CONTEXT LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce **Context Language Models (CLMs)**, language models that natively manage their own context. We implement this by treating the context as an editable file; this naturally extends to multi-agent systems where multiple agent contexts coexist as files. Applying CLMs out of the box outperforms SOTA context-management strategies across a variety of tasks: 11.4% higher accuracy with 21.5% fewer FLOPs on BrowseComp-Plus, 5% higher scores with 59% fewer FLOPs on 12-hour EdgeBench, and a 65% larger speedup gain at equal spend on a 24-hour multi-repository agent-swarm task. Moreover, by shifting context management from external harness control to intrinsic model behavior, CLMs naturally enable both **in-context and parametric learning** of context-management strategies. We show that CLMs can be steered with natural-language instructions evolved through a standard skill-optimization loop, improving held-out accuracy by up to 26.9 points on a context-management task while reducing compute. We also introduce an online reinforcement learning method for CLMs, improving Qwen3.5-9B performance on BrowseComp-Plus by 47.6% while using 12% fewer FLOPs. Finally, we co-design **Suffix Cache Reuse** for CLM serving, further reducing compute by 35% relative to standard SGLang at matched performance.

1 INTRODUCTION

Context is a cornerstone that allows a language model (LM) to process and retain information over time. Context management is not traditionally a native LM capability. Instead, prior work mostly relies on harnesses, either hand-engineered (Cassano & Rush, 2026; OpenAI, 2026; Merrill et al., 2026) or optimized offline by agents (Lee et al., 2026). Recent work adds a constrained set of tools such as compaction, offloading, and retrieval, to the agent's action space (Yan et al., 2026; Yu et al., 2026; Li et al., 2026c; Liu et al., 2026; Zhang et al., 2025a). In contrast, we show that giving LMs unrestricted access to their context outperforms human-designed baselines, enabling adaptive and creative context-management strategies to emerge. Our findings echo The Bitter Lesson (Sutton, 2019): we should let LMs search for and learn better strategies beyond human priors.

Concretely, we introduce **Context Language Models (CLMs)**, which are natively capable of managing their own context. We show existing LMs can be turned into strong CLMs and can be further improved through in-context learning and reinforcement learning. Formally, a CLM parametrized by θ makes context an artifact of the LM: $c_{t+1} = f_\theta^{\text{CLM}}(c_t)$, where f_θ^{CLM} can be an arbitrary function controlled by CLMs. In contrast, a standard LM simply appends new tokens to the existing context: $c_{t+1} = c_t \oplus f_\theta^{\text{LM}}(c_t)$.

We implement CLMs by treating **context as a file**. Specifically, we mirror the context into a storage space with LM write access. The LM can either append newly generated tokens or use Bash to freely edit the context file, with each modification immediately synchronized to the LM's live context for the next turn. This design naturally extends to multi-agent systems, where multiple context files can coexist and be managed by CLMs for *agent-swarm* or *subagent* workloads.

Arbitrary context edits in CLMs bring new challenges for existing serving systems because they typically reuse cached states only for matching prefixes, forcing re-prefilling after in-the-middle edits. To account for this, we introduce **prefix-reuse FLOPs**, which capture trajectory-wide inference cost accounting for decoding, prefilling, and re-prefilling in standard LM serving, and show that CLMs remain more compute-efficient under standard serving through better context management. We further develop **Suffix Cache Reuse (SCR)**, which reuses cached states beyond the matching prefix to

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

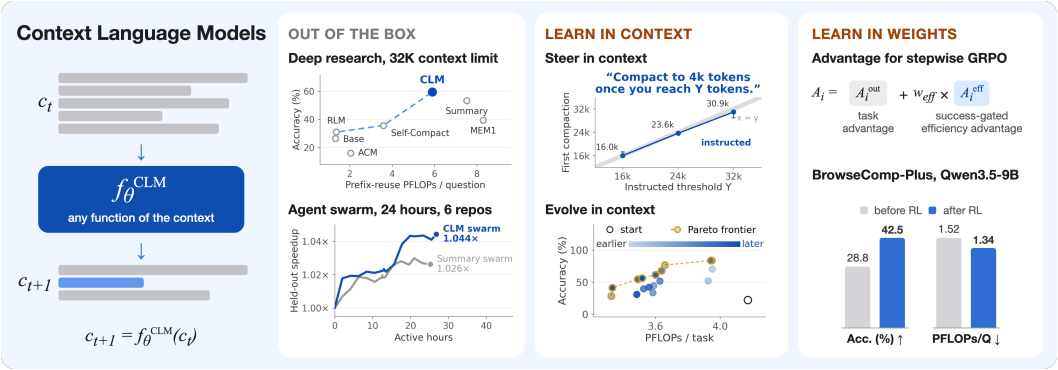


Figure 1: **Context Language Models natively manage their own context.** CLMs work out of the box and can be further improved through in-context learning and reinforcement learning.

reduce re-prefilling while empirically preserving task performance. In addition, we introduce **ContextBench** as a diagnostic benchmark for fine-grained context management. By decoupling context management from reasoning and knowledge, ContextBench reveals the limitations of existing context management methods.

We show that CLMs, applied out of the box, outperform existing baselines and task-specific harnesses across diverse long-horizon tasks, ranging from hundreds to thousands of turns and up to 24 hours of runtime. Compared with existing harness-defined and action-based methods, CLM achieves 11.4% higher accuracy with 21.5% fewer prefix-reuse FLOPs than the strongest baseline on the deep-research benchmark BrowseComp-Plus (Chen et al., 2025), while matching the strongest baseline's accuracy on the terminal-coding benchmark TerminalBench 2.1 (Merrill et al., 2026) with 29.5% fewer FLOPs. On mathematical optimization tasks, CLM outperforms specialized evolutionary harnesses such as OpenEvolve (Sharma, 2025) by up to 16.8% (Heilbronn) and 3.0% (circle packing). On long-running software optimization, CLM outperforms Codex-style summarization: on 12-hour EdgeBench (Zhu et al., 2026) (a 10-task subset), CLM scores 5% higher while using 59% fewer prefix-reuse FLOPs, and on a 24-hour six-repository agent-swarm task, it achieves a 65% larger speedup gain at equal spend. Moreover, when Suffix Cache Reuse is further applied, it helps reduce server-side compute by 35% with matched performance compared with standard SGLang serving. We also show qualitative examples from CLMs, revealing novel emergent behaviors such as defining and maintaining trackers for multi-agent orchestration, introducing a new chat role for internal notes, and defining reusable context-management functions.

By shifting context management from external harness control to intrinsic model behavior, CLMs naturally enable **in-context learning** and **parametric learning** for context management. We first show that users can **steer** context management simply by telling the agent their desired strategy. In addition, CLMs can **evolve** an in-context skill document that captures useful context-management procedures for future reuse, improving held-out accuracy on ContextBench by up to 26.9 points at lower compute. For training, we introduce a **success-gated efficiency advantage** in stepwise GRPO (Shao et al., 2024) that rewards efficient CLM trajectories among successful ones. Training Qwen3.5-9B on deep research tasks in this manner improves CLM from 28.8% to 42.5% on BrowseComp-Plus, outperforming a Codex-style summary harness trained with the same recipe by 0.4 points while using 38.8% fewer FLOPs. Overall, we show that context management can be a native LM capability, and CLMs enable better strategies to be searched for and learned.

2 Related Work

From harness-defined to action-based context management. Most existing harnesses compact accumulated histories according to a fixed harness policy (Cassano & Rush, 2026; OpenAI, 2026; Merrill et al., 2026; Zhou et al., 2026), such as at a predefined length threshold or at every turn. Recent work gives the model increasing control through human-defined actions: AutoCompact (Zhang et al., 2026b) and Self-Compact (Li et al., 2026b) let the model decide when to compact; Context-as-a-Tool (Liu et al., 2026) exposes model-triggered compaction over a predefined portion of the

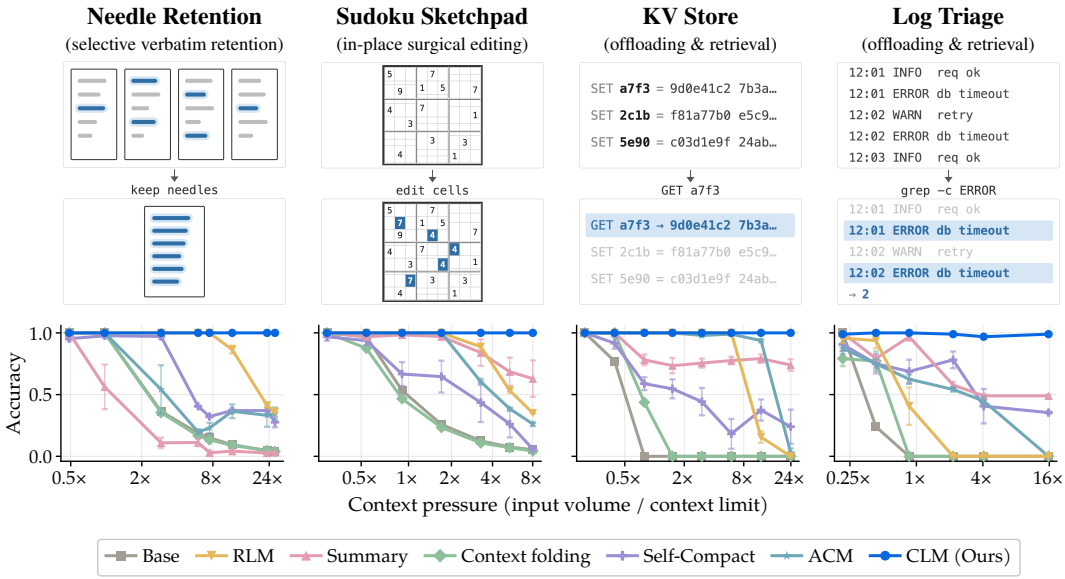


Figure 2: Illustration of the four tasks in ContextBench and a performance comparison of CLM against baselines using GPT-5.4 with a 32K context limit.

context; ACM (Li et al., 2026c) adds model-triggered offloading and retrieval; and Sculptor (Li et al., 2026a) lets the model select context fragments to operate on. Across this progression, model autonomy increases but remains restricted to a human-defined action space. Our work pushes this autonomy to its limit by granting the model full agency over its context.

Context as a REPL variable. Recursive Language Models (RLMs) (Zhang et al., 2025a) treat a long input as a read-eval-print loop (REPL) variable that LMs can recursively access on demand. This addresses when and what information to *read* into the context. However, RLMs do not address how the live context itself should be managed. Retrieved information is still appended to the live context, which continues to grow over time. In contrast, our work makes the live context editable, giving CLMs full control over their context.

3 Pilot Study with ContextBench: A Diagnostic for Context Management

We start with a pilot study showing how existing context management strategies can fail in simple tasks. To isolate context management from other reasoning or knowledge capabilities, we developed **ContextBench**, a diagnostic evaluation suite with the four synthetic tasks shown in Figure 2 (details in Appendix B): **Needle Retention** tests selective verbatim retention, simulating the need to preserve important information over time; **Sudoku Sketchpad** tests surgical in-place updates to the live context by maintaining a Sudoku board as users stream in moves; **KV Store** and **Log Triage** test exact recall through offloading and retrieval of massive values and working logs. We run ContextBench with different context management strategies, including Mini-SWE-Agent (Yang et al., 2024) (the base harness without context management), RLM (Zhang et al., 2025a), Codex-style Summary (OpenAI, 2026), MEM1 (Zhou et al., 2026), Context Folding (Sun et al., 2025), Self-Compact (Li et al., 2026b), and CLM as will be introduced in Section 4. We fix the context limit at 32K and vary the context pressure (the ratio of input volume to context limit) up to $24\times$.

The results in Figure 2 show that these fixed strategies cannot adapt well to the live context: Summary-based compaction can lose or hallucinate information on Needle Retention and Sudoku Sketchpad; methods without flexible in-place editing must regenerate the full Sudoku state for ev-

ery fine-grained user edit[1]; and standard coding tools can offload information on KV Store and Log Triage but cannot evict it from the live context on demand. As a result, none of the existing methods performs perfectly even on these simple tasks. These failures motivate fully adaptive, model-controlled context management.

4 Context Language Models (CLMs)

4.1 Formal Definition and Implementation

Context Language Models (CLMs) generalize the append-only context transition of a standard LM to a model-controlled context transition. Standard LMs append model output to the current context:

$$c_{t+1} = c_t \oplus f_\theta^{\mathrm{LM}}(c_t), \tag{1}$$

where $\oplus$ denotes concatenation. In contrast, CLMs delegate full responsibility for maintaining the context to the CLM itself, directly creating the next context:

$$c_{t+1} = f_\theta^{\mathrm{CLM}}(c_t), \tag{2}$$

where f_θ^{CLM} can be arbitrary context manipulations. Eq. 2 subsumes prior approaches that expose a set of context-management tools through the harness. However, prior work requires context-management functions to be predefined in the harness. Our work instead makes CLMs responsible for defining these functions themselves as a meta-capability, either explicitly as reusable tools or implicitly through their planning.

Context-as-a-file implementation for CLMs. To implement CLMs, we mirror the LM's live context as a directly editable file and provide its path in the system prompt. The LM can edit this file using general Bash commands, just as it would edit other files in storage. Unlike ordinary files, edits to the context file are automatically synchronized with the LM's context and sent to the LLM server for continued generation. When the LM does not edit the context file, the generated tokens are appended to the existing context by default. This implementation balances context reuse with the flexibility to edit the context.

Multi-agent extensions of CLMs. Our implementation naturally extends to multi-agent workflows by allowing multiple context files to coexist and remain synchronized with their respective LLM servers. For example, an **agent swarm** can be implemented by initializing the workspace with multiple context files, while **subagents** can be initialized and terminated by creating and deleting additional context files.

Efficiency metrics for CLMs. A common serving optimization is prefix-cache reuse, in which cached states are reused for matching prefixes, while all tokens from the first prefix mismatch onward must be re-prefilled, as can occur after an in-the-middle edit. To account for this, we measure theoretical inference FLOPs using a metric we call **prefix-reuse FLOPs**. Formally,

$$\mathrm{FLOPs}_{\text{prefix-reuse}} = \underbrace{\mathrm{FLOPs}_{\mathrm{prefill}}\big(\text{unmatched context suffix}\big)}_{\text{tokens from the first prefix mismatch onward}} + \underbrace{\mathrm{FLOPs}_{\mathrm{decode}}\big(\text{generated tokens}\big)}_{\text{new output tokens}}. \tag{3}$$

4.2 In-Context Learning and Reinforcement Learning for CLMs

By treating context management as an LM-native capability, CLMs can learn better strategies in context or in weights.

Steering CLMs with in-context instruction or skill documents. Let s denote an in-context instruction or skill document. CLMs can be steered by simply providing s as additional in-context guidance to the CLM:

$$c_{t+1} = f_\theta^{\mathrm{CLM}}(c_t; s). \tag{4}$$

[1]Sculptor (Li et al., 2026a) should support in-place string replacement, but we could not include it because no official code is available.

Evolving CLMs with an optimization loop. CLMs can also be optimized through textual evolution. For task instance x, let $\tau(x; s)$ be the trajectory induced by Eq. 4, and $R(\tau)$ a trajectory-level reward. We optimize

$$s^* = \arg\max_s \mathbb{E}_{x\sim\mathcal{D}} \left[R(\tau(x; s))\right], \tag{5}$$

while keeping everything else fixed. In our implementation, we use a prompt-evolution loop (Agrawal et al., 2026): In each round, the agent produces rollouts on the training split, and a proposer model uses the resulting traces to generate candidate skills. We evaluate these candidates on the development split and select the skill for the next round. After evolution concludes, we evaluate the final selected skill once on the held-out test split. The optimizer may be either a stronger external model (*assisted evolution*) or the agent model itself (*self-evolution*), allowing context-management skills to evolve in context.

Reinforcement Learning for CLMs. CLMs can also learn context-management strategies through reinforcement learning and internalize them in model weights. Since context edits change the input across turns, we use stepwise GRPO (Shao et al., 2024). For each prompt, we sample a group of complete agent trajectories and compute the standard GRPO advantage from their trajectory-level outcome rewards. We then assign each trajectory's advantage to all of its constituent segments, so every model call is trained with the outcome of the full trajectory.

Outcome rewards provide only weak supervision for context editing, as successful trajectories can contain inefficient edits and failed trajectories useful ones. Simply rewarding edit frequency or removed context volume is also undesirable, as the LM may reward-hack by making unnecessary edits that discard important information or hurt prefix reuse. We therefore introduce a **success-gated efficiency advantage** that rewards lower inference cost only among successful trajectories. Let c_i denote the prefix-reuse FLOPs of trajectory τ_i, and let $\mathcal{G}_g^+$ denote the successful trajectories in group g. We define $\bar{c}_g = \frac{1}{|\mathcal{G}_g^+|}\sum_{k\in\mathcal{G}_g^+} c_k$, and

$$A_i^{\text{eff}} = \begin{cases} \text{clip}\left(\dfrac{\bar{c}_g - c_i}{\bar{c}_g}, -1, 1\right), & i \in \mathcal{G}_g^+, \\ 0, & i \notin \mathcal{G}_g^+. \end{cases} \tag{6}$$

When there are fewer than two successful trajectories in a group, we set $A_i^{\text{eff}} = 0$ for all trajectories. Thus, the efficiency signal only re-ranks among successful trajectories by inference cost. We combine outcome and efficiency advantages as $A_i = A_i^{\text{out}} + w_{\text{eff}} A_i^{\text{eff}}$ to encourage trajectories that are both correct and efficient.

4.3 (More) Efficient CLM Serving with Suffix Cache Reuse

What if we want to serve CLMs even more efficiently and reduce the re-prefilling overhead? We introduce **Suffix Cache Reuse (SCR)**. As shown in Figure 3, when B is replaced by B' after an edit, SCR reuses the cached states of all surviving tokens, including C, and only reprefills the newly inserted or appended tokens B'. Surviving suffix tokens C thus retain stale cache states that encode the previous prefix, which can even be beneficial in some cases, as it retains richer information from the past. By contrast, standard prefix-cache reuse must re-prefill all tokens after the first mismatch (B' and C). In some standard chat-serving setups, reasoning tokens from previous turns are stripped before the next turn, forcing the preserved suffix to be re-prefilled. SCR can also reuse the cached states of these preserved tokens. **Throughout the paper, we evaluate efficiency under standard serving and report the additional compute savings from SCR separately in Section 5.3.**

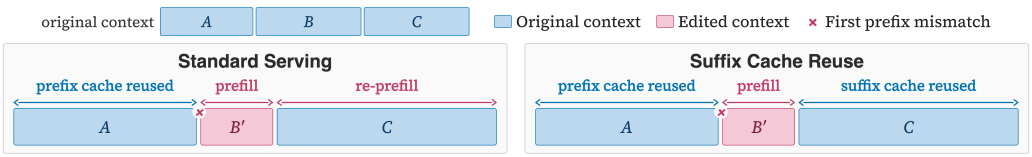


Figure 3: Comparison of standard serving and serving with Suffix Cache Reuse.

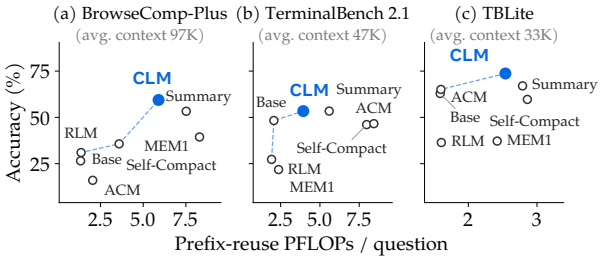


Figure 4: **Comparison of CLMs against action-based and harness-defined baselines on coding and deep research tasks.** All methods use Qwen3.6-27B with a 32K context limit and a 100-turn cap. Accuracy and inference cost; blue dashed lines indicate the Pareto frontier.

Method	Circle packing (↑)	Heilbronn (↑)	Min-max/ min-dist (↑)	Erdős overlap (↓)
OpenEvolve	2.541	0.03127	0.07690	0.38123
OpenEvolve-Agent	2.525	0.03053	0.07724	0.38167
CLM	2.618	**0.03653**	**0.07758**	**0.38094**
CLM (subagents)	**2.636**	0.03617	0.07758	0.38109

Table 1: **Comparison of CLMs against OpenEvolve on mathematical optimization problems.** Best-of-run scores with Claude 4.6 Sonnet, 32K context, and at most 100 attempts or five hours.

5 Results

5.1 Evaluating CLMs Out of the Box on Long-Horizon Agentic Tasks

We evaluate CLMs across long-horizon coding, deep research, and open discovery tasks, spanning tens to thousands of agent turns and runtimes from hours to a full day. Our evaluation covers both single- and multi-agent settings, including subagent and agent-swarm workloads.

5.1.1 Coding and Deep Research Tasks

We first evaluate CLMs on two terminal-coding benchmarks, TerminalBench 2.1 (TB2.1) (Merrill et al., 2026) and TBLite (OpenThoughts-Agent team, 2026), and on the deep-research benchmark BrowseComp-Plus (BCP) (Chen et al., 2025).[2] We compare CLMs against MEM1 (Zhou et al., 2026), Self-Compact (Li et al., 2026b), ACM (Li et al., 2026c), and recursive language models (RLM) (Zhang et al., 2025a) with a shared Mini-SWE-Agent backbone (Merrill et al., 2026). To ensure a controlled comparison independent of training data, we evaluate all methods out of the box without training. We report performance and prefix-reuse FLOPs on these benchmarks for Qwen3.6-27B with a 32K context budget, with full details in Appendix E.

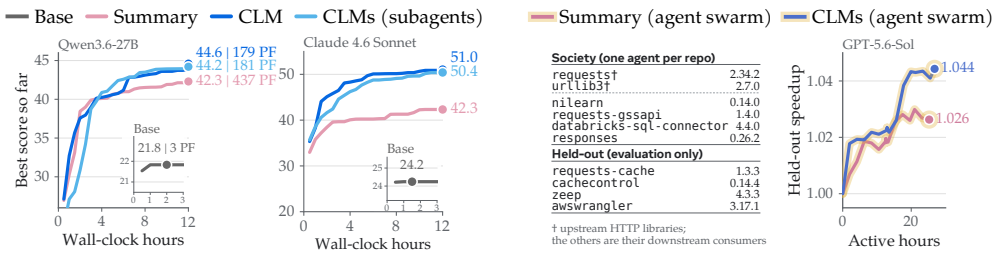
CLMs outperform harness-defined and action-based baselines. On BCP, CLMs outperform all baselines, scoring 59.4% at a 32K context limit and exceeding the strongest baseline, Codex-style summarization, by 11.4% relative. CLMs also use 21.5% and 28.9% fewer prefix-reuse FLOPs than the next two strongest methods, Codex-style summarization and MEM1, respectively. On coding benchmarks, CLMs match the strongest baseline on TB2.1 while using only 70% of its prefix-reuse FLOPs, and exceed it on TBLite (73.7% against 67.0%) with 91% of its FLOPs.

5.1.2 Open Discovery Problems

Open discovery problems provide longer horizons as our testbeds. We consider three types of open discovery problems with increasing horizons: (1) **Mathematical optimization:** four open problems popularized by AlphaEvolve (Novikov et al., 2025) and OpenEvolve (Sharma, 2025): circle packing, min-max/min-distance 2D, Erdős minimum overlap, and the Heilbronn triangle problem. (2) **Single-repository optimization:** ten EdgeBench (Zhu et al., 2026) tasks (EdgeBench-10; Appendix F.2), where the agent optimizes within a repository for up to 12 hours. (3) **Multi-repository optimization with agent swarms:** six repositories jointly optimized by multiple agents and evaluated on held-out downstream packages. Runs last over 24 hours.

Mathematical optimization: CLMs vs. specialized evolutionary workflows. On mathematical optimization, we compare against OpenEvolve (Sharma, 2025), a specialized AlphaEvolve-style (Novikov et al., 2025) workflow for program generation, evaluation, and evolutionary selec-

[2]Deep research requires search tools. Instead of training the agent against a fixed tool interface, we expose the search tools as in-context skills. This follows our less-is-more design principle: we keep as little as possible hard-coded at the harness level, so that tools can be flexibly defined, added, or revised at inference time without retraining.



(a) EdgeBench-10 single-repository optimization. (b) Software World multi-repository optimization.

Figure 5: **Long-horizon repository optimization.** (a) EdgeBench-10 single-repository optimization with a 32K context budget. Curves show best-of-three scores over 12 hours for Qwen3.6-27B and Claude 4.6 Sonnet; end labels give final scores and mean compute per trial (PF = prefix-reuse PFLOPs). (b) Software World multi-repository optimization with GPT-5.6-Sol and a 272K context budget. Six agents jointly optimize interdependent repositories and are evaluated on four unseen downstream packages; the right panel shows geometric-mean speedup over 17 evaluation tasks.

tion. We also include OpenEvolve-Agent, which replaces its proposer with a Mini-SWE-Agent that can interact with the environment before each submission. For CLM, we use the same base harness with a minimal Bash interface and provide the evolutionary algorithm as in-context guidance, leaving planning and context management to the agent. Using Claude 4.6 Sonnet and the same evaluator, CLM achieves the highest best-of-run score on all four problems (Table 1; progress curves in Figure 12). This shows that a general agent with direct context control can outperform a specialized evolutionary workflow with less fixed orchestration.

Single-repository optimization under single-agent and subagent settings. On EdgeBench-10, agents optimize a repository for up to 12 hours with verifier feedback; we report the best score over three seeds per task. Figure 5 compares the base harness, Codex-style summarization, CLM, and CLMs with up to five concurrent subagents under a 32K context budget. With Qwen3.6-27B, CLM reaches 44.6 using 179 prefix-reuse PFLOPs per trial, versus 42.3 and 437 PFLOPs for summarization; the subagent variant reaches 44.2 at 181 PFLOPs. With Claude 4.6 Sonnet, CLM and its subagent variant reach 51.0 and 50.4, compared with 42.3 for summarization. We find that subagents provide little additional benefit on this single-repository benchmark.

Multi-repository optimization with agent swarms. We evaluate CLM on `Software World`, where six agents jointly optimize interdependent Python repositories and are evaluated on four unseen downstream packages (Figure 5b, left). This provides an extrinsic test of whether improvements transfer beyond the repositories the agents directly observe. Compared with a summary-based agent swarm at equal spend, CLM delivers 65% more speedup over the unoptimized starting point (Figure 5b, right). Full setup and scoring details are provided in Appendix F.3.

5.2 Learning Context-Management Strategies in Context or in Weights

CLMs make context management an intrinsic model behavior that can be learned. In this section, we present in-context learning and RL results for CLMs using the methods described in Section 4.2.

Steering context management by simply talking to CLMs. Users can steer CLMs toward a desired context-management strategy through natural-language instructions. We demonstrate this with three behaviors: triggering compaction at a specified context length, compacting around semantic sub-question boundaries, and backing up the context before compaction. Each behavior is induced by a single sentence appended to the task prompt. As shown in Figure 6, the agent adapts its context-management policy accordingly, without any change to the harness or model parameters. Measurement details are provided in Appendix G.

Evolving context management via textual evolution with CLMs. We apply the in-context evolution loop from Section 4.2 to ContextBench with a 32K context budget. In **assisted evolution**, Qwen3.6-27B starts without any context-management instruction, with Claude Fable 5.1 serving as the skill proposer; in **self-evolution**, Opus 5 serves as both the agent and the proposer. Figure 7 shows results on KV Store from ContextBench, where both settings improve over their initialization

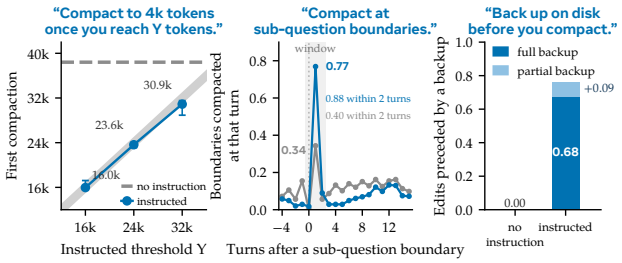


Figure 6: **One sentence in the prompt changes the context-management policy.** Natural-language instructions steer compaction timing, semantic boundaries, and backup behavior. Gray denotes no instruction and blue the instructed setting; exact prompts are in Appendix G.

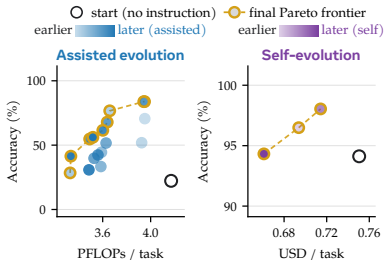


Figure 7: **Textual evolution for CLMs on KV Store (32K budget).** Assisted evolution uses Qwen3.6-27B with Claude Fable 5.1 as proposer; self-evolution uses Opus 5 for both.

Method	Acc. (%) ↑	PFLOPs / Q ↓
Summary	34.7 → 42.1	4.01 → 2.19
CLM	28.8 → **42.5**	1.52 → **1.34**

Table 2: **RL results on BrowseComp-Plus with Qwen3.5-9B.** Performance before and after training on OpenResearcher. The RL checkpoint is selected on a held-out training set.

and expand the performance–cost Pareto frontier, with evolved skills that can strictly dominate the starting point. Additional results are in Appendix C.

Reinforcement learning for CLMs. We post-train Qwen3.5-9B on OpenResearcher using the reward formulation from Section 4.2 and evaluate on held-out BrowseComp-Plus. Before training, CLM with Qwen3.5-9B underperforms the summary harness by six points due to the smaller model's limited context-management capabilities. After RL, it gains 13.7 points to 42.5%, matching the trained summary harness while using 1.34 versus 2.19 PFLOPs per question. Adding the efficiency reward further reduces inference cost without a clear loss in accuracy for either CLM or the summary harness. Full results and reward ablations are provided in Appendix D.

5.3 (More) Efficient Serving with Suffix Cache Reuse

As shown in Figure 8, SCR effectively reduces cache re-prefilling, matching the standard SGLang serving with 65.0% of its empirical prefix-reuse FLOPs on BCP. In addition, SCR is not limited to CLMs. Serving engines commonly strip prior reasoning tokens from chat histories, causing subsequent preserved tokens to be re-prefilled. We show that SCR can also reduce this re-prefilling cost in this more general setting.

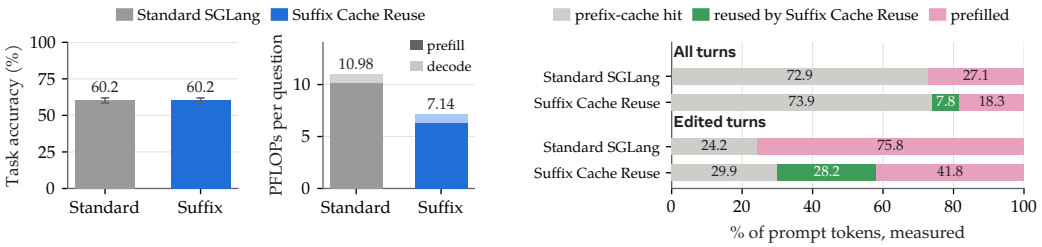


Figure 8: **Comparison of Suffix Cache Reuse and standard SGLang serving on BCP with Qwen3.6-27B.** *Left*: Task accuracy and prefix-reuse FLOPs per question. *Right*: Server-side compute decomposition, showing the fraction of prompt tokens by compute type across all turns and turns following context edits.

(a) CLM builds in-context scoreboards and trackers to orchestrate and monitor subagents with in-place editing.

```
open(p,"w").write("""[[CTX_TURN 1 role=assistant]]
## STATE -- Erdős Minimum Overlap Problem (compact)
LEDGER TOP: 0.9992491 ...
AGENTS: 21 launched, 5 currently running ...
KEY FILES: /workspace/subctx_4/h_best_final.npy ...
FINDINGS: All methods plateau at 0.381157 ...""")
```

Erdős minimum overlap, step 455

```
new="""## ORCHESTRATOR STATE (compact)
Budget: 7/100 used. All 5 slots BUSY (subctx_0..4 ...
Dead ends: simple grid(0.822), hexagonal(0.9977) ...
Next: score my own candidates while workers run ..."""
```

Circle packing $N = 26$, steps 194, 198

(c) CLM writes for loops to remove past irrelevant search results or compact overly long outputs when creating a new view.

```
for t in turns[1:]: ...
    result += f'\n[[CTX_TURN search]]\nSearched: {m.group(1).
↪ strip()}. No relevant results.\n'
```

BrowseComp-Plus, step 1509

```
while i < len(lines): ...
    if len(body) > 500: ...
        if 'bcp_search' in body: ...
            result.append(f"[Searched: {query}]")
        elif 'bcp_get_document' in body: ...
            result.append(f"[Retrieved doc {docid}]")
    else:
        result.extend(body_lines)
```

BrowseComp-Plus, step 13

(b) CLM creates a new role alongside the original template roles for its own internal notes.

```
re.sub(r"\[\[CTX_TURN 4 .*?(?=\[\[CTX_TURN 16)",
  """[[CTX_TURN 4 role=notes]]
STATUS: ... James Gallagher (docid=58939) ...""")
```

BrowseComp-Plus, step 40

(d) CLM defines and reuses a function to conveniently compact old results with a reference to its maintained note.

```
progress = """[Search progress: VERIFIED ...]"""
s = re.sub(..., progress, s)

def compact_turns(text):
    return re.sub(..., lambda m: ...split('\n')[0]
        + '\n[search results - see progress note]', text)
s = compact_turns(s)
```

BrowseComp-Plus, step 109

(e) CLM reproduces effective behaviors from existing baselines, preserving important facts and future TODOs in the summary.

```
re.sub(r"\[\[CTX_TURN 2.*",
  "[SUMMARY: ... Kader Asmal Excellence Award
   launched 2011 by Mrs A Motshekga ...]")
```

BrowseComp-Plus, step 20

```
new="""[EXPLORATION LEDGER - 86 scored attempts ...
Best score: 0.9931 (sum_radii=2.6177) from ...
UNTRIED IDEAS (priority order):
1. Gradient clipping norm=1.0 with 12k steps
2. Try lam=2200+uniform(0,2800) with 12k steps ..."""
```

Circle packing $N = 26$, step 332

Figure 9: **Qualitative examples of CLM context-management behaviors**.

5.4 Qualitative Examples

We show qualitative examples of CLMs in Figure 9, revealing both novel context-management behaviors and effective compaction strategies. For multi-agent orchestration, CLM maintains an in-context scoreboard and updates agent status through 163 in-place edits while keeping the context at only 6–8K tokens (a). It can create new internal roles such as "notes" when rewriting its context (b), and use loops to remove irrelevant search results or compact overlong observations (c). CLM can also define and reuse helper functions: in (d), it invokes 'compact_turns' 37 times to maintain a progress note while compacting detailed observations. Finally, it reproduces effective compaction behaviors by compressing 21K tokens into answer-relevant summaries or preserving untried ideas for future explorations (e).

6 Discussion and Future Work

Safety implications of a model-editable context. Granting models write access to their live context enables more flexible context management, but it also creates new safety challenges. In particular, prompt injection may become more consequential when injected content can influence how the model rewrites its own working memory, including whether important constraints are preserved and how past interactions are summarized. As editable context becomes more widely used, future work should study these new attack surfaces and develop mechanisms that preserve the flexibility of model-controlled context while maintaining its integrity.

Future directions: scaling CLM RL and distilling existing harnesses into CLMs. Future work can scale RL training to help CLMs discover stronger context-management strategies, and explore harness-to-CLM distillation, in which strategies from existing harnesses are transferred into CLMs. This is motivated by the view that standard LMs only map input tokens to next-token distributions, while harnesses determine how the context is constructed and updated. Since many harness operations can be expressed as context transformations, they can potentially be translated into CLM actions and eventually internalized into model weights. From this perspective, harnesses act as a form of procedural memory or task-specific skill that can be developed externally and later absorbed by CLMs for more general use.

AI USE STATEMENT

We used generative AI tools to polish author-written text, assist with parts of the experimental and analysis code, and help produce figures from our results. All AI-assisted work was reviewed by the authors, who take full responsibility for the final text, claims, code, and figures.

ETHICS STATEMENT

This work involves no human subjects or personal data. ContextBench is synthetic, and all other benchmarks are public. The capability we study also raises safety concerns. An agent that can rewrite its own context may remove constraints, discard records of past actions, or be influenced by injected content to alter its working memory, making its behavior harder to audit. In our experiments, every context edit is issued through an explicit command and logged together with the context before and after the edit, which allows us to inspect behaviors such as those in Figure 9. We believe deployed systems should preserve similar audit trails. We discuss these risks further in the Discussion and leave defenses that preserve context flexibility without compromising integrity to future work.

REPRODUCIBILITY STATEMENT

We will release our code, ContextBench, and the configurations used for the main experiments. ContextBench is described in Appendix B; evaluation settings, budgets, seeds, and graders for the coding and deep-research benchmarks are in Appendix E; the open-discovery setups are in Appendix F; steering, skill-evolution, and RL details are in Appendices G, C, and D. Several results use proprietary API models whose versions may change or be retired; we therefore report the model versions and budgets used.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026.

Federico Cassano and Sasha Rush. Training Composer for longer horizons. Cursor Research Blog, March 2026. URL `https://cursor.com/blog/self-summarization`. Published March 17, 2026; accessed July 29, 2026.

Aaron Chan, Ahmed Shalaby, Alexander Wettig, Aman Sanger, Andrew Zhai, Anurag Ajay, Ashvin Nair, Charlie Snell, Chen Lu, Chen Shen, et al. Composer 2 technical report. *arXiv e-prints*, pp. arXiv–2603, 2026.

Zhi Chen, Zhensu Sun, Yuling Shi, David Lo, and Lingxiao Jiang. Are performance-optimization benchmarks reliably measuring coding agents? *arXiv preprint arXiv:2607.01211*, 2026.

Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, et al. Browsecomp-plus: A more fair and transparent evaluation benchmark of deep-research agent. *arXiv preprint arXiv:2508.06600*, 2025.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. In *Proceedings of Machine Learning and Systems*, 2024.

Junhao Hu, Wenrui Huang, Weidong Wang, Haoyi Wang, Tiancheng Hu, Qin Zhang, Hao Feng, Xusheng Chen, Yizhou Shan, and Tao Xie. EPIC: Efficient position-independent caching for serving large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, pp. 24391–24402, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model

serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

Mo Li, LH Xu, Qitai Tan, Long Ma, Hongyong Song, Ting Cao, and Yunxin Liu. Sculptor: Empowering llms with cognitive agency via active context management. In *International Conference on Learning Representations*, volume 2026, pp. 153411–153440, 2026a.

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

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

Yujiang Li, Zhenyu Hou, Yi Jing, Jie Tang, and Yuxiao Dong. CompactionRL: Reinforcement learning with context compaction for long-horizon agents. *arXiv preprint arXiv:2607.05378*, 2026d.

Shukai Liu, Bo Jiang, Jian Yang, Yizhi Li, Jinyang Guo, Xianglong Liu, and Bryan Dai. Context as a tool: Context management for long-horizon swe-agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 20604–20617, 2026.

Mike A Merrill, Alexander G Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *arXiv preprint arXiv:2601.11868*, 2026.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

OpenAI. Codex CLI: A coding agent for the terminal. `https://github.com/openai/codex`, 2026. Software, version 0.146.0, accessed July 29, 2026.

Bespoke Labs OpenThoughts-Agent team, Snorkel AI. OpenThoughts-TBLite: A High-Signal Benchmark for Iterating on Terminal Agents. https://www.openthoughts.ai/blog/openthoughts-tblite, February 2026.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Asankhaya Sharma. Openevolve: an open-source evolutionary coding agent, 2025. URL `https://github.com/algorithmicsuperintelligence/openevolve`.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. Scaling long-horizon llm agent via context-folding. *arXiv preprint arXiv:2510.11967*, 2025.

Richard S. Sutton. The bitter lesson. `http://www.incompleteideas.net/IncIdeas/BitterLesson.html`, 2019.

Shengguang Wu, Hao Zhu, Yuhui Zhang, Xiaohan Wang, and Serena Yeung-Levy. Automem: Automated learning of memory as a cognitive skill. *arXiv preprint arXiv:2607.01224*, 2026.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z Pan, et al. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12805–12825, 2026.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2405.15793.

Xinyu Yang, Tianqi Chen, and Beidi Chen. APE: Faster and longer context-augmented generation via adaptive parallel encoding. In *International Conference on Learning Representations*, 2025.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. CacheBlend: Fast large language model serving for RAG with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 94–109, 2025.

Yi Yu, Liuyi Yao, Yuexiang Xie, Qingquan Tan, Jiaqi Feng, Yaliang Li, and Libing Wu. Agentic memory: Learning unified long-term and short-term memory management for large language model agents. *arXiv preprint arXiv:2601.01885*, 2026.

Alex L Zhang, Tim Kraska, and Omar Khattab. Recursive language models. *arXiv preprint arXiv:2512.24601*, 2025a.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin godel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*, 2025b.

Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. Hyperagents. *arXiv preprint arXiv:2603.19461*, 2026a.

Xuan Zhang, Longtao Zheng, Cunxiao Du, Bo An, and Xin Dong. Autocompact: Learning when to compact context in long-horizon coding agents, 2026b. URL https://autocompact.github.io/.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, 2024.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. MEM1: Learning to synergize memory and reasoning for efficient long-horizon agents. In *International Conference on Learning Representations*, volume 2026, pp. 58413–58438, 2026.

Deyao Zhu, Xin Zhou, Shengling Qin, Xuekai Zhu, Hangliang Ding, Shu Zhong, Zixin Wen, Zhonglin Xie, Chenhui Gou, Linxuan Ren, et al. Edgebench: Unveiling scaling laws of learning from real-world environments. *arXiv preprint arXiv:2607.05155*, 2026.

A Extended Related Work

Rule-based context management. Most existing harnesses fall into this category, including Cursor (Cassano & Rush, 2026), Codex (OpenAI, 2026), and Terminus2 (Merrill et al., 2026). These systems trigger context compaction when the context length exceeds a predefined threshold, typically using a human-written summarization prompt or a fixed static pipeline. In this setting, the model does not decide when context management occurs: the harness triggers compaction according to a fixed rule, although the model may generate the compacted representation itself. MEM1 (Zhou et al., 2026) replaces the length trigger with a per-turn update: at every turn the model rewrites a single internal state from the previous state and the new observation, and the raw history is discarded. MEM1 trains the contents of this update with reinforcement learning, but its per-turn update schedule and state-transition structure are fixed by the harness.

Model control within a constrained action space. This line of work gives the model limited control over context management by prompting it to choose from a predefined set of context-related actions, such as LLM-summary compaction, storage offloading, and retrieval (Yan et al., 2026; Yu et al., 2026; Li et al., 2026c). Context-as-a-Tool (Liu et al., 2026) exposes model-triggered compaction within a predefined structured context workspace. Self-Compact (Li et al., 2026b) lets the model decide when to compact its context; Context Folding (Sun et al., 2025) lets it branch into a sub-trajectory for a subtask and fold that sub-trajectory into a concise summary when it returns; and Sculptor (Li et al., 2026a) provides tools that let the model select context fragments to operate on. However, these approaches still restrict the model to a human-defined action space: they expose particular operations for compaction, offloading, retrieval, or branching rather than a general primitive for arbitrary context transformations.

Model-controlled context through scaffold and agent optimization. Closely related work explores outer-loop optimization of agent harnesses, memory scaffolds, or the agent program itself (Lee et al., 2026; Wu et al., 2026; Zhang et al., 2025b; 2026a). In these approaches, an agent starts from an initial harness and improves it using past trajectories and evaluation feedback, usually through an evolutionary self-modification loop that proposes, evaluates, and selects new agent variants. This offers another route toward model-controlled context policies: the agent can improve the harness that governs future executions. However, this control is indirect. Once the optimized harness is deployed, its policy remains fixed within each new rollout. Related evolutionary methods search over programs or prompts rather than harnesses. AlphaEvolve (Novikov et al., 2025) and its open-source counterpart OpenEvolve (Sharma, 2025) evolve programs through a fixed proposer–evaluator workflow, and GEPA (Agrawal et al., 2026) evolves prompts by reflecting on rollouts.

Context as a variable in the environment. Recursive language models (RLMs) (Zhang et al., 2025a) are the closest in spirit to CLM: both treat context as a variable the agent's actions can touch. An RLM keeps two variables. The long input is loaded into a persistent REPL as `state`, which the model's code can read, transform, and recurse over; the model's own conversation is `hist`, which only grows and is never mutated. CLM collapses the two: the variable passed to the action step is the live context itself—what RLMs call `hist`—and the action may rewrite it, not only read it. RLMs thus let the model manipulate an external context object, while its live generation history remains append-only; CLMs instead make that live context itself writable.

Non-prefix KV cache reuse. Serving systems such as vLLM (Kwon et al., 2023) and SGLang (Zheng et al., 2024) reuse cached KV states only for exactly matching prefixes. Several works relax this restriction. Prompt Cache (Gim et al., 2024) precomputes the attention states of reusable prompt modules and reuses them across prompts. CacheBlend (Yao et al., 2025) and EPIC (Hu et al., 2025) reuse the KV caches of independently encoded text chunks regardless of the preceding prefix, recomputing a small subset of tokens to restore cross-chunk attention, and APE (Yang et al., 2025) encodes contexts independently in parallel and adjusts the attention distribution to better approximate sequential encoding. In streaming generation, StreamingLLM (Xiao et al., 2024) keeps the KV states of a few initial attention-sink tokens and a sliding window of recent tokens, so cached states remain in use after the middle of the context is evicted. These methods target retrieval-augmented generation, in-context learning, and streaming inputs, where the reused content is fixed and assembled by the system. Suffix Cache Reuse addresses a different source of

cache misses: in-place edits that the agent makes to its own context, after which the unchanged suffix is reused instead of re-prefilled.

Reinforcement learning for context management. Several recent works use reinforcement learning to improve context-management behavior over long agent trajectories. Composer (Cassano & Rush, 2026; Chan et al., 2026) and CompactionRL (Li et al., 2026d) train models within harnesses that apply length-triggered, LLM-summary-based compaction. AutoCompact (Zhang et al., 2026b) further gives the model control over when to compact, rather than relying on a fixed length trigger.

B CONTEXTBENCH

ContextBench consists of four synthetic tasks. In every task, content is streamed to the agent one chunk at a time through a forward-only cursor, and the total volume exceeds the context budget.

Tasks. **Needle Retention**: each chunk contains a few needles that must be retained verbatim and filler text that can be discarded; the agent succeeds if the needles are exactly recoverable from its context at the end of the run. **Sudoku Sketchpad**: the agent maintains a Sudoku board rendered as a pencil-mark panel and receives one fully specified move per turn, re-printing the board after each move. **KV Store**: the agent receives a stream of SET and GET operations over random values and must report the stored value for each GET. **Log Triage**: a service log is streamed in batches, after which the agent answers incident questions that require the exact message of a log line or an exact count of matching lines.

B.1 SETUP

The results in Figure 2 use gpt-5.4 as the agent, a 32,768-token context budget with a 2,048-token reserve, and four to eight seeds per pressure level; a seed regenerates the task instance. Task messages are pushed to the agent; the three streaming tasks are delivered in 4K-token chunks. The observation cap is 50,000 characters for every method, and budget reminders and plugins are off. The reward is the fraction of queries answered correctly (Needle Retention, KV Store, Log Triage) or of gold board cells reproduced (Sudoku Sketchpad, best per-cell accuracy over the panels the agent displays), averaged over seeds.

Context pressure. The horizontal axis of Figure 2 is the total input the environment pushes to the agent over the episode, divided by the 32,768-token context limit. Each task reaches higher pressure through one generator knob (Table 3): Needle Retention pushes more document chunks, KV Store writes more records before the same 24 queries, Log Triage pushes more log lines before the same 24 queries, and Sudoku Sketchpad plays more moves, with the board counted once per move.

Baselines. The summary harness compacts at 75% of the budget. ACM (Li et al., 2026c) uses the offload and retrieve tools of its released implementation unchanged, with an instruction of comparable length to the CLM method description. RLM (Zhang et al., 2025a) is the released implementation ported unchanged: the task input is a REPL variable to which each streamed turn is appended. Self-Compact (Li et al., 2026b) uses the released prompts unchanged, with its two compaction thresholds rescaled to the 32K budget (28,160 and 10,620 tokens). Context Folding is a port of FoldAgent (Sun et al., 2025) in which the model calls the fold and branch tools itself.

C SELF-EVOLUTION: FULL RESULTS

C.1 SELECTION PROTOCOL

Each proposed skill is screened on six training tasks with one seed; skills that pass are evaluated on the development split (34 tasks, each at three pressure levels; 102 instances). Accuracy is the mean reward; cost is prefix-reuse PFLOPs per task for the Qwen agent and API cost in USD per task for the API agents. A skill enters the Pareto frontier if no previously evaluated skill, including the starting point, is at least as accurate and at least as cheap. The held-out test split is used only for the final evaluation. In each round, the proposer reads rollouts of the current skill, including

Family	Knob varied	Levels (knob value → total input)	Pressure
Needle Retention	document chunks pushed (K needles, C chunks of about 4K tokens)	4→17K, 8→33K, 24→97K, 48→198K, 60→247K, 96→384K, 192→756K, 224→881K	0.52×–27×
Sudoku Sketchpad	moves; the board is re-rendered once per move	4→11K, 8→17K, 15→29K, 30→55K, 60→106K, 100→175K, 150→263K	0.33×–8×
KV Store	records written in batches of 100, then 24 queries	130→8.8K, 260→15K, 520→28K, 1000→51K, 2000→100K, 4000→198K, 8000→393K, 16000→784K	0.27×–24×
Log Triage	log lines pushed in batches, then 24 queries	160→9.8K, 320→17K, 640→31K, 1600→72K, 3200→135K, 12800→517K	0.3×–16×

Table 3: **How context pressure is produced in ContextBench.** For each family one generator knob is varied; total input is every message the environment pushes over the episode (task instruction included; for Sudoku the board is counted once per move, as the protocol requires it to be re-printed), counted with the o200k tokenizer. Context pressure is total input divided by the 32,768-token context limit. The knob values are the levels drawn in Figure 2.

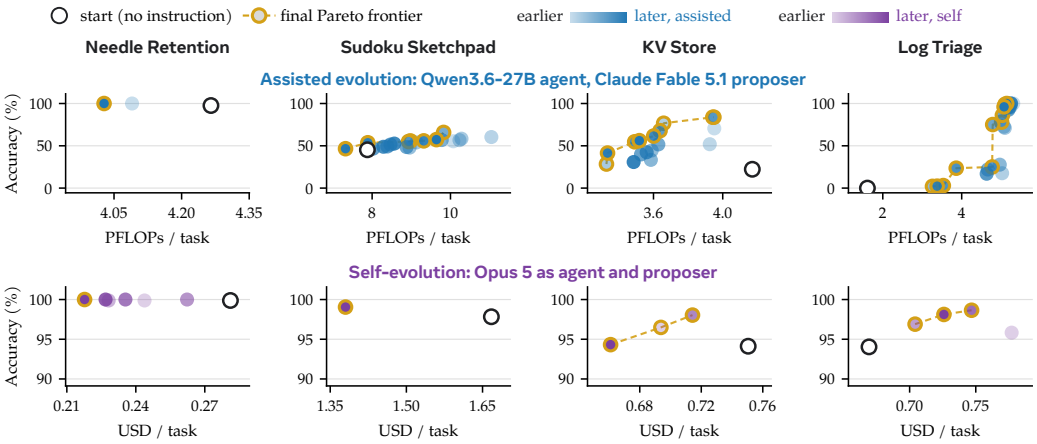


Figure 10: **Evolving context-management skills on ContextBench (32K budget), all four tasks.** **Assisted evolution** (top): Qwen3.6-27B serves as the agent, while Claude Fable 5.1 proposes the skills; the x-axis shows prefix-reuse PFLOPs per task over solved runs. **Self-evolution** (bottom): Opus 5 serves as both the agent and the skill proposer; the x-axis shows API cost per task in USD. Skills are proposed based on training-split rollouts and never use the held-out evaluation set.

successful and failed rollouts on the same instance, and proposes at least four rewrites. Only the skill text changes; tasks, seeds, grader, harness, tools, and all other prompts are fixed.

D Reinforcement Learning Results

Figure 11 shows the reward ablation for Section 5.2. The RL checkpoint in Table 2 is selected on 500 held-out OpenResearcher questions disjoint from the training prompts and from BrowseComp-Plus.

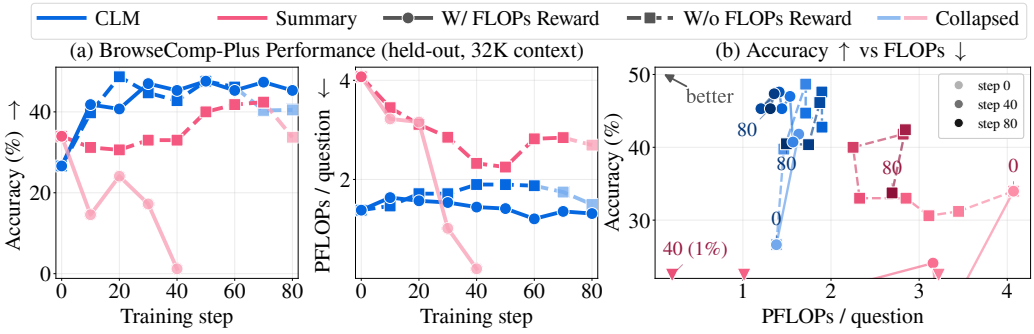


Figure 11: Ablation of reward design using Qwen3.5-9B with a 32K peak context limit.

E Details of Harness Designs and Evaluation

Benchmarks. TerminalBench 2.1 and TBLite are terminal-coding benchmarks scored by their own test harnesses. BrowseComp-Plus is a deep-research benchmark over a fixed document corpus; the agent searches and reads through skill-provided tools and submits a final answer.

Budgets. Qwen3.6-27B and Qwen3.5-9B run at a 32K context budget on a 32,768-token serving window, counted over the full message list at every turn. On BrowseComp-Plus the agent's budget is the window minus the 4,096-token generation cap (28,672 tokens). Step limits are shared by all methods within a benchmark: 100 steps on BCP, 80 on TB2.1, and 64 on TBLite.

Methods. All methods except MEM1 and RLM use the Mini-SWE-Agent harness with the same tools and prompts. The summary harness compacts at 75% of the budget and ACM at 60%. MEM1 runs its released driver with the same base model as the other methods: at every turn the model rewrites a single state block and the raw history is discarded. RLM runs its released harness: the task input lives in a Python REPL, the root model can call the model recursively on slices of it, and sub-calls count toward cost. For Claude models, the CLM harness delivers the edited context as a user turn rather than an assistant prefill.

Seeds, grading, and cost. Open models run three seeds where a standard deviation is shown. A run that ends without an answer is scored as wrong. Cost is prefix-reuse PFLOPs per question (Eq. 3).

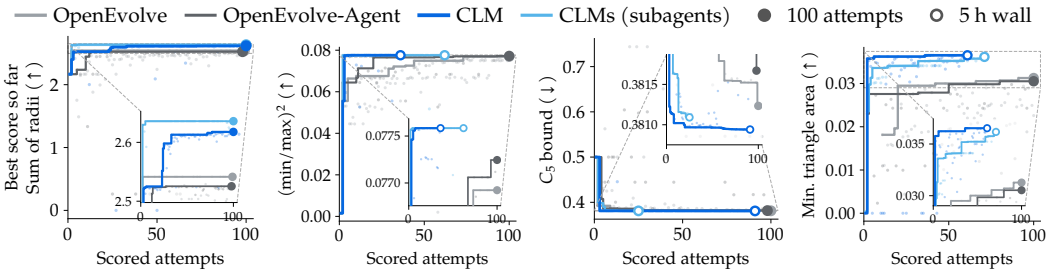
E.1 BrowseComp-Plus Grading

We follow the grading setup of ACM (Li et al., 2026c): the simple-evals BrowseComp grader template with Qwen3.5-27B as the judge at temperature 0, which marks an answer correct only if it is correct and complete against the gold entity (830 questions). The only change from ACM is the judge model (ACM uses GPT-5).

F Open Discovery Problems: Setup and Measurement

F.1 Mathematical Optimization Problems

Tasks. We use four problems popularized by AlphaEvolve and OpenEvolve, each with a fixed evaluator that scores a submitted program: circle packing (sum of radii of 26 circles in the unit



(a) Circle packing (↑). (b) Min-max/min-dist (↑). (c) Erdős min-overlap (↓). (d) Heilbronn triangle (↑).

Figure 12: **Best-so-far score versus evaluator-scored attempts on four open optimization problems.** All runs use Claude 4.6 Sonnet with a 32K context limit and stop after 100 attempts or five hours. Lines show the best score so far, dots individual scored candidates, and insets the final-score range.

square, higher is better), Erdős minimum overlap (the C_5 upper bound, lower is better), min-max/min-distance 2D (the ratio $(\min / \max)^2$ of pairwise distances, higher is better), and the Heilbronn triangle problem (normalized minimum triangle area, higher is better).

Methods. OpenEvolve runs MAP-Elites evolution with fixed proposer prompts, program database, parent sampling, island migration, and evaluation cascade. OpenEvolve-Agent replaces its proposer with a Mini-SWE-Agent that can interact with the environment before each submission. CLM uses the Mini-SWE-Agent harness with editable context and a short on-disk description of the evolutionary-search workflow; CLM (subagents) can additionally spawn subagents.

Budget. All methods use Claude 4.6 Sonnet and the same evaluator. A run stops at 100 evaluator-scored attempts or five hours, whichever comes first. Following AlphaEvolve and OpenEvolve, we report the best-of-run score, the standard metric for open optimization problems.

F.2 Single-Repository Optimization Problems

Tasks. EdgeBench-10 is a fixed subset of ten EdgeBench tasks (Table 4) that covers its task families and range of context demand. Each task is a repository with a benchmark suite and a verifier; the agent may submit the repository state at any time and receives its verifier score.

EdgeBench-10 task (benchmark id)	Category	Language	What the verifier scores	Start: files / LOC
`ad_placement_optimization`	Combinatorial optimization	C++	solution score over judge cases	1 / 20
`apple_incremental_game`	Combinatorial optimization	Python	solution score over judge cases	3 / 101
`graph_node_classification`	Science & ML	Python	held-out accuracy (CPU-only judge)	2 / 418
`grid_turing_robot`	Combinatorial optimization	Python	solution score (lower is better)	3 / 440
`juliet_vulnerability_analyzer`	Software engineering	Python	hidden evaluator on the Juliet suite	1 / 9
`openrct2_theme_park_ai`	Games & simulators	JavaScript plugin	park value in a headless OpenRCT2 run	6 / 5,950
`schemathesis_datagen_pipeline`	Software engineering	Python	fraction of the test suite passing	416 / 112,663
`triangulation_coloring_optimization`	Combinatorial optimization	Python	coloring "ugliness" (lower is better)	5 / 431
`vehicle_routing_time_windows`	Combinatorial optimization	C/C++	CVRPTW solution quality	3 / 513
`wesnoth_tactical_ai`	Games & simulators	Python	headless Wesnoth matches	0 / 0

Table 4: The ten EdgeBench-10 tasks. Scores are rescaled by the official EdgeBench judge to 0–100. "Start" gives the files and lines of code in the working directory before any edit.

Setup. The agents are Qwen3.6-27B and Claude 4.6 Sonnet with a 32K context budget. Each trial runs for twelve hours of wall-clock time with no turn limit, and each method is run on the ten tasks with three seeds each. We compare the base harness, Codex-style summarization, and CLM alone and with up to five parallel subagents.

Measurement. A trial's score is its best verifier score over all submissions, with the state at the horizon counted as a final submission (zero if none). The curves in Figure 5 show this score over time, and, since each twelve-hour trial is an independent search for the best repository state, each task's score is the best over three independent trials.

F.3 Multi-Repository Optimization Problems

Setup. Six agents each maintain one repository on a shared code forge (Figure 5b, left): the HTTP libraries `requests` and `urllib3` and four of their downstream consumers. Agents can read the other repositories, open issues and pull requests, and cut releases, with no task-specific instructions. Every agent uses GPT-5.6-Sol with a 272K context budget; runs last more than 24 hours.

Evaluation. We install the latest agent-optimized versions of the two upstream libraries and run 17 CPU benchmarks from held-out downstream packages that the agents never see, measuring deterministic instruction counts (Chen et al., 2026). The score is the geometric mean speedup over the initial version. A change that breaks a held-out package's tests is discarded and yields no speedup for that package.

Baseline. The baseline uses the same agents, base model, and context budget; only the context-management strategy differs. Each agent runs in short sessions that start from an empty context, reload a persistent memory file that the agent updates before each session ends, and use the harness's built-in summarization within a session.

G Steering Experiments

All steering experiments use Claude 4.6 Sonnet running CLM on BrowseComp-Plus, with no retries and periodic budget reminders off; the agent sees its context size only through a one-line readout after each tool result. Each condition appends one sentence, given verbatim below, to the task message and is compared with a control on the same tasks.

G.1 (a) Compaction Threshold

> Monitor your context size: every tool result reports your current size. Whenever it exceeds Y tokens, immediately compact your context down to about X tokens. For example: rewrite /tmp/.live_ctx/LIVE_CTX_MAIN.txt to keep only your key findings and current plan (about X tokens).

We vary Y with a 48K budget and $X{=}4$K on 30 tasks per condition, and measure the context size at the first compaction.

G.2 (b) Semantic Boundaries

> Please only compress at the boundaries of subtasks. Once a subtask is accomplished, throw out all the context for that subtask. For example: after you write /workspace/answer_1.txt, rewrite /tmp/.live_ctx/LIVE_CTX_MAIN.txt to drop the Question 1 material, keeping only your answer_1 conclusion.

Each session chains four questions, and writing an answer file marks a sub-question boundary. We report the share of boundaries followed by a compaction at each turn offset over 189 sessions per condition.

G.3 (C) LOSSLESS BACKUP

> Before every edit to your context, you must back up the current context file into a new file inside a new folder named compaction_backup; never overwrite or delete earlier backups.

On 200 tasks we record, for every context edit, whether the agent first saved a full or partial copy of the context file.