truncation of the oldest turns once a prompt would exceed the limit less the reply's 2,048 tokens, while the authors' launcher leaves context management off, so an overflowing transcript ends the episode, as happens in 29% of its episodes at 64K. Their accuracies are not detectably different at any scale (0.635 vs. 0.634, 0.542 vs. 0.507, 0.139 vs. 0.139; $p = 0.839$, 0.804, 1.000).

Configurations. On LOCA-bench PACT uses $\theta = 55{,}000$ under $B = 60{,}000$, the forecast ($k = 4$ continuations of at most 160 tokens at temperature 0.9, each predicting four actions), risk-controlled retention at $\alpha = 0.10$ (Angelopoulos et al., 2024), and neither folding nor the loop guard; the agent samples at temperature 1.0. On BrowseComp-Plus the window is 32,768 tokens, $\theta = 30{,}037$ (the same fraction of the window), $G = 163{,}840$ or 327,680, the forecast is off, folding and the loop guard are on, and decoding is greedy. Commit-with-recall fires at $0.9\,G$ and ranks passages of 1,200 characters that overlap by 200.

Step limit. The 35-step limit ends 31–55% of 64K episodes before an answer (PACT 55%, VISTA 46%, our ReAct 40%, the authors' 31%) and 11–29% at 16K and 32K.

Compute. Qwen3.8-27B was served by vLLM with prefix caching. Every LOCA-bench run used one server on four 24 GB RTX 3090 GPUs (tensor parallel, fp8 KV cache); the BrowseComp-Plus runs used one 80 GB H100 or two 48 GB A6000 GPUs (tensor parallel). Methods compared on the server's count shared one server at the same time. gpt-oss-20b was served on two 24 GB RTX 3090 GPUs with a bf16 KV cache, at the model's default reasoning effort, with its reasoning tokens counted against the token budget in every method. Nemotron-3-Ultra was called through a hosted API at its maker's sampling settings (temperature 1.0, top-p 0.95). Claude Haiku 5.5 was called through Anthropic's API, which fixes the model's sampling and accepts no temperature, so no request of either method set one. The BrowseComp-Plus judge, Qwen3-32B, ran on the same kinds of GPU; within each set every method shares one judge.

B Proof of Proposition 1

Setting. An episode has steps $s = 1, \dots, T$. At step s the harness sends the prompt P_s, the model returns the completion a_s, and the tool returns the observation o_s. Prompts are token sequences, $|x|$ is the length of x, $x \circ y$ is concatenation, and $x \preceq y$ means that x is a prefix of y. Let $\tilde{P}_{s+1}$ be the prompt that would be sent at step $s+1$ without context management, the transcript of P_s with a_s and o_s appended, and let $Q_s = P_s \circ a_s$ be what the server holds in its cache after step s. Step $s \geq 2$ is a *compaction step* if $P_s \neq \tilde{P}_s$. A constant offset, such as the system prompt and the task, is absorbed into θ and κ.

Assumptions. (R) Rendering is append-only: appending messages to a transcript appends tokens to its prompt, and a completion is rendered as it was decoded. Hence $Q_s \preceq \tilde{P}_{s+1}$ and $|\tilde{P}_{s+1}| = |P_s| + |a_s| + |o_s|$, where $|o_s|$ includes the message delimiters. (T) Threshold: $P_{s+1} = \tilde{P}_{s+1}$ whenever $|\tilde{P}_{s+1}| \leq \theta$. (K) Size after a compaction: if s is a compaction step, then $|P_s| \leq \kappa < \theta$.

Cost. For $s \geq 2$ let p_s be the length of the longest common prefix of Q_{s-1} and P_s, so that an attention-only cache reuses p_s tokens and the server computes $U_s = |P_s| - p_s$. Of these, K_s are tokens that also occur in Q_{s-1} after position p_s (the instrument matches whole messages) and $N_s = U_s - K_s$ are new. Step s is an *append* if $p_s = |Q_{s-1}|$ and an *edit* otherwise.

Lemma 1. *Every step $s \geq 2$ that is not a compaction step is an append, and $K_s = 0$.*

Proof. Such a step has $P_s = \tilde{P}_s$, and by (R) $Q_{s-1} \preceq \tilde{P}_s = P_s$, so $p_s = |Q_{s-1}|$. No token of Q_{s-1} lies after position p_s, hence $K_s = 0$. □

Lemma 2. *Let $t_1 < \cdots < t_C$ be the compaction steps. For $k = 1, \dots, C-1$, $\sum_{s=t_k}^{t_{k+1}-1} (|a_s| + |o_s|) > \theta - \kappa$.*

Proof. By (K), $|P_{t_k}| \leq \kappa$. Every step s with $t_k < s < t_{k+1}$ is not a compaction step, so $P_s = \tilde{P}_s$, and by (R) $|P_s| = |P_{s-1}| + |a_{s-1}| + |o_{s-1}|$. Summing these identities, $|\tilde{P}_{t_{k+1}}| = |P_{t_k}| + \sum_{s=t_k}^{t_{k+1}-1} (|a_s| + |o_s|)$. Since t_{k+1} is a compaction step, (T) gives $|\tilde{P}_{t_{k+1}}| > \theta$. Hence the sum exceeds $\theta - |P_{t_k}| \geq \theta - \kappa$. □

Corollary 1. *Under the conditions of Proposition 1, $\sum_s U_s \leq \sum_s N_s + \kappa\,(1 + \Lambda/(\theta - \kappa))$, independent of T.*

Proof of Proposition 1. By Lemma 1 every edit is a compaction step, so an episode has at most C edits. The index sets $\{t_k, \dots, t_{k+1} - 1\}$, $k = 1, \dots, C-1$, are disjoint subsets of $\{1, \dots, T\}$, so Lemma 2 gives $\Lambda \geq \sum_{k=1}^{C-1} \sum_{s=t_k}^{t_{k+1}-1} (|a_s| + |o_s|) > (C-1)(\theta - \kappa)$ whenever $C \geq 2$, that is $C < 1 + \Lambda/(\theta - \kappa)$; the bound holds trivially when $C \leq 1$. For the second inequality, $K_s = 0$ at every step that is not a compaction step (Lemma 1), and at a compaction step $K_s \leq U_s = |P_s| - p_s \leq |P_s| \leq \kappa$ by (K). Summing over the C compaction steps gives $\sum_s K_s \leq \kappa C$. Corollary 1 follows from $U_s = N_s + K_s$. □

Remark. No bound of this form holds for a policy that edits on every step. If every step $s \geq 2$ is an edit after whose first change at least $d \geq 1$ cached tokens are kept, then $\sum_s K_s \geq (T-1)\,d$, which grows linearly in T. A status message re-sent as the last message of every prompt is such a policy: in Q_{s-1} the completion a_{s-1} follows the old message, so the first change is at the old message and a_{s-1} is prefilled again, giving $d \geq \min_s |a_{s-1}|$. On a hybrid model, whose recurrent layers resume only from saved states, the server also recomputes from the last saved state before p_s, which adds to U_s at every step and most at edits (Appendix N).

Where PACT is ahead	Against	PACT vs. baseline	p
Accuracy beyond the budget	vs ReAct (ours) at 64K	0.278 vs 0.139	0.041
Accuracy beyond the budget	vs ReAct (authors') at 64K	0.278 vs 0.139	0.052
Re-prefill (instrument, incl. policy calls)	vs VISTA at 64K	125k vs 151k tokens (-17%)	0.011
Uncached prompt tokens (server, incl. policy calls)	vs VISTA at 64K	194k vs 270k tokens (-28%)	<0.001
Uncached prompt tokens (server, incl. policy calls)	vs VISTA at 16K	61k vs 95k tokens (-36%)	<0.001
Uncached prompt tokens (server, incl. policy calls)	vs ReAct (ours) at 64K	187k vs 233k tokens (-20%)	0.030
Episodes hitting the lossy trim	vs ReAct (ours) at 64K	7% vs 50%	<0.001
Episodes lost to the context	vs ReAct (authors') at 64K	0% vs 29%	<0.001

Table 2. LOCA-bench comparisons in which PACT is ahead, each with its paired test on the tasks both methods completed. Rows with $p > 0.05$ are directional only. Accuracy against VISTA at 64K is not detectably different (-0.014, 95% CI $[-0.14, 0.11]$), and at 64K PACT takes more steps and decodes more tokens than every baseline (Table 4).

C Paired Comparisons and Cost per Correct Answer

Table 2 lists every LOCA-bench comparison in which PACT is ahead, Table 3 gives re-prefilled tokens per correct answer, and Table 4 gives every paired comparison at 64K.

Method	16K	32K	64K
PACT	73k	167k	412k
VISTA	**99k (1.34×)**	**271k (1.63×)**	498k (1.21×)
ReAct (ours)	80k (1.09×)	170k (1.02×)	**1,202k (2.92×)**
ReAct (authors')	**108k (1.47×)**	191k (1.14×)	**983k (2.38×)**

Table 3. Re-prefilled tokens per correct answer on LOCA-bench, with each baseline's ratio to PACT; 32K is the first run, whose bold VISTA cell rests on an accuracy lead the second run did not reproduce. Bold: the 95% bootstrap interval over tasks lies above parity, on the 70, 72 and 71 tasks every method completed at 16K, 32K and 64K. The instrument's count is used so that methods that did not share the server at the same time stay comparable. VISTA's 16K ratio has the interval $[1.07, 1.69]$.

D Pre-registered 32K Replication

The second 32K run was planned before it existed, because the run-to-run variation of §5.6 predicts reversals of the size the first run showed (Table 5).

Against	Metric	n	Difference	95% CI	p	n to resolve
ReAct (ours)	accuracy, all episodes	72	0.139*	[0.019, 0.259]	0.041	attained
	submitted	72	0.097	[−0.040, 0.235]	0.230	284
	window blown	72	0.00	[0.00, 0.00]	1.000	—
	re-prefill tokens†	72	−51,460*	[−87,805, −15,116]	0.006	attained
	decoded tokens†	72	3,793*	[1,746, 5,839]	0.000	attained
	steps	72	4.5*	[2.1, 6.9]	0.000	attained
ReAct (authors')	accuracy, all episodes	72	0.139	[0.012, 0.265]	0.052	118
	submitted	72	−0.250*	[−0.396, −0.104]	0.002	attained
	window blown	72	−0.29*	[−0.40, −0.18]	0.000	attained
	re-prefill tokens†	72	−18,272	[−44,084, 7,540]	0.162	284
	decoded tokens†	72	3,903*	[2,090, 5,716]	0.000	attained
	steps	72	5.6*	[3.0, 8.1]	0.000	attained
VISTA	accuracy, all episodes	74	−0.014	[−0.138, 0.111]	1.000	12,356
	submitted	74	−0.054	[−0.207, 0.099]	0.597	1,170
	window blown	74	−0.04	[−0.09, 0.01]	0.250	188
	re-prefill tokens†	74	−26,114*	[−45,949, −6,278]	0.011	attained
	decoded tokens†	74	3,236*	[1,338, 5,133]	0.001	attained
	steps	74	1.0	[−1.3, 3.2]	0.395	794

Table 4. PACT against every baseline at 64K, paired. Difference is PACT minus the baseline: positive favours PACT for accuracy and submission, negative favours it for window blown, re-prefill, decoded tokens and steps. "n to resolve": tasks needed for the observed difference to reach $p < 0.05$. †The policy's own probe calls included.

Accuracy difference	run 1	run 2	pooled
PACT − VISTA	+0.135 (0.087)	−0.014 (1.000)	+0.062 (0.234)
ReAct − VISTA	+0.181 (0.024)	−0.055 (0.541)	+0.063 (0.308)
PACT − ReAct	−0.028 (0.832)	+0.041 (0.701)	+0.007 (1.000)

Table 5. The 32K comparisons on two independent runs of the same 75 tasks, the second pre-registered before it existed (McNemar p per run; exact sign-flip p over tasks for the pooled per-task means). Neither the first run's lead of PACT over VISTA nor its loss of VISTA against ReAct reproduced.

E Accuracy, Deficit to the Best Method, and the Crossover

Table 6 gives accuracy at every scale, Table 7 each method's deficit to the best one, and Table 8 the crossover indexed by the unmanaged agent.

Method	16K	32K	64K
ReAct (ours)	0.635 (74)	0.486 (71)	0.139 (72)
ReAct (authors')	0.634 (71)	0.507 (73)	0.139 (72)
VISTA	0.640 (75)	0.418 (73)	0.284 (74)*
PACT	0.653 (75)	0.473 (75)	0.267 (75)*

Table 6. Accuracy (tasks). At 32K each method run twice is pooled over both runs (per-task mean; the authors' ReAct ran once; Table 5). *: differs from our ReAct at that scale ($p < 0.05$; McNemar at 16K and 64K, exact sign flip at 32K).

Method	16K	32K	64K
PACT	best (0.653)	−0.028 (0.832)	−0.014 (1.000)
VISTA	−0.013 (1.000)	**−0.181 (0.024)**	best (0.284)
ReAct (ours)	−0.027 (0.824)	best (0.542)	**−0.155 (0.019)**
ReAct (authors')	−0.056 (0.523)	−0.028 (0.804)	**−0.153 (0.007)**

Table 7. Accuracy deficit to the best method at each scale on the first run, paired on shared tasks (McNemar p in parentheses; bold: $p < 0.05$). Both unmanaged agents are significantly behind at 64K and PACT is behind nowhere. VISTA's 32K deficit did not reproduce in a pre-registered second run (Table 5): pooled over both runs no method is significantly behind at 32K.

	peak/budget	trim	VISTA	PACT
16K	0.66	16%	+0.000	+0.027
32K	0.98	36%	−0.063	+0.007
64K	1.07	50%	**+0.155**	**+0.139**

Table 8. The crossover indexed by the unmanaged agent rather than by the benchmark: its median peak prompt as a fraction of the enforced budget, how often its lossy trim fired, and each managed policy's paired accuracy gap to it (32K pooled over both runs; Table 5). Bold: $p < 0.05$ (McNemar at 16K and 64K; at 32K the exact sign flip finds neither gap significant). Management helps once the prompt exceeds the limit and has no detectable effect below it. The change in VISTA's gap from 32K to 64K, +0.221 within task, is significant (exact sign flip, $p = 0.011$, 70 tasks).

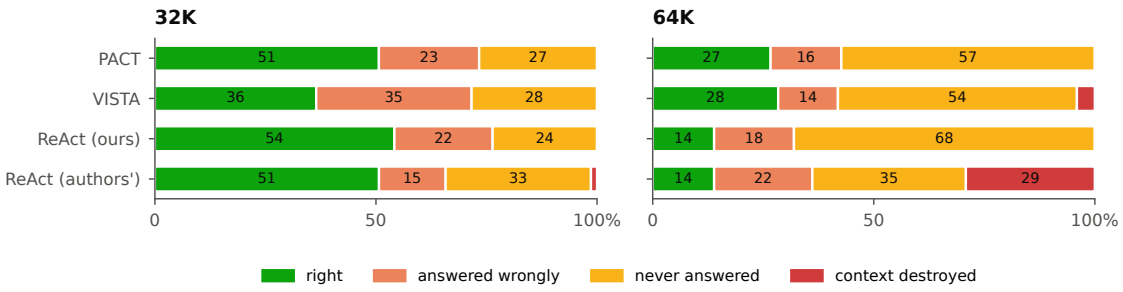

Figure 8. How episodes end (%; 32K: first run). Near the budget (32K) VISTA answers about as often as ReAct and is wrong more often; beyond it (64K) the authors' ReAct loses 29% of episodes to a destroyed context and ours mostly stops answering. PACT loses none to overflow; it answers less often than the authors' ReAct but, when it does, is right about as often as VISTA (59% vs. 61%).

F Outcome Decomposition

Figure 8 and Table 9 show how LOCA-bench episodes end.

Effect of VISTA's dashboard near the limit. On the first run VISTA's 32K deficit was precision, not completion (Figure 8); on the 27 tasks where VISTA archived nothing it held against our ReAct ($p = 0.013$) though not the authors' ($p = 0.11$), an observational split that points at what VISTA adds rather than what it removes. The authors' own dashboard-off variant is the manipulation: without the dashboard VISTA answers more precisely (0.73 against 0.51, $p = 0.057$) but less often (0.59 against 0.70, $p = 0.096$) and 2.4 steps later ($p = 0.013$), so its accuracy barely moves (+0.056, $p = 0.52$). The dashboard appears to trade precision for completion, though only the step difference is significant.

Method	n	submits (%)	right when it does (%)	overall (%)	repeats (%)
32K					
ReAct (ours)	72	75.0	70.4	54.2	1.6
ReAct (authors')	73	60.3	72.7	50.7	2.2
VISTA	74	70.3	50.0	36.5	2.6
PACT, plain stubs	75	60.0	71.1	46.7	2.8
PACT, plain stubs, no forecast	75	57.3	62.8	45.3	3.0
PACT (ours)	75	68.0	66.7	50.7	1.7
64K					
ReAct (ours)	72	29.2	38.1	13.9	1.9
ReAct (authors')	72	63.9	19.6	13.9	1.3
VISTA	74	44.6	60.6	28.4	2.6
PACT (ours)	75	38.7	58.6	26.7	3.3

Table 9. How episodes end at 32K (first run) and 64K: share that submit an answer, share of those that are right, overall accuracy, and share of tool calls that repeat an earlier call.

Variant	Metric	n	difference	95% CI	p	n to resolve
PACT, plain stubs, no forecast	accuracy, all episodes	75	0.013	[−0.120, 0.147]	1.000	14,908
	submitted	75	0.027	[−0.115, 0.168]	0.851	4,168
	re-prefill tokens†	75	16,096*	[4,676, 27,516]	0.006	attained

Table 10. The plain-stub variant with its forecast probes against the same variant without them, 32K, paired. Difference is with probes minus without. †The policy's own probe calls included.

G Forecast Ablation

Component ablations. At 64K the forecast was zero for every block in 60% of compactions, and in 99% the limit forced evictions beyond what it allowed. Removing the forecast, the cut pricing or the selection did not measurably change accuracy (0.213, 0.280 and 0.230 against 0.267 for full PACT; $p \geq 0.48$, MDE about 0.16) or re-prefill ($p \geq 0.32$).

At 32K the forecast was ablated on the plain-stub variant, an earlier configuration of PACT whose stubs carry the call, the archive file and a 160-character preview, without the result's shape (Table 10).

H Accuracy and Cost at Each Scale

Figure 9 places every LOCA-bench method on accuracy and re-prefill at once.

I Crossover by Task Family

Figure 10 checks that the crossover is not one task family's effect.

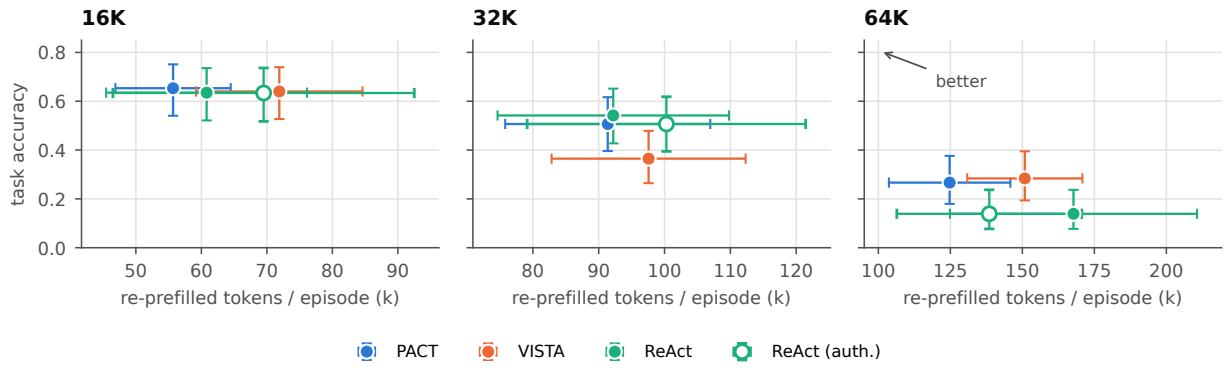

Figure 9. Accuracy against re-prefilled tokens per episode at every scale (95% intervals on both axes; up and left is better; 32K: the first of two runs, Table 5). PACT has the lowest re-prefill at every scale, significantly below VISTA at 16K and 64K by the paired test ($p = 0.002$ and $p = 0.011$; the intervals drawn here are marginal, so they overlap where the paired difference does not), and is never significantly less accurate than the best method. The instrument's count is shown because it does not depend on server load; the server's own count agrees where the methods shared the server.

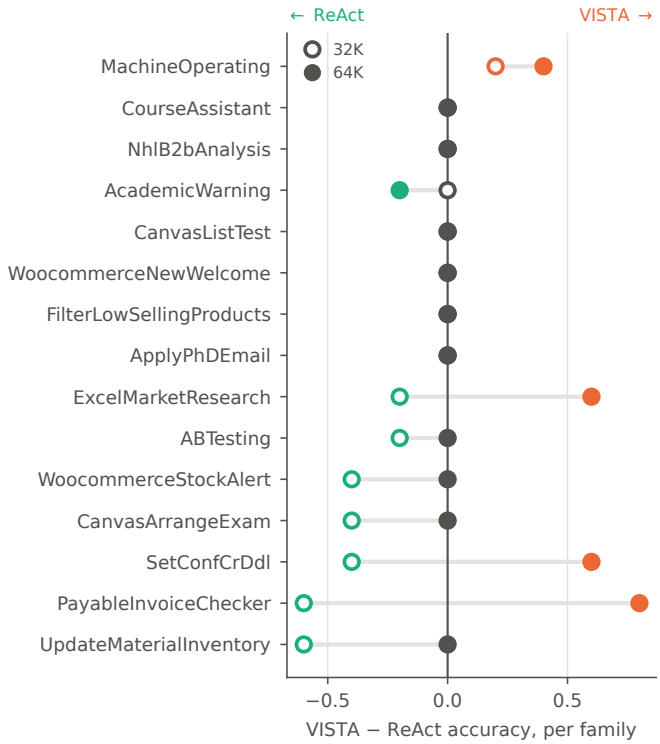

Figure 10. The crossover family by family: mean VISTA − ReAct accuracy over each family's five seeds, hollow at 32K (first run) and filled at 64K. Six families are level at both scales; seven sit left of zero near the budget and move to or past zero beyond it; one favours VISTA at both; one moves against. The gap rises in 8, falls in 1 and is unchanged in 6. The headline is not one family's effect.

J Complete BrowseComp-Plus Results

Method	Acc.	Uncached	Unc./correct	Cycles
PACT	0.48	206K	0.43M	6/45
VISTA (authors')	0.48	374K	0.78M (1.82×)	24/48
ReAct, truncate (authors')	0.38	262K	0.69M (1.60×)	33/57
ReAct, clear (authors')	0.38	432K	1.14M (2.65×)	30/58
ReAct, mask (authors')	0.42	403K	0.96M (2.23×)	20/54

Table 11. First held-out BrowseComp-Plus set (100 queries), columns as in Table 1. The ratio's interval excludes parity for VISTA and the three ReAct baselines.

On the first held-out set PACT matches VISTA (0.48 each, 9:9) while the server re-computes 45% fewer prompt tokens ($p < 0.001$); per correct answer VISTA costs 1.82× as many uncached tokens ($[1.32, 2.47]$). It is more accurate than the authors' truncate and clear baselines (0.38 each, 13:3, $p = 0.021$, secondary tests that would not survive a correction over the three) and leads mask only directionally (0.42, $p = 0.146$). Pooled with the second set (300 queries), PACT is non-inferior to VISTA (0.50 against 0.48, lower bound -0.021), more accurate than truncate (41:6), clear (42:10) and mask (39:10) under Holm, and cheaper per correct answer (VISTA 1.92×, $[1.62, 2.29]$). The second set was judged in a separate session with the same judge model; within each set every method shares one judge.

Second held-out set. Against VISTA, discordant pairs 22:16 and non-inferiority lower bound -0.021; uncached prompt tokens $p < 0.001$. Against the truncate, clear and mask baselines, discordant pairs 28:3, 29:7 and 30:7, each $p < 0.001$ after Holm's correction.

Third held-out set. Its one primary test, fixed before the run, asked whether PACT is more accurate than VISTA under the lenient rule, which grades the answer forced when the budget runs out; under that rule PACT had led on the second set in a secondary analysis. The test was not met (0.510 vs. 0.498, 37:32, $p = 0.315$). The secondary tests under the strict rule were met (non-inferiority, lower bound -0.032; uncached tokens 169K vs. 374K, $p < 0.001$, and VISTA needs 2.23× PACT's per correct answer, $[1.92, 2.60]$). Pooled over the three sets (700 queries), a secondary analysis fixed with the third set's plan, PACT is non-inferior (lower bound -0.016), and VISTA needs 2.08× its uncached tokens per correct answer ($[1.86, 2.34]$).

Prompt length. At the same 32K window, VISTA's prompt, whose dashboard and protocol sit on top of its managed blocks, exceeded the window in 138 of 200 runs on the second set and 252 of 400 on the third (up to 65.6K tokens), PACT's in none.

Commit-with-recall confirmation. The confirmation reran VISTA, PACT and PACT with commit-with-recall side by side on one server, a different one from the set's first run; the set had served once, for PACT against VISTA, and nothing was tuned on it. The mechanism's design drew on how runs exhausted the budget on the second and third sets, and its development gate used 130 design queries. On the rerun's server VISTA scored 0.43 and PACT 0.50, against 0.48 each in the first run (Table 11); every comparison here is paired within the rerun. The primary test (more accurate than VISTA, one-sided exact McNemar) and two Holm-corrected secondary tests were fixed before any run. Runs that exhausted the budget: 0 with the commit, 51 for VISTA, 46 for PACT. Accuracy 0.54 against VISTA's 0.43 (15:4, $p = 0.010$) and PACT's 0.50 (8:4, $p = 0.194$, not significant); uncached

prompt tokens a query 249K against VISTA's 480K ($p < 0.001$); per correct answer VISTA needs 2.42× its uncached tokens ($[1.87, 3.21]$, 95% bootstrap). Under the authors' rule, which grades forced answers, it leads VISTA 0.55 to 0.47 (12:4, $p = 0.038$).

gpt-oss-20b. A first pre-registered comparison ran on the second held-out set, with PACT in the ReAct loop, which ends a run on a turn with neither text nor a tool call. Its accuracy test was not met (0.300 vs. 0.390, discordant pairs 4:22, lower bound -0.131), and its cost test was (VISTA needs 7.15× PACT's uncached tokens per correct answer); 27 of PACT's 200 runs ended on such an empty turn, which VISTA's loop retries. The comparison in §5.1 was then planned on the third held-out set, never run on this model, with PACT using VISTA's own retry and the agent settings of the Qwen runs. It met the margin (lower bound -0.071). Its secondary test that the retry raises PACT's accuracy was not met (23:19, $p = 0.322$).

Nemotron-3-Ultra. Nemotron-3-Ultra (550B parameters, 55B active) ran on NVIDIA's API at its maker's sampling settings (temperature 1.0, top-p 0.95) for both methods, PACT with VISTA's empty-turn retry, on the first 100 questions of the second held-out set: the set was reduced from 200 for the API's rate limits before any run was judged. Discordant pairs 11:10; non-inferiority $p = 0.008$, lower bound -0.065; tokens per question (prompt plus completion, every call; the API reports no cache counts) 664K against 966K ($p < 0.001$). The runs had continued past the first 100 questions before the amendment took effect, and VISTA's last 21 were completed afterwards. On all 200 questions, an exploratory analysis outside the pre-registered set, PACT remains non-inferior (0.470 against 0.485, discordant pairs 19:22, lower bound -0.068) with 26% fewer tokens. Those 21 answers were judged by the same judge model on a hosted fp8 endpoint, which agreed with our judge on 99 of 100 re-judged answers. A second large model, GLM-5.3, was pre-registered alongside and abandoned because its API was unavailable before any of its runs was judged; the Holm correction still counts it, and both tests clear the stricter threshold (0.025).

Claude Haiku 5.5. A separately pre-registered test ran Claude Haiku 5.5 on Anthropic's API on the same first 100 questions, with the model's own sampling for both methods; it accepts no temperature, so PACT's loop guard samples its one step at that setting. Discordant pairs 7:6; non-inferiority $p = 0.001$, lower bound -0.049; tokens per question 324K against 504K ($p < 0.001$), or \$0.036 against \$0.054 a question at the API's list price. The answers were judged by the same judge model on the hosted fp8 endpoint described above.

K Published Context Managers on Claude Haiku 5.5

Setup. Two separately pre-registered tests compare PACT with the context managers in VISTA's own comparison (Xu et al., 2026a), all run from the VISTA authors' code. The first covers three published managers: periodic summarization of the trajectory as in SLIM (Yen et al., 2026), Active Context Compression (Verma, 2026), in which the agent writes a knowledge block that replaces older rounds, and the branch-and-return interface of Context-Folding (Sun et al., 2026), prompted without its reinforcement-learning stage. The second covers VISTA's four fixed external policies: stale-observation masking (Lindenbauer et al., 2025; Zhang et al., 2026a), tool-result clearing, skeleton compression and ReAct-style truncation. All methods ran Claude Haiku 5.5 on the same 100 held-out questions with the same window, budget, retriever and judge as the comparison with VISTA (Appendix J). For each baseline the test asks whether PACT is non-inferior at a margin of 0.10 and whether it uses fewer tokens, with the Holm correction within each test.

Results. PACT is non-inferior to all seven baselines (Table 12, Figure 11). It ties skeleton compression and answers 4–7 more questions in 100 than each of the others, but no accuracy difference is significant by McNemar's test ($p \geq 0.07$), so we claim non-inferiority only. It uses 20–35% fewer tokens than six of the baselines ($p < 0.001$ each) and about as many as SLIM, and at the API's list price it is the cheapest of the nine methods per question.

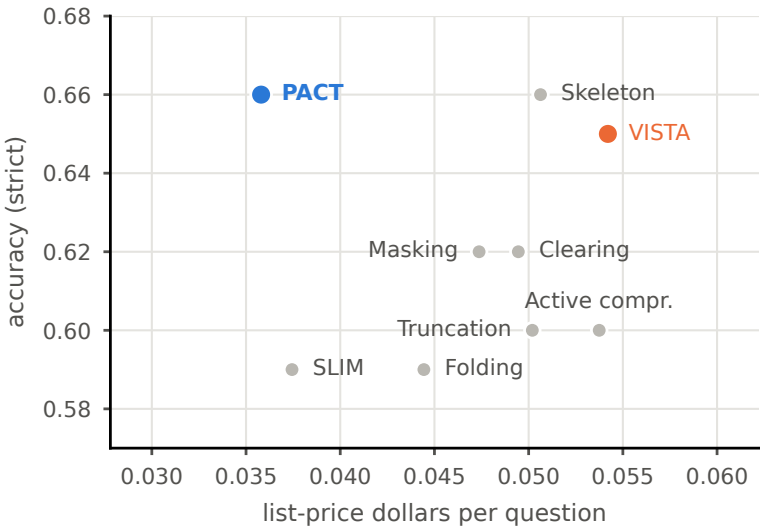

Figure 11. Accuracy against list-price dollars per question on Claude Haiku 5.5 (100 held-out BrowseComp-Plus questions, strict rule): PACT, VISTA and the seven published context managers of Table 12 (Active compr., Active Context Compression; Folding, Context-Folding; Masking, stale-observation masking; Clearing, tool-result clearing; Skeleton, skeleton compression; Truncation, ReAct-style truncation). PACT is as accurate as the best of them at the lowest cost.

Method	Acc.	W:L	Lower	Tokens	$/q
PACT	0.66	–	–	324K	0.036
VISTA	0.65	7:6	−0.049	504K	0.054
SLIM	0.59	11:4	+0.007	338K	0.037
Active Context Compression	0.60	9:3	+0.004	496K	0.054
Context-Folding	0.59	10:3	+0.012	406K	0.044
Stale-observation masking	0.62	6:2	−0.006	437K	0.047
Tool-result clearing	0.62	7:3	−0.012	457K	0.049
Skeleton compression	0.66	4:4	−0.047	469K	0.051
ReAct truncation	0.60	7:1	+0.015	466K	0.050

Table 12. PACT against VISTA and seven published context managers on Claude Haiku 5.5, 100 held-out BrowseComp-Plus questions (strict rule). W:L, the questions only PACT or only the method answered; Lower, the one-sided 95% lower bound of PACT's accuracy minus the method's; Tokens, prompt plus completion tokens per question over every call; $/q, list-price dollars per question.

L Extension: Ledger, Summary and Continuation Note

Overview. After the main experiments we extended PACT with three components and compared the result with ACM (Li et al., 2026b), whose agent compresses its own history into a structured memory through a `manage_context` tool. Each component was first measured on design questions; each claim below comes from a pre-registered test on questions not used to design it, unless marked exploratory. All runs use BrowseComp-Plus at $W = 32{,}768$ and $B = 163{,}840$ with the benchmark's judge.

Reasoning ledger. The task prompt asks the agent to begin every tool turn with one or two plain sentences: its candidate answers, the clues verified so far with their docids, and the next check. The notes are the agent's own text, so they stay in context when the tool results they cite are archived.

Summary at compaction. After each compaction that archives something, one extra call of the agent's model summarises the full text of the archived rounds with ACM's summary instruction. The summary enters the transcript after those rounds and is never compacted. The rounds stay in PACT's verbatim archive, behind their stubs, so the summary adds an account of what was learned without replacing any evidence.

Continuation note. With the summary, Claude models often stop early. Over 730 Claude Haiku 5.5 runs, 16% ended with "the answer could not be determined" while more than half of the budget was unspent, against 4% with the ledger alone and 1% for ACM in our harness. On the 300-question set where we traced them, 44 of 46 came at most five steps after a summary. Nemotron-3-Ultra and MiniMax-M3.1-Flash show no such stops. After each summary the harness therefore appends one fixed message: "[*Harness*] *Earlier rounds are archived and summarized in the memory above. This is not a request to answer: research budget left is about N tokens* ($P\%$). *Continue with the next step the memory names, or a better one you can see now.*" The message is charged to the budget like any harness text.

Effect of each component. Table 13 adds the components one at a time on Claude Haiku 5.5. The ledger and the summary were developed on 530 design questions. The ledger's gain was confirmed on 300 questions unseen at the time (0.673 vs. 0.633, one-sided $p = 0.048$). The summary missed its development bar of five points. Its gain over the ledger was confirmed on Nemotron-3-Ultra (13:2, $p = 0.004$) but not on Claude Haiku 5.5 (18:14, $p = 0.298$) or MiniMax-M3.1-Flash (9:3, $p = 0.073$). The note was tested in a pre-registered comparison on 200 questions, with both arms run fresh through OpenRouter's Anthropic endpoint. There it raised accuracy from 0.645 to 0.695 (17:7, one-sided $p = 0.032$) and cut early stops from 43 to 14 ($p = 2 \times 10^{-7}$), at 1.15× the cost. It fired in 16 of the 17 questions it gained.

Added to	n	Before	After	W:L	Cost
PACT: ledger	530	0.628	0.658	45:29	0.79
+ ledger: summary	530	0.658	0.696	54:34	0.84
+ summary: note	200	0.645	0.695	17:7	1.15

Table 13. Adding the extension's components on Claude Haiku 5.5. Before and After, accuracy without and with the component; W:L, questions only the version with it or only the one without it answered; Cost, tokens (first two rows) or dollars (last row) relative to the version without it. Two-sided $p = 0.081$ and 0.042 for the first two rows (design data); one-sided $p = 0.032$ for the note (pre-registered).

Comparison with ACM in one harness. We ported ACM's mechanism into our harness: its two tools, its prompts and its summariser, with the searcher, budget and judge shared with PACT. Table 14 lists the pre-registered comparisons. PACT with the ledger and the summary is non-inferior to ACM at a margin of 0.05 on Claude Haiku 5.5, Nemotron-3-Ultra and MiniMax-M3.1-Flash. It uses 0.55× and 0.73× ACM's tokens on the first two; on MiniMax most runs of both methods exhaust the budget and the tokens are equal. On Claude Sonnet 5.5 the test fails, partly because several of PACT's Sonnet runs ended without any answer, which ACM's never did. With the continuation note, on 200 further questions (both methods run fresh through OpenRouter), PACT scores 0.715 against ACM's 0.665 (22:12) with 0.64× its tokens ($p = 7 \times 10^{-10}$). It runs out of budget on 3 questions against 55, and is more precise when it answers (0.88 vs. 0.75). Its non-inferiority is clear (lower bound +0.002), but the pre-registered test of higher accuracy was not met ($p = 0.061$).

Backbone	PACT	ACM	W:L	Lower	Tokens
Haiku 5.5	0.695	0.685	15:13	−0.034	0.55
Sonnet 5.5	0.680	0.750	4:11	−0.133	0.65
Nemotron-3	0.560	0.480	14:6	+0.008	0.73
MiniMax-M3.1	0.350	0.240	14:3	+0.045	0.99
Haiku, + note	0.715	0.665	22:12	+0.002	0.64

Table 14. PACT with the ledger and the summary (last row: and the note) against ACM's mechanism in our harness. Haiku: 200 held-out questions (last row: 200 further ones); the others: 100. Lower, the one-sided 95% lower bound of the accuracy difference (non-inferiority margin −0.05); Tokens, PACT's tokens per question relative to ACM's.

The tool layer, and official ACM. ACM's own runner also changes what the tools return: a document already seen in the episode comes back as a 128-token preview, and a fetched document is cut at 8,192 tokens. We added this view to our harness as an option applied to every method alike. On MiniMax it lifts ACM's mechanism from 0.240 to 0.410 (22:5, two-sided $p = 0.002$), because ACM's runs exhaust the budget less often (44 against 60), and PACT only from 0.350 to 0.430 (13:5). With the view on both, the two are level (10:8; the pre-registered non-inferiority test was not met, lower bound −0.050). PACT archives large results in any case, so on MiniMax most of its lead over ACM's mechanism without the view comes from the tool layer. On Nemotron-3-Ultra, with the view on both, PACT stays non-inferior (0.640 vs. 0.610, 11:8, lower bound −0.042; pre-registered) with 0.73× ACM's tokens. In its own runner, official ACM is more accurate than PACT with the ledger and the summary on Claude Haiku 5.5, 0.756 against 0.696 over 730 questions (PACT 33:77), while PACT uses 0.62× its tokens. It is also more accurate on MiniMax, 0.510 against PACT's 0.430 with the view (PACT 7:15; pre-registered non-inferiority not met), at about the same tokens (0.94×). There, PACT exhausts the budget and then answers wrongly in most of the questions official ACM gains.

Commit-with-recall, and official ACM on Nemotron. PACT's own commit-with-recall (§3) targets exactly these runs: at 90% of the budget it ends research and answers from the recalled archive. Added to the ledger and the summary, with the tool view, it raised accuracy on 100 MiniMax design questions from 0.450 to 0.650 (23:3; 13:2 where the commit fired). In a pre-registered confirmation on Nemotron-3-Ultra its gain over the version without it was not significant (0.720 vs. 0.640, 14:6, $p = 0.058$), but against official ACM in its own runner it was non-inferior and nominally more accurate: 0.723 against 0.566 on the 83 questions both completed (PACT 16:3, lower bound +0.075;

superiority, not pre-registered, $p = 0.002$), at 2.2× official ACM's tokens. Official ACM's other runs failed on the endpoint's rate limits and are excluded. A pre-registered replication on the 200 questions of a second Nemotron-3-Ultra set, with both runners riding out the rate limits and no question excluded, confirmed it: PACT scored 0.660 against official ACM's 0.570 (PACT 25:7, lower bound $+0.045$), non-inferior ($p = 2 \times 10^{-7}$) and more accurate ($p = 0.001$; both hold after Holm), at 2.1× official ACM's tokens. On 11 of PACT's wins official ACM gave no answer: it reached its 100-turn cap, or its runner ended the conversation when the context filled. On the other questions PACT won 14:7 (exploratory). On MiniMax-M3.1-Flash the same pre-registered test, on the 100 questions of a set that informed our diagnosis of official ACM's lead there (both arms on the same free endpoint; PACT's requests were streamed, since the endpoint ends a non-streamed reply after about 300 s), found PACT non-inferior: 0.610 against official ACM's 0.540 (PACT 15:8, lower bound -0.008, $p = 0.006$), but not more accurate ($p = 0.105$), at 0.67× official ACM's tokens. Official ACM gave no scorable answer on 38 questions and PACT on 19; on 13 of PACT's wins official ACM gave no answer, and on the questions it answered PACT won 2:8 (exploratory). We therefore claim token efficiency and non-inferiority to ACM's mechanism in one harness, higher accuracy than official ACM on Nemotron-3-Ultra at a higher token cost, and non-inferiority to it on MiniMax at fewer tokens, but not on Claude Haiku 5.5, where official ACM in its own runner remains more accurate than PACT in ours (inside ACM's runner PACT is at least as accurate on Haiku and MiniMax; see below). On Claude Haiku 5.5 the full extension (ledger, summary, note, commit-with-recall and the tool view) scored 0.725 against official ACM's 0.780 on the 200 questions of the second held-out set (PACT 8:19, lower bound -0.097; pre-registered non-inferiority not met), with 0.67× its tokens.

Errors are abstentions (exploratory). Over the Claude Haiku 5.5 sets, PACT with the ledger and the summary answers wrongly on 7.9% of questions, ACM's mechanism in our harness on 18.2%, and official ACM on 8.6%. When PACT answers, it is right 0.90 of the time. Most of its errors are therefore abstentions, which the continuation note targets. This comparison was not pre-registered.

Where official ACM's lead comes from (exploratory). Both runners query the same searcher. In the Claude Haiku 5.5 comparison of the full extension with official ACM (0.725 against 0.780), 48 of the 55 queries that both issued word for word returned the same top-five list, and the two surfaced an evidence document on almost as many questions (182 and 185 of 200). On the 19 questions only official ACM answered, PACT abstained on 12, 6 of them with a gold document already among its search results. On MiniMax, PACT with the ledger and the summary (without commit-with-recall) exhausted the budget on 15 of the 15 questions only official ACM answered, though both surfaced evidence on as many questions (90 and 89 of 100). Official ACM's lead thus comes from when the agent stops, not from what it retrieves.

PACT inside ACM's own runner. When the agent stops is shaped by the runner as well as by context management. We therefore ran PACT inside official ACM's runner, unchanged except for its context interface: the `manage_context` and `query_memory` tools, the context-size marker after each tool result, and the prompt text about them are removed, and PACT with the ledger, the summary, the note and commit-with-recall manages the context instead, within ACM's usable window (W less its 10,000-token output reservation). In a pre-registered test on the 300 questions of a third Claude Haiku 5.5 set (one that informed the diagnosis behind the continuation note), it scored 0.767 against official ACM's 0.743 (PACT 23:16, lower bound -0.011, non-inferiority $p = 2 \times 10^{-4}$) with 0.80× its tokens ($p = 0.004$), \$11.22 against \$13.79 at list price. It never exhausted the budget, where official ACM did 23 times; its higher accuracy is not significant ($p = 0.168$). With the runner

held fixed, PACT's context management is at least as accurate as ACM's at lower cost, so official ACM's lead over PACT in our harness comes from its runner. The same pre-registered test on the 100 questions of a MiniMax-M3.1-Flash set (one that informed our diagnosis of official ACM's lead on that backbone) found more: PACT scored 0.660 against official ACM's 0.540 (PACT 16:4, lower bound $+0.049$), non-inferior ($p = 4 \times 10^{-5}$) and more accurate ($p = 0.006$; both hold after Holm), with $0.95\times$ its tokens (not significantly fewer, $p = 0.190$). Official ACM exhausted the budget 35 times and PACT 10; on 14 of PACT's wins official ACM gave no answer, and on the questions it answered PACT won 2:4 (exploratory).

M GAIA Test

Setup. GAIA (Mialon et al., 2024) asks questions that need web search, reading attached files and several steps of reasoning. Its 301 test tasks have hidden answers, and 71 of them come with an attachment. Both methods run the VISTA authors' GAIA client with Wikipedia search, a 12,288-token window and a budget of 80,000 new tokens, with the Qwen3.8-27B backbone in bf16, thinking off and the settings used on the 165 validation tasks. Our vLLM server ran 234 of the 301 tasks per method, and a hosted bf16 deployment of the same weights ran the other 67, both methods of a task on the same deployment. PACT runs as on BrowseComp-Plus (§3.4) with commit-with-recall (§3.3) and one more small-window mechanism. A tool result larger than half the window, which the next compaction would release unread, is archived verbatim when it arrives, and the agent sees a note on its size and its passages most relevant to the task, ranked as in commit-with-recall, within a quarter of the window.

Pre-registration. The test was registered before any test task ran. Its configuration was fixed on the validation tasks, where it scored 0.376 and VISTA 0.382. The leaderboard reports only aggregate accuracy, so the primary hypothesis, non-inferiority at a margin of 0.10, is judged by the one-sided 95% Wald lower bound of the unpaired difference. The secondary hypothesis is a lower uncached prompt cost per task, by a one-sided paired t test over the same tasks. Client timeouts and connection errors were rerun once. Any other error counts against the method and is submitted as an empty answer.

Results. PACT scores 0.346 and VISTA 0.355, a difference of -0.010 with a 95% interval of $[-0.086, +0.066]$ and a one-sided lower bound of -0.074, above the margin of -0.10. PACT recomputes 505K prompt tokens per task and VISTA 1,043K ($p < 0.001$), so VISTA spends $2.01\times$ PACT's uncached tokens per correct answer. VISTA exhausted its budget in 145 runs and PACT in 2. VISTA's 6 errors are requests whose prompt outgrew the server's context window, and they count as wrong answers under the rule; PACT had 0. By level, PACT scores 0.484, 0.340 and 0.102, and VISTA 0.462, 0.358 and 0.143.

Claude Haiku 5.5. A second pre-registered test repeats the comparison on all 301 test tasks with Claude Haiku 5.5 as the backbone, through Anthropic's API with the model's default sampling and adaptive thinking, and otherwise the same settings. The API reports no cache hits, so the cost hypothesis counts every token per task, prompt and completion. PACT scores 0.372 and VISTA 0.346, a difference of $+0.027$ with a 95% interval of $[-0.050, +0.103]$ and a one-sided lower bound of -0.038. PACT uses 122K tokens per task and VISTA 162K, 25% fewer ($p < 0.001$). No run exhausted its budget or failed; one PACT run ended on an empty reply from the model and counts as wrong.

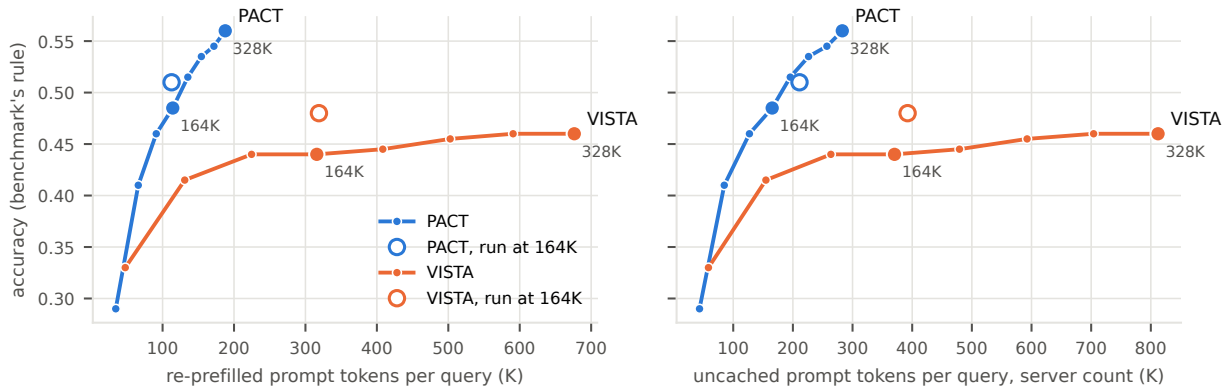

Figure 12. Accuracy (benchmark's rule) against prompt tokens per query as the trajectory budget grows to 327,680, read off the runs at that budget (200 held-out queries); left the instrument's count, right the server's. Large markers: 163,840 and 327,680. Hollow markers: the runs made at 163,840, which completion read off the larger run matches on 0.83 (PACT) and 0.82 (VISTA) of queries.

By level, PACT scores 0.581, 0.321 and 0.143, and VISTA 0.527, 0.308 and 0.122. At list price the whole test cost $4.35 for PACT and $5.72 for VISTA.

N Accuracy versus Token Budget, and Server-Side Caching

Results at the largest budget. PACT at 327,680 is more accurate than VISTA at the same budget (discordant pairs 31:11, $p = 0.001$) and than VISTA at 163,840 (31:15, $p = 0.013$). At 327,680 VISTA needs 3.50× PACT's uncached tokens per correct answer (95% bootstrap $[2.79, 4.36]$), and the instrument counts 188K a query against 676K. Doubling the budget moved VISTA by -0.020 and PACT by $+0.050$ (difference $+0.070$, $[-0.005, +0.145]$). Against VISTA at 163,840, PACT at 327,680 costs 283K against 392K by the server's count and 188K against 319K by the instrument's.

Server-side cost of a cut. The backbone is hybrid (48 of its 64 layers are Gated DeltaNet). vLLM caches it in its 'align' mode, which saves the recurrent state only on a block boundary where a scheduler step ends, and a later request can resume only from such a state (Pan et al., 2025): every nonzero hit the server reports on these runs is a multiple of 1,568 tokens. On the second held-out set at the default budget (five methods on one server), the server served under half of the prefix the instrument counts as kept on 79% of PACT's cuts that keep at least 2K tokens, against 22% of its appends, and at those cuts it recomputed 2.0× what the instrument charges. Depth still matters, less than the instrument assumes: a cut keeping under 8K tokens recomputed 99% of the prompt (instrument 67%), one keeping 8–16K 94% (41%), 16–24K 76% (28%), and 24K or more 52% (14%). In the run at 327,680, with two methods on the server, far fewer appends went cold, but cuts keeping under 8K tokens still did. VISTA, which rewrites the tail of its prompt, got under half its kept prefix served on 26% of its steps (1.2× the instrument). This is consistent with PACT's saving coming from cutting rarely. Every cost claim on this benchmark rests on the server's count.

O Case Studies

Single tasks illustrate how the methods behave; they do not establish an effect, because at 64K 24 of 67 LOCA-bench tasks change outcome between two runs with identical inputs (§5.6). Each case below was chosen by a fixed rule from the tasks on which the methods disagree, and the counts before it show how typical its pattern is.

Failure patterns on LOCA-bench. At 64K, on the 71 tasks that every method completed, PACT is right and our ReAct wrong on 15. In 9 of them the runner's truncation removed every assistant and tool message, which ends the episode, and in 3 the agent reached the 35-step limit. PACT is right and VISTA wrong on 10; in 7 of them VISTA reached the step limit after a median of 7 steps spent on archive and recover actions. PACT compacted in 16 of its 18 winning tasks, and its truncation fallback never fired in them. In the other direction, VISTA is right and PACT wrong on 11 tasks, 6 of them at the step limit and 5 with a wrong submission.

Case 1: a large tool result (LOCA-bench, 64K). Selection: among the 7 tasks where both baselines failed, those where each failed by its most common mechanism (truncation for ReAct, the step limit for VISTA), and of these the one whose number of VISTA context-tool steps equals the median of 7. The task, UpdateMaterialInventory (seed 456), asks the agent to read new paid WooCommerce orders, compute raw-material use from a bill of materials in a spreadsheet, and update the inventory sheet and the store's stock. All three agents request a page of 100 orders, a result of about 55,000 tokens (Table 15, Figure 13). Our ReAct appends it at step 2, and the runner's truncation then removes all nine assistant and tool messages, which ends the episode. VISTA's admission limit rejects the page. VISTA then pages through the orders ten at a time and spends 7 of its 35 steps on archive and recover actions ("the earlier archives didn't take"); it computes the correct totals and writes the same sheet values as PACT, but reaches the step limit before any stock update. PACT archives the page at step 4, which shrinks the prompt from 79,787 to 16,680 tokens, and the agent queries the archived file with shell and Python commands in later steps; it submits at step 21. The evicted block was the newest one, so the compaction discarded no cached tokens.

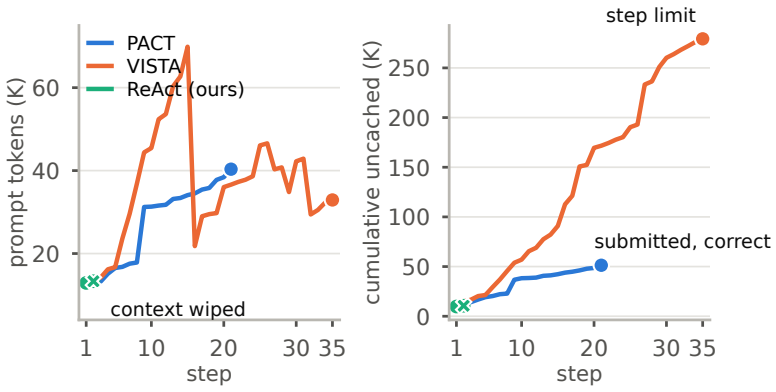

Figure 13. Case 1: prompt tokens per step (left) and cumulative uncached prompt tokens by the server's count (right). PACT submits at step 21 after 51K uncached tokens; VISTA reaches the step limit after 279K; our ReAct loses its context at step 2.

Failure patterns on BrowseComp-Plus. On the first held-out set, PACT with commit-with-recall is right and VISTA wrong on 15 queries, and the reverse holds on 4. All 15 are VISTA runs that exhausted the budget; in 4 of them VISTA's forced answer matches the gold answer, so they turn on

Method	Step	Action	Prompt	Event
PACT	4	`orders_list(perPage=100)`, `products_list`	15,143	55,309-token result archived; prompt 79,787 → 16,680
	5	shell command on the archived file	16,512	archive read
	14–15	Python on the archived file	34,085	"paid orders are all 120"
	18	Python: material consumption	35,818	demand totals
	19–20	sheet update; four stock updates	38,412	
	21	`claim_done`	40,342	submitted, correct
VISTA	4	`orders_list(perPage=100)`	16,204	rejected by the admission limit
	5	`orders_list(perPage=10)`	16,717	"The orders list is too large"
	11	archive four blocks	52,384	dashboard action
	15	archive blocks B16–B38	69,871	"the earlier archives didn't take"
	16	next page of orders	21,794	51,369 cached tokens discarded
	19	recover a block	29,751	"Orders 61-70 weren't saved"
	33	Python: material consumption	30,518	same totals as PACT
	35	sheet update	32,923	step limit; no stock update
ReAct (ours)	2	`orders_list(perPage=100)` and two other calls	13,322	result of 191,218 characters
	–	(runner, no model call)	–	"Context trimming removed all 9 assistant/tool messages"

Table 15. Case 1, key steps (prompt tokens as sent). The authors' ReAct also lost its context, at model call 5.

the strict rule. Of the 15 PACT runs, 8 answered before the commit line and 7 through the commit call, and in 6 of these 7 the gold answer appears among the run's tool results. Across the set the commit call fired in 48 of 100 runs and was right in 11. On the second set, plain PACT is right while VISTA and the VISTA authors' ReAct baseline with truncation are both wrong on 11 queries, 9 of them with both baselines out of budget.

Case 2: answering within the budget (BrowseComp-Plus). Selection: among the 7 queries won through the commit call, the median by number of requests (29). Both agents see the answer, the band TENDER, in a search result by their second request (Table 16). VISTA lists it in its own archive summary at call 7 but rejects it ("None matched all criteria yet"), issues one identical query 37 times, drops TENDER from later summaries, and exhausts the budget at call 58; its closing call answers "5 Seconds of Summer". PACT with commit-with-recall also repeats searches, and the harness compacts 11 times, but when the meter passes 0.9 of the budget at request 29, one call without tools over the recalled passages answers "TENDER" and cites four documents. Plain PACT also found TENDER on its own for this query, so the query separates commit-with-recall from VISTA, not from PACT.

Case 3: a loss (LOCA-bench, 64K). Selection: PACT's most common loss mode is the step limit (6 of its 11 losses to VISTA), and of these the lower median by PACT's peak prompt. In PayableInvoiceChecker (seed 2024), PACT archives a 34,218-token database dump at step 4 and later queries the archive successfully (550 invoice dates at step 10). The agent then calls a PDF tool name that the server does not provide, four times, and tries to parse the receipts by hand until it reaches the

Method	Request	Action	Prompt	Event
PACT+commit	2	two searches	6,209	a result contains TENDER
	3	search "TENDER band Nadir …"	11,668	the agent names the candidate
	6	two searches	16,117	compaction, six results stubbed
	9	two searches	20,071	both calls repeat earlier calls (loop guard)
	28	two searches	25,410	eleventh compaction
	29	commit call, no tools	31,979	meter 149,936 of 163,840; answers "TENDER", correct
VISTA	2	two searches	6,778	a result contains TENDER
	7	archive	33,044	own summary lists TENDER: "None matched all criteria yet"
	16	search	8,236	first of 37 identical queries
	40	archive	33,756	TENDER no longer in the summary
	58	search	16,443	budget exhausted (162,809)
	59	closing call, no tools	15,980	forced answer "5 Seconds of Summer"

Table 16. Case 2, key requests on the first held-out BrowseComp-Plus set (prompt tokens as sent).

step limit. VISTA's admission limit rejects the same dumps; VISTA switches to counting queries, reads the eight receipts with the PDF reader and submits at step 23. Nothing in the log ties this loss to an evicted block: it is a tool-use error that the context policy neither causes nor repairs.

Case 4: ending a search early (Claude Haiku 5.5, BrowseComp-Plus). Selection: among the held-out questions that PACT answered correctly and at least four of the seven published managers of Appendix K got wrong, the one with the most failing baselines, then the most that spent their budget, then PACT's fewest tokens. The pattern is not rare. PACT is right and at least four baselines are wrong on 7 of the 100 questions, and on all 7 most of the failing baselines spent their budget. PACT is also right with fewer tokens than every baseline on 13 questions.

PACT answered in 16 steps (Table 17). It compacted four times and archived 26 search results, so its prompt fell back to about 20K tokens after each compaction and never sat at the limit. Five baselines filled the window by their sixth step and then re-sent 40–46K tokens on every step until the budget ran out, 27 or 28 steps in, without an answer. VISTA also answered, after 31 steps and 2.5× PACT's tokens. Two lossy managers, SLIM and Active Context Compression, answered in 9 steps with about half of PACT's tokens, so on this question their summaries kept what was needed.

Method	Correct	End	Steps	Peak prompt	Tokens
PACT	yes	answered	16	39K	421K
VISTA	yes	answered	31	51K	1038K
SLIM	yes	answered	9	44K	221K
Active Context Compression	yes	answered	9	45K	212K
Context-Folding	no	budget spent	28	45K	1132K
Stale-observation masking	no	budget spent	28	45K	1080K
Tool-result clearing	no	budget spent	27	46K	1047K
Skeleton compression	no	budget spent	27	46K	1047K
ReAct truncation	no	budget spent	28	45K	1124K

Table 17. Case 4, one held-out BrowseComp-Plus question on Claude Haiku 5.5. Correct under the strict rule; a run that spent its budget ended without an answer. Tokens are prompt plus completion over every call.

P Evaluation Integrity

Harness repairs. (1) The benchmark installs its mock services from an unpinned dependency; ours resolved a major version under which the Google Cloud, Sheets, WooCommerce, calendar and Snowflake servers failed at start-up, and the runner continued without them. Fixed by a lockfile shared by both codebases and a per-environment tool check that fails a run unless every declared server registers. (2) The Python and terminal tools ran unconfined beside each task's ground truth, which agents read in 7–21% of episodes of every method; they now run in a `bwrap` sandbox whose test also asserts the probe *can* see the answer key when nothing is hidden. (3) A 50-call tool-use limit silenced every tool, including the submission tool, from about turn 18; the pre-flight check now drives an episode past 50 calls. (4) The two codebases' step limits differed by one and a 60-second request timeout truncated long prompts; both are matched and asserted. In the analysis, significance was dispatched by metric name, so accuracy received a paired t instead of McNemar's exact test, and "steps" carried two definitions across codebases; both are fixed and the step count comes from the instrument both codebases share. The sandbox of (2) at first bound the benchmark under one spelling of its path while the Python server was launched under another, so our methods ran without a Python tool; a comparison of the tools offered at step 1 across codebases caught it, the affected runs were discarded, and the pre-flight check now fails on any missing server.

Residual access to benchmark sources. The sandbox binds the benchmark's source tree read-only because the confined tool servers run from it, so a confined agent can list and read each environment's source, graders and data generators included (the per-task answer keys stay hidden). A normal trajectory never contains that path. It appears in 12 of the 1,275 reported episodes (four for ReAct, seven for PACT, five of them with folding, and one for VISTA), mostly as file listings from a broad `find`, and all twelve scored zero. Every method had the same access, so no paired comparison is affected. We did not change the sandbox during the study, so that every LOCA-bench comparison shares it.