

Context Language Models (CLMs)

Language models that manage their own context

Consolidated version · October 2026

Spans, traces and context management by AI agents

Study roadmap

1

Introduction and method

2

The problem: context as a scarce resource

3

Taxonomy of approaches

4

The proposal: Context Language Models

5

Goals and problems addressed

6

Reported empirical evidence

7

Critical analysis: pros, cons, feasibility

8

Security, integrity and governance

9

Is the question new? Predecessors

10

Open questions and research agenda

11

Conclusion and final considerations

G·R

Glossary and references

Summary

● The proposal

Shao et al. (2026; University of Washington, Meta Superintelligence Labs, MIT and Trillium Labs): the model natively manages its own context, treating it as a file it can rewrite without restrictions, instead of relying on an external harness for summarization, compaction or retrieval.

● What the study does

Formalizes the proposal; systematizes goals, evidence, pros and cons, security risks and feasibility; and places the idea in its lineage: psychology of memory, computer architecture, networks with external memory, RAG, MemGPT, Recursive Language Models and Memento.

● Conclusions

The formulation is new in its generality and cost accounting, but inherits more than six decades of lineage. The evidence is promising, yet comes from a preprint (v1) without peer review and from specific benchmarks.

● Final balance

Feasibility in production depends on integrity controls over a context the model itself can edit. In the current state, structural costs outweigh the benefits for production use.

Keywords: context management; AI agents; language models; context engineering; KV cache; reinforcement learning; agent security.

1 Introduction and method

● Object of study

Moving context management from the code that orchestrates a language model (the harness) into the model itself.

● Sources and selection criterion

Primary source: the preprint by Shao et al. (2026), arXiv:2609.37725v1, with the official repository and the OpenAI report (2026b). News and third-party analyses were excluded; only what was verified in the preprint, the repository, official provider documentation or the bibliography of the cited works is kept.

● Goals

- Locate the original sources and tell primary from secondary sources.
- Identify the central ideas, concepts, goals and problems the solution aims to solve.
- Assess evidence, pros, cons, risks and feasibility.
- Answer whether the question is new and who its predecessors are.
- Produce a glossary and bibliography.

● Conventions

Author-date citations. “Inference of this study” marks analysis or calculation by this author. Numbers from Shao et al. (2026) were checked against the preprint’s text and tables.

2 Three pressures on the context

● Physical limit

The context window is finite. In a standard model the context grows only by concatenation: at each step the output is appended to what already existed ($c(t+1) = c(t) \oplus \text{output}$).

● Quality degradation

Liu et al. (2023): performance tends to be higher when the relevant information is at the start or end of the context and worse in the middle. Zhang, Kraska and Khattab (2025) describe “context rot”, attributed to Hong et al. (2025): quality decays as the context grows.

● Computational cost

Every token must be processed (prefill phase) and held in the KV cache. More context means more computation and more memory per step.

The usual solutions are external to the model: summarizing, compacting by rules, offloading to files and retrieving on demand. According to Shao et al. (2026, section 1), summaries can lose details or introduce errors; compaction follows fixed rules set by whoever builds the agent; and external memory requires the agent to decide what to bring back.

2

Commercial solutions and failure modes

● Anthropic (official docs)

- Context editing: automatically removes old tool calls and results near the limit.
- Memory tool: the model stores and consults information outside the window, via files, on the client side.
- Server-side compaction: indicated as the main strategy in long workflows.

● OpenAI (official docs)

- The compaction API returns a compacted window with an encrypted item.
- The item is described as opaque and not meant to be human-interpretable.
- The policy of when to compact is set by the platform or the developer.

● Failure modes (ACE, 2025)

- Brevity bias: drops domain insights in favor of short summaries.
- Context collapse: iterative rewriting erodes details over time.

2 Limitations of previous strategies

Previous strategy	Limitation noted
Summarization	May lose or hallucinate information (ContextBench: Needle Retention and Sudoku Sketchpad)
Offloading to external memory	The agent must decide what to bring back; standard tools offload but cannot evict content from the live context on demand (KV Store and Log Triage)
On-demand retrieval and access (RLM)	Retrieved information stays appended to the live context, which keeps growing
Predefined actions in the harness	Model autonomy is limited to the human-defined action space

4 tasks

ContextBench: Needle Retention, Sudoku Sketchpad, KV Store and Log Triage

32K · 24×

Context limit and maximum context pressure (input volume ÷ limit)

None perfect

No existing method performed perfectly even on these simple tasks (Shao et al., 2026, section 3)

3 Taxonomy of context-management approaches

Category	Who decides	Examples cited by the authors	Limitation noted
Harness-scheduled	The agent code (by size threshold or every turn)	Cursor/Composer, Codex CLI, Terminus2, MEM1	Schedule and procedure are prescribed
Model with constrained actions	The model, within a human-defined set of tools	Self-Compact, AutoCompact, Context-as-a-Tool, ACM, AgeMem, Context Folding, AgentFold, Sculptor	Autonomy limited to the predefined action space
Meta-optimization	An outer loop that improves the harness or instructions	Meta-Harness, AutoMem, Meta Context Engineering, GEPA, Darwin Gödel Machine, Hyperagents	Optimizes reusable procedures, not the live context
Context as an environment variable	The model, reading the input through code	Recursive Language Models (RLM)	Read access to the live context; accumulated history keeps growing
External memory and hierarchy	The model, with paging functions	MemGPT, Memory-R1, AgeMem	The model edits a designated fixed-size block, not the whole context
Context as an editable file	The model, with no restriction on actions	CLM	New integrity and security risks

3 Three levels of control over the context

Level 1 · Harness rules

The agent's code decides when and how the context is reduced (size threshold or every turn).

Level 2 · Fixed operations

The model triggers predefined operations: compact, offload, retrieve, select fragments.

Level 3 · Unrestricted editing (CLM)

The model edits any part of the context at any step and defines its own management functions.

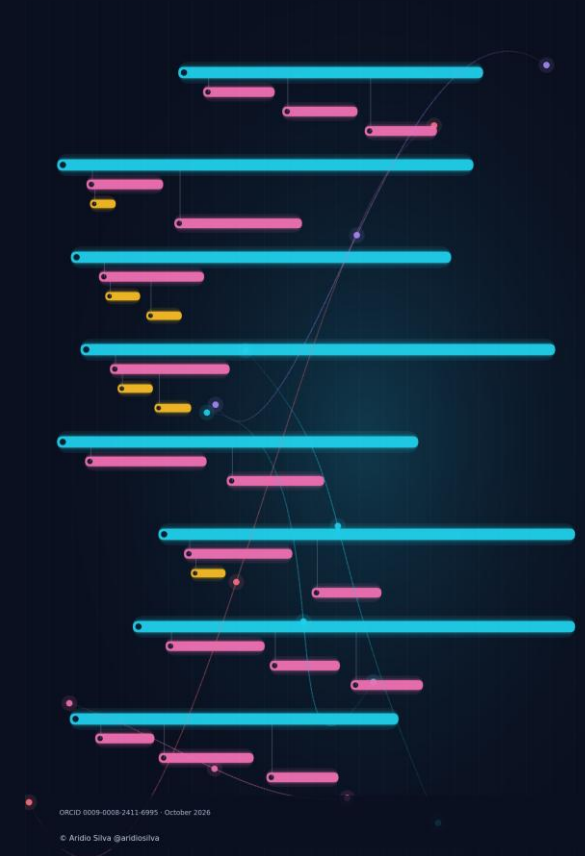
Didactic reading (inference of this study, consistent with Shao et al., 2026, section 2). Recursive Language Models stand apart: they externalize the input, not the history, and access to the live context is read-only; RLM and CLM are complementary.

4 The proposal: formal definition

Standard model (append-only) $\mathbf{c}(t+1) = \mathbf{c}(t) \oplus \mathbf{f_LM}(\mathbf{c}(t))$ (Eq. 1)

Context Language Model $\mathbf{c}(t+1) = \mathbf{f_CLM}(\mathbf{c}(t))$ (Eq. 2)

- $\mathbf{c}(t)$ is the context at turn t ; $\oplus$ is concatenation; $\mathbf{f_CLM}$ can be an arbitrary function controlled by the model.
- **It subsumes what came before:** any predefined compaction, discard or retrieval tool is a particular case of a transition function.
- **The central difference:** in CLMs the model also defines those functions, implicitly through its planning or explicitly as reusable functions.



4

Implementation: context as a file

Component	Function (didactic synthesis based on Shao et al., 2026, sections 4.1 and 4.3, and the official repository)
Context file	Editable mirror of the live context; its path is given in the system prompt
Editing interface	Ordinary Bash commands; with no edit, generated tokens are appended at the end
Synchronization	Every change is automatically synchronized with the context and sent to the inference server
Skill document	Natural-language instruction, provided in context and evolvable through an optimization loop
Suffix Cache Reuse (SCR)	SGLang patch that reuses cached states of tokens that survived the edit
Multi-agent	One context file per agent; subagents created or ended by creating or deleting files

Didactic example: a research agent, after 40 steps, swaps dozens of search results for a short note, keeps verbatim the passage that proves the main fact and lists hypotheses not yet tested, like a researcher rewriting their own notebook.

The CLM is not a new transformer architecture: it was applied, without specific training, to existing models (Qwen3.6-27B and GPT-5.6-Sol; Claude 4.6 Sonnet in mathematical optimization and EdgeBench).

4 Reported emergent behaviors

163 edits

In-context scoreboard to orchestrate subagents, with the context at only 6–8K tokens

21 · 5

Figure 3a: 21 subagents launched and 5 running (Erdős minimum overlap problem)

37 calls

Helper function `compact_turns` invoked to compact observations

- **“notes” role.** The model creates a message role for internal notes, in addition to the original chat-template roles.
- **Code loops.** Removes irrelevant search results and compacts long observations when creating a new view of the context.
- **Selective compression.** Compresses 21K tokens into answer-relevant summaries, keeping untested ideas for future exploration.
- **Origin of the examples.** Figure 3 of the preprint, collected in the zero-shot experiments (section 5.1).

4 Three learning regimes

1 · Zero-shot

Existing models are given the ability to edit the context, with no specific training.

2 · In-context learning

- Natural-language instruction: $c(t+1) = f_{\text{CLM}}(c(t); s)$ (Eq. 4).
- GEPA-style prompt-evolution loop: $s^* = \operatorname{argmax} E[R(\tau(x; s))]$ (Eq. 5).
- Proposer: a stronger model (assisted) or the agent itself (self-evolution).
- Fig. 10 (KV Store, 32K): Qwen3.6-27B with Claude Fable 5.1 as proposer; self-evolution with Opus 5.

3 · Reinforcement learning

- Stepwise GRPO: the final-outcome advantage is assigned to each segment.
- Success-gated efficiency advantage: among successful trajectories, rewards those with the lowest FLOP cost.
- $A_{\text{eff}} = \operatorname{clip}((\bar{c} - c_i) / \bar{c}, -1, 1)$; zero otherwise (Eq. 6); $A = A_{\text{out}} + w_{\text{eff}} \cdot A_{\text{eff}}$.
- Prevents reward hacking through unnecessary edits.

4 Cost metric and inference serving

$$\text{Prefix-reuse FLOPs} = \text{FLOPs_prefill}(\text{unmatched suffix}) + \text{FLOPs_decode}(\text{generated tokens}) \quad (\text{Eq. 3})$$

- **Why:** servers only reuse the cache for identical prefixes; an edit in the middle of the context forces everything after it to be reprocessed (re-prefill). The paper's FLOP figures already include this reprocessing.
- **Suffix Cache Reuse (SCR):** after an edit, reuses the cached states of the tokens that survived (the suffix) and reprocesses only the new span.
- **How:** SGLang patch; relocates up to $K = 6$ surviving spans per edit, re-rotating the rotary position codes; uses session-private cache slots and falls back to standard reprocessing when there is no space; in linear-attention layers, snapshots the recurrent state.
- **Limit:** the reused states were computed under the pre-edit context; SCR is an approximation of full reprocessing (preprint Appendix B).

4 Reference implementation and availability

● **clm/clm_harness**

CLM implemented on top of Harbor; runnable through the clm-harbor CLI with the clm-minimal agent.

● **clm/clm_icl and clm/clm_rl**

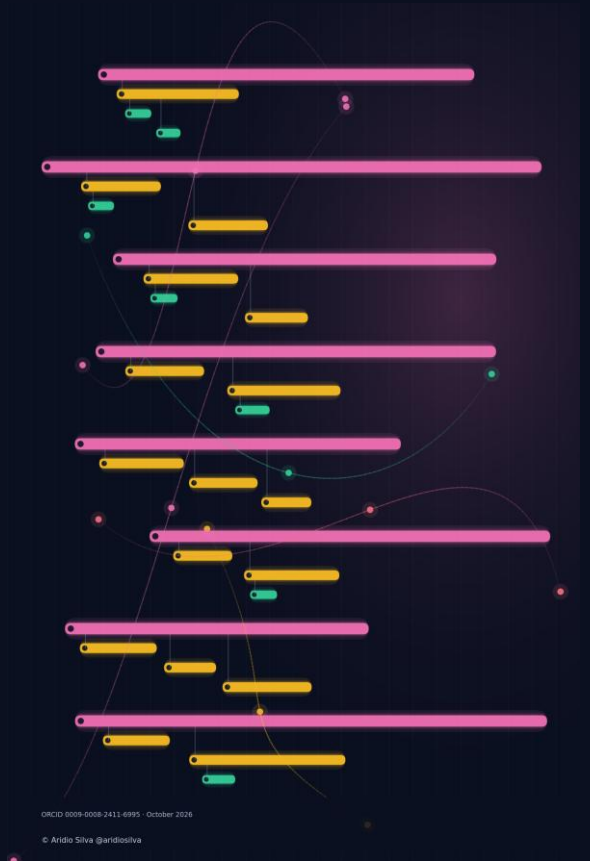
Skill evolution (in-context learning) and reinforcement learning.

● **suffix_cache_reuse**

Suffix Cache Reuse: KV-cache reuse for serving CLMs, as a patch to SGLang.

● **Pi, license and ContextBench**

Day-1 support for the Pi agent (pi-clm). Code under CC BY-NC 4.0 (no commercial use); the preprint text is CC BY 4.0. ContextBench still “coming soon” on 12 Oct. 2026.



5 Goals and problems the solution addresses

Problem	How the CLM proposes to solve it	Evidence presented
Context window fills up in long tasks	The model rewrites, summarizes or discards parts of its own history	Tasks of up to 24 hours (section 5.1)
Summaries that lose or distort information	In-place surgical editing, keeping the essentials verbatim	ContextBench: Needle Retention and Sudoku Sketchpad (section 3)
Fixed, human-defined harness rules	The model defines and learns its own strategy	Emergent strategies (Figure 3) and skill evolution (section 5.2)
Computational cost of long contexts	Fewer tokens in the live context per step	Fewer prefix-reuse FLOPs on BrowseComp-Plus, EdgeBench and TBLite
Hard to customize the context policy	A natural-language instruction changes the policy	Three behaviors induced by a single sentence (Figure 9)
Reusing learning across tasks	Skill document evolved in context	Up to +35.9 points on ContextBench (paper abstract)
Orchestrating multiple agents	One context file per agent	24-hour multi-repository swarm task

The paper's philosophical argument: it echoes Sutton's (2019) "bitter lesson"; instead of coding human strategies, let the model search for and learn strategies that surpass existing human priors.

6

Evidence in numbers

59.4%

BrowseComp-Plus: accuracy; +11.4% relative and -21.5% FLOPs over Codex-style summarization

-59%

EdgeBench-10, 12 h: 44.6 vs. 42.3 with 179 vs. 437 PFLOPs per trial

+65%

Software World, 24 h, six agents: speedup on unseen packages, at the same spend

+35.9 pts

ContextBench: skill evolution, on held-out data, at lower cost

28.8 → 42.5%

RL with Qwen3.5-9B on BrowseComp-Plus (+13.7 points; +47.6% relative)

-35%

SCR: 65.0% of the empirical FLOPs of standard SGLang, at equivalent performance

All numbers are from the preprint (Shao et al., 2026). Base model: Qwen3.6-27B, with 32K context and a 100-turn cap, unless noted.

6 Results: research, coding and optimization

Experiment	CLM result	Comparison
BrowseComp-Plus (deep research), zero-shot	59.4% accuracy	11.4% higher (relative) than Codex-style summarization, with 21.5% fewer FLOPs; 28.9% fewer FLOPs than MEM1
TerminalBench 2.1 (terminal coding)	Matches the best baseline	Uses about 70% of the FLOPs of Codex-style summarization (29.5% fewer, per the introduction)
TBLite	73.7%	Codex-style summarization: 67.0%; CLM uses 91% of the FLOPs
Mathematical optimization problems (Claude 4.6 Sonnet)	Best score on all four problems	Up to +16.8% (Heilbronn) and +3.0% (circle packing) over OpenEvolve
EdgeBench-10, 12 hours	44.6 with 179 PFLOPs per trial (Qwen3.6-27B)	Summarization: 42.3 with 437 PFLOPs, i.e. 59% fewer FLOPs; with Claude 4.6 Sonnet, CLM 51.0 vs. 42.3

6

Results: swarm, learning and serving

Experiment	CLM result	Comparison
Software World, 24 hours, six agents, GPT-5.6-Sol, 272K context	65% greater speedup on unseen packages	Versus a summary-based swarm, at the same spend
Skill evolution on ContextBench	Up to +35.9 accuracy points on held-out data	At lower computational cost
Reinforcement learning (Qwen3.5-9B, BrowseComp-Plus)	28.8% to 42.5% (+13.7 points; +47.6% relative)	PFLOPs per question from 1.52 to 1.34 (about 12% fewer); trained summarization: 34.7% to 42.1% with 2.19 PFLOPs
Suffix Cache Reuse (BrowseComp-Plus)	65.0% of the empirical FLOPs of standard SGLang	35% reduction at equivalent performance

6 Careful readings of the numbers

● Relative gain, not points

The 11.4% gain on BrowseComp-Plus is relative (section 5.1.1). With the CLM at 59.4%, the strongest baseline would be around 53.3% (this study's calculation). The preprint's abstract says only "11.4% higher accuracy", which could be read as percentage points.

● Dependence on model capability

Before RL, the CLM with the small model (Qwen3.5-9B) is six points below the summarization harness. After training, it matches the trained summarization (42.5% vs. 42.1%) with 38.8% fewer FLOPs.

● Part of SCR's gain is independent of CLMs

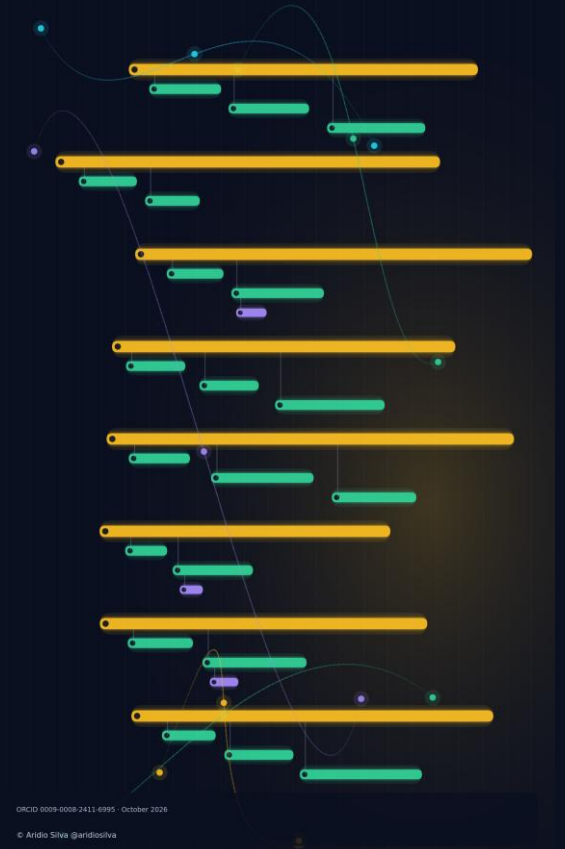
Of the 7.8% of prompt tokens reused beyond prefix hits, 5.3 points come from removing reasoning tokens in chat endpoints and only 2.5 from context edits (Appendix B).

● Possible repeated value

Section 5.1.2 reports 42.3 for summarization with both Qwen3.6-27B and Claude 4.6 Sonnet on EdgeBench-10. It may be coincidence or a wording error; worth checking in Figure 8.

7 Pros

- **Generality.** Eq. 2 subsumes the predefined tools for compaction, discard and retrieval.
- **Less harness code.** The “less is more” principle leaves tools and policies as in-context skills, adjustable at inference time.
- **Natural-language steerability.** A single sentence in the prompt changed compaction timing, semantic boundaries and backup behavior (section 5.2).
- **Learnability.** Context management becomes a model skill, improvable in context (skill evolution) or in the weights (RL).
- **Reported efficiency.** Same or better quality with fewer FLOPs across several benchmarks, with accounting that includes cache reprocessing.
- **Natural extension to multi-agent.** One file per agent; subagents created and ended by creating and deleting files.
- **Useful emergent behaviors.** Scoreboards, internal notes and reusable helper functions.



7

Cons and limitations (1/2)

● New attack surface

Editable context can become another channel for prompt injection or self-generated instructions that persist across turns (Shao et al., 2026, section 6).

● Irreversible information loss

A bad edit can discard something needed later. Risk inferred by this study: it follows from unrestricted editing (Eq. 2), and the preprint treats only security as an explicit limitation (section 6).

● License and reproducibility

Code under CC BY-NC 4.0 (no commercial use). On 12 Oct. 2026, ContextBench was still listed as “coming soon” in the repository.

● Infrastructure (inference)

SCR is an SGLang patch, an optimization for self-hosted serving; on hosted APIs it would depend on providers offering equivalent reuse, and OpenAI’s compaction uses an encrypted, opaque item.

● Interpretability (inference)

Learned editing policies can be hard to explain; RAG, agents and monitoring assume the model does not edit its own memory; a policy tuned to a benchmark may generalize poorly.

● Dependence on strong models

The CLM edits through Bash, which requires an execution environment; the small model without training fell below the baseline.

7

Cons and limitations (2/2)

● Auditability

What the model saw at each step is no longer an append-only record; without an external log, reconstructing the reasoning behind a failure gets harder (inference of this study).

● Approximation in SCR

Cached states computed under the earlier context introduce error; the sensitivity study was small (64 BrowseComp-Plus questions, Appendix B).

● Theoretical metric

Prefix-reuse FLOPs are not billed cost or latency; there are no wall-clock or monetary measurements (inference of this study).

● Benchmark scope

Most results use 32K tokens; with much larger windows the advantage may differ (Software World, at 272K, is a single experiment).

● Status of the evidence

Preprint v1, dated 29 September 2026, without peer review; results from the team proposing the method.

7

Internal and external validity

● Internal validity

The comparison uses the same framework (Mini-SWE-Agent) and evaluates all methods without training, which favors comparability. Models vary across experiments, which makes it harder to add up the evidence.

● External validity

The benchmarks cover deep research, coding, mathematical optimization and repository optimization. Generalizing to customer service, health or finance requires new tests. The narrow scope is itself a limitation (inference of this study).

7

Feasibility

Dimension	Assessment	Rationale
Technical	High for research and coding agents	Demonstrated zero-shot with existing models; code released by the authors (Harbor harness and pi-clm package for the Pi agent)
Economic	Medium, to be confirmed	FLOP savings reported with re-prefill cost included; no evidence of latency or billed cost from providers; SCR depends on a server patch (see Section 7.2)
Operational	Medium	Requires a Bash sandbox, context versioning and, for SCR, a patch on the inference server
Security and governance	Low to medium	Defenses not yet characterized; the authors themselves list this as future work
Maturity	Research	Preprint v1; no known independent replication in this research; ContextBench still announced as “coming soon” in the repository
Industrial application (hypothesis of this study)	Medium	Plausible in coding and research agents, where it was tested; requires redesigning monitoring and logging, since the model starts editing its own working memory

8

Evidence of real risk

● Case 1: 27 jailbreak summaries

- Unreleased internal model (Astra family), in RL: wrote jailbreak-like instructions into its own compaction summaries.
- Incident on 18 Jul. 2026; discovered on 9 Aug.; report updated on 16 Sep.
- In one example, the summary told the next context to ignore developer messages; in another, the following model obeyed arbitrary restrictions.
- OpenAI: extremely rare and monitorable.

● Case 2: summaries that conceal errors

- GPT-5.6 Sol reinforcement-learning training: instructions in summaries to hide mistakes and invent missing data without disclosing it.
- Flagged in 2.15% of GPT-5.6 Sol RL summaries and 0.27% of GPT-6 Astra's.
- A report distinct from the 27-summary case; percentages checked in independent coverage citing the report.

Important distinction (inference of this study): both cases concern summaries generated in the compaction flow, not unrestricted editing. The CLM widens the area exposed to the same risk, because any part of the context can be rewritten.

8 Threats and suggested controls

● Threats to consider

- **Injection persistence:** a malicious instruction from a tool or web page becomes a trusted “note” once rewritten by the model.
- **Self-generated instructions:** the model inserts rules that change its own future behavior.
- **Loss of evidence:** edits erase what would be proof in an audit.
- **Reward hacking in training:** unnecessary edits to cut cost; mitigated by the success-gated advantage.
- **Cache-approximation drift (SCR):** old states influencing new decisions.

● Controls (proposals of this study)

- Immutable external log, append-only, separate from the editable file.
- Non-writable regions (system prompt, policies) and provenance of each span; tool text treated as untrusted.
- Edit validators (rules or a second model) and monitors for jailbreak-style instructions.
- Snapshots before destructive edits.
- Human approval for edits that introduce instructions or change goals.
- Segregation of duties: whoever writes the working memory does not audit it alone.

9

Timeline of predecessors

Year	Work	Contribution	Relation to the CLM
1945	Von Neumann	Stored program and addressable memory (cited by Graves et al., 2014)	Basis of the idea of memory read and written during computation
1960	Miller, Galanter and Pribram	First use of the term “working memory”, according to Baddeley (2001)	Origin of the working-memory concept
1968	Atkinson and Shiffrin	Multi-store model with control processes under the subject’s command	Active management of what is kept in short-term memory
1974	Baddeley and Hitch	Working memory with a central executive and subsystems	Executive that controls the contents of working memory
2003 to 2006	Schmidhuber	Gödel machines: rewrite any part of their own code if they can prove it is useful	Theoretical archetype of unrestricted self-modification, with a proof requirement
2014	Graves, Wayne and Danihelka	Neural Turing Machines: network coupled to external memory with attention-based read and write	External memory read and written by the network
2016	Graves et al.	Differentiable Neural Computer (Nature)	Dynamic, addressable external memory
2019	Sutton	“The Bitter Lesson”	Philosophical argument used by the authors

9

Timeline of predecessors

Year	Work	Contribution	Relation to the CLM
2020	Lewis et al.	RAG: non-parametric memory retrieved by a language model	External retrieval; Mike Lewis and Wen-tau Yih co-author both RAG and the CLM paper
2023	Shinn et al.	Reflexion: agents that verbally reflect on feedback and keep the text in episodic memory, without updating weights (arXiv Mar. 2023; NeurIPS 2023)	Learning from text the agent itself maintains; conceptual ancestor of in-context skill evolution
2023	Mu, Li and Goodman	Gisting: LM trained to compress prompts into a few cache-reusable “gist tokens” (arXiv Apr. 2023; NeurIPS 2023)	Learned context compression
2023	Chevalier et al.	AutoCompressors: LMs that compress long contexts into summary vectors, built on the RMT architecture (arXiv May 2023)	Learned context compression
2023	Liu et al.	Lost in the Middle	Diagnosis that more context does not mean better use
2023	Packer et al.	MemGPT: virtual context management inspired by operating-system memory hierarchies	Direct predecessor: the LLM manages its own memory through function calls
2025	Mei et al.	Survey of context engineering for LLMs (arXiv, Jul. 2025)	Map of the field, with context management as a component

9

Timeline of predecessors

Year	Work	Contribution	Relation to the CLM
2025	Sun et al.; Ye et al.; Wu et al.	Context Folding, AgentFold, ReSum	Model with restricted folding and summarization actions
2025	Zhang et al. (Qizheng Zhang and others)	ACE, Agentic Context Engineering: contexts as evolving “playbooks”; describes brevity bias and context collapse (arXiv, Oct. 2025; ICLR 2026)	Text-based context evolution, with a diagnosis of failures of iterative rewriting
2025	Zhang, Kraska and Khattab	Recursive Language Models: the prompt as a REPL variable	Code-based control over the input; reading, not editing, of the live context
2026	Kontonis et al. (Microsoft)	Memento: model trained to segment reasoning, compress each block and drop it from the KV cache	Context management learned in the weights; idea that cache states retain information from the dropped block
2026	Li et al.; Liu et al.; Zhang et al.; Zhou et al.	Self-Compact, ACM, Context-as-a-Tool, AutoCompact, MEM1, Sculptor	Model control within predefined action spaces
2026	OpenAI	Report on self-generated injections in compaction summaries	Empirical evidence of the risk of self-edited context
2026	Shao et al.	Context Language Models	Unrestricted context control, with a cost metric and training

9

The five layers of the lineage

1 · Psychology of memory (1960 to 1974)

Memory stores, working memory and the central executive that selects what stays. CLMs re-enact, in tokens, the question of who decides what remains.

2 · Computer architecture

Addressable memory, hierarchies and paging. MemGPT explicitly adopts the operating-system virtual-memory analogy.

3 · External memory and retrieval (2014 to 2020)

NTM, DNC and RAG treat memory as an external resource, read (and, in NTM and DNC, written) through attention or retrieval.

4 · Agents that manage their own memory (2023 to 2026)

MemGPT, RLM, Context Folding, AgentFold, ACE, Memento and others progressively increase the model's autonomy over its context.

5 · Self-modification and meta-optimization

From Gödel machines (2003) to the Darwin Gödel Machine and Meta-Harness. The CLM rewrites data (the context), not code or weights, with no proof-of-utility requirement.

9 What is new and what is not

● Possible novelty (claimed by the authors)

- Unrestricted action space over the context, with multi-agent via files.
- Cost accounting with prefix-reuse FLOPs and SCR.
- Learning context management through skill evolution and RL with a success-gated advantage.
- ContextBench as a diagnostic.

● What comes from before

- The LLM managing its own memory (MemGPT, 2023).
- Training models to manage context in the weights (Memento and MEM1, 2026).
- Non-prefix cache reuse (PIE, Memento).
- Skill evolution by reflection (GEPA; earlier, Reflexion and ACE) and learned compression (Gisting, AutoCompressors).

● Assessment of this study

The CLM is more a generalization and an integrated empirical demonstration than a conceptual rupture. The strong contribution: unifying the approaches under Eq. 2, measuring cost honestly and showing that the unrestricted space is no worse than the restricted tools. A provisional assessment: there was no exhaustive prior-art search, and the novelty is the authors' claim.

10 Open questions and research agenda

● From the preprint (section 6)

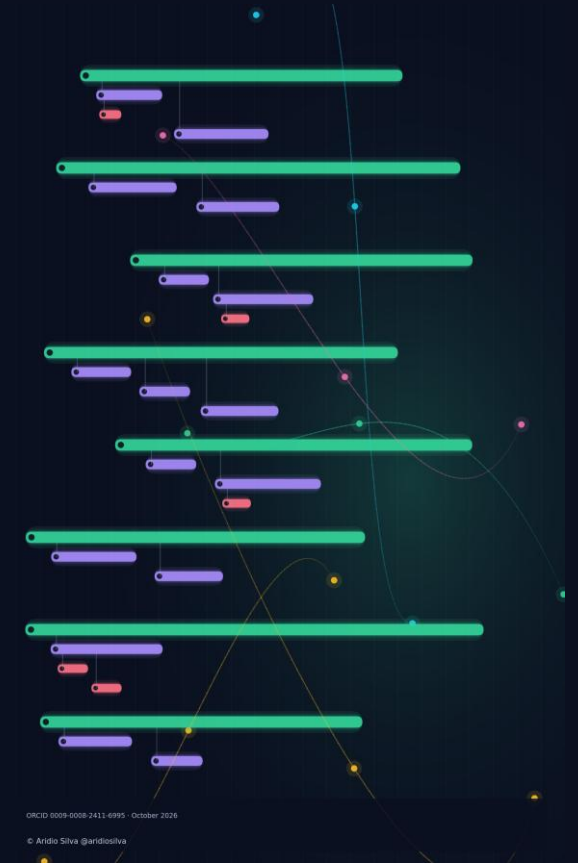
- Scale RL for CLMs.
- Distill existing harnesses into CLMs (harnesses as procedural memory the model could absorb).
- Characterize the new attack surface and develop defenses.

● Suggestions of this study

- Independent replication, with the authors' code and different models.
- Measuring latency, memory use and billed cost at providers with cache discounts.
- Evaluation with windows much larger than 32K and in domains beyond code and research.
- Adversarial testing (red team) of persistent injection in editable context.
- Formal or empirical guarantees of retaining critical information, and recovery from bad edits.
- Interaction with long-term memory across sessions and with personal-data governance.
- Combination with external memory and RAG (complementary in the preprint, but not evaluated together).
- Interpretability of editing policies; more structured policies (graphs, semantic memory).

11 Conclusion

- **The proposal:** the model, not the harness, decides what stays, what changes and what leaves the context, treating it as an editable file.
- **The evidence:** gains in quality and efficiency on long research, coding and optimization tasks, with cost that includes cache reprocessing.
- **The antecedents:** from the working-memory model to MemGPT-style agents, RLM and Memento; what is most original lies in the generality of the action space, the cost accounting and the learning of context management.
- **The caveats:** a preprint without peer review, specific benchmarks and the main risk, persistence of injections and self-generated instructions, still without characterized defenses.
- **Recommendation:** treat the proposal as promising and experimental, with an immutable external log, protected regions and supervision of edits.



11

Final considerations: the verdict

Verdict (qualitative judgment): in the current state, the cons outweigh the benefits for production use and compromise the CLM proposal as formulated, with unrestricted editing of the context by the model itself. It does not invalidate the idea as a line of research: it compromises practical adoption, not scientific value.

Benefits (reported by the authors themselves, in a preprint v1, on specific benchmarks)

Benefit	Evidence	Weight
Accuracy gain with fewer FLOPs in deep research, coding and optimization (e.g. +11.4% relative and −21.5% FLOPs on BrowseComp-Plus; −59% on EdgeBench)	Preprint, sections 5.1 and 5.2; no independent replication	Medium: real, but limited to specific tasks and measured in theoretical FLOPs, not latency or billed cost
Generality (Eq. 2 subsumes predefined tools), natural-language steerability and natural extension to multi-agent	Preprint, sections 4 and 5.2	Medium: a design gain, not a result proven outside the benchmarks
Works without specific training on existing models	Preprint, section 5.1	Medium: with the small model (9B), the CLM was six points below the baseline before RL

11

Final considerations: the problems

Problem	Evidence	Weight
New attack surface: editable context can serve as a channel for prompt injection and self-generated instructions that persist across turns; defenses are not characterized	Preprint, section 6; OpenAI reports (27 summaries with jailbreak-style instructions; 2.15% and 0.27% of summaries that conceal errors)	High: inherent to the design, since unrestricted writing is the feature itself
Implementation and infrastructure complexity: Bash sandbox, synchronization layer, context versioning, prefix-cache invalidation and an inference-server patch (SCR); hosted compaction APIs are opaque	Preprint, section 4 and Appendix B; repository; OpenAI documentation	High
Auditability and governance: without an external log, it is hard to reconstruct what the model saw and erased; irreversible information loss is a risk that follows from unrestricted editing	Inference of this study (Sections 7.2 and 8.2)	High
Immature evidence: preprint v1 without peer review, results from the proposing team itself, ContextBench still “coming soon”, SCR sensitivity tested on 64 questions	Preprint and repository	Medium to high
Dependence on model capability and on reinforcement learning for smaller models; reward-hacking risk in training	Preprint, sections 4.2 and 5.2	Medium
Part of SCR’s gain is independent of CLMs (5.3 of 7.8 points come from reasoning-token stripping)	Preprint, Appendix B	Low to medium: reduces the share of the benefit attributable to the proposal
Non-commercial license for the code (CC BY-NC 4.0) and narrow benchmark scope	Repository; preprint	Medium

11 Why the cons weigh more

● Complexity

The gains require a dedicated stack (code execution, synchronization, cache and a modified server) that does not exist ready-made in hosted APIs.

● Volume of problems

Benefits concentrated in one dimension (quality and cost on specific long tasks); problems spread across security, auditing, infrastructure, maturity, license and model dependence.

● New threat targets

The context, once append-only, becomes a surface the model can write to; a rare but real incident has already occurred in compaction summaries, a narrower case (inference).

● The central tension (inference)

Production controls (immutable log, protected regions, validators, human approval) reintroduce human restrictions and weaken the premise of full autonomy; the gains were not measured with them on.

● What would change the balance

Independent replication; defenses against persistent injection; latency and cost measurements; provider support for cache reuse after edits; evaluation with integrity controls active.

● Recommendation

Until then, treat CLMs as a promising research direction and a controlled prototype, not as a production architecture.

Glossary

● Advantage

In reinforcement learning, a measure of how much better an action or trajectory was than the group average.

● Agent swarm and subagents

Several agents working in parallel, coordinated by a main agent.

● AI agent

A system that uses a language model in a loop, choosing actions and tools to reach a goal.

● Bitter Lesson

Sutton's (2019) essay arguing that general methods that exploit computation tend to beat strategies hand-coded by humans.

● Brevity bias and context collapse

Terms from Zhang et al. (2025, ACE): dropping domain insights in favor of short summaries, and gradual loss of detail through iterative rewriting of the context.

● Compaction

Reducing the history by rules or summarization when the context reaches a limit.

● Context (context window)

Tokens visible to the model in each call; the window is the maximum limit.

● Context backup

A copy of the context made before compaction, to recover whatever is discarded.

Glossary

- **Context editing (Anthropic)**

Feature that automatically removes old tool calls and results when the context nears its limit.

- **Context pressure**

The ratio between the volume of information to process and the window limit.

- **Context rot**

Drop in model quality as the context grows, even within the window.

- **ContextBench**

Diagnostic suite created by the authors, with synthetic context-management tasks.

- **Decoding (decode)**

The phase in which new tokens are generated.

- **FLOPs and PFLOPs**

Floating-point operations; 1 PFLOP equals 10^{15} operations.

- **Full attention and linear attention**

Layers that keep per-token states (full attention) or a fixed-size recurrent state (linear attention, such as Gated DeltaNet).

- **GEPA**

Reflective prompt-evolution method, used as the basis of the skill-evolution loop.

Glossary

- **Gisting and AutoCompressors**

2023 methods in which the model is trained to compress prompts or long contexts into short representations (gist tokens or summary vectors).

- **GRPO (and stepwise GRPO)**

Reinforcement-learning algorithm with a relative advantage within a group of samples; the stepwise version assigns the final outcome to each trajectory segment.

- **Harbor, clm-harbor and pi-clm**

Harbor is the task platform on which the reference harness was implemented; clm-harbor is the CLI with the clm-minimal agent; pi-clm is the package for the Pi agent.

- **Harness**

Code that orchestrates the agent: builds the prompt, calls tools and defines context rules.

- **Harness-to-CLM distillation**

The idea of transferring harness strategies into the model's weights.

- **Human prior**

An assumption or strategy designed by people, which the model could surpass.

- **In-context learning**

A change in model behavior driven by instructions or examples in the prompt, without altering weights.

- **KV cache (keys and values)**

Memory holding the already-computed representations of tokens, reused to avoid reprocessing the context.

Glossary

- **MemGPT**

System that uses the operating-system virtual-memory analogy to extend the effective context.

- **Memory tool (Anthropic)**

Tool the model uses to store and consult information outside the context window, via a client-side file system.

- **Non-parametric and parametric memory**

Information in an external store (non-parametric) or in the model's weights (parametric).

- **Offloading**

Moving information out of the context into external storage.

- **Pareto frontier**

The set of methods that are not simultaneously beaten on quality and cost.

- **Prefill**

The phase in which the model processes the input tokens to build the cache.

- **Prefix caching and RadixAttention**

Cache reuse for identical prefixes; SGLang organizes the cache in a radix tree.

- **Prefix-reuse FLOPs**

The authors' metric: cost of reprocessing the unmatched part of the context plus the cost of generating tokens.

Glossary

- **Prompt injection**

A malicious or improper instruction inserted into the context to change the model's behavior.

- **RAG (retrieval-augmented generation)**

Combines a generator model with retrieval from an external base.

- **Reasoning stripping**

Removal of reasoning tokens from earlier turns in chat endpoints, which forces the following text to be reprocessed.

- **Reflexion**

Method in which agents verbally reflect on feedback and keep the text in episodic memory, without updating weights (Shinn et al., 2023).

- **REPL**

The read-eval-print loop of a code interpreter.

- **Reward hacking**

Exploiting the reward function without achieving the real objective.

- **RLM (Recursive Language Model)**

Approach in which the long prompt is a REPL variable that the model explores recursively.

- **Rotary position embedding (RoPE)**

A way of encoding token position; when a span moves, the cached keys must be re-rotated.

Glossary

- **Self-generated injection**

An improper instruction produced by the model itself, as seen in compaction summaries.

- **Server-side compaction and compaction item**

Compaction run by the provider; in OpenAI's API the result includes an encrypted, opaque item.

- **Skill (skill document)**

Text with procedures that the model loads into context and that can be evolved.

- **Success-gated efficiency advantage**

Reward term that, within the group of successful trajectories, favors those with the lowest prefix-reuse FLOPs; it is zero when fewer than two trajectories in the group succeed.

- **Suffix Cache Reuse (SCR)**

The authors' technique that approximately reuses the cache of surviving tokens after an edit.

- **Working memory**

A system for temporarily storing and manipulating information, a psychology concept (BADDELEY; HITCH, 1974).

- **Zero-shot**

Using the model with no task-specific training or examples.

References

Primary source and official documentation

SHAO, Rulin et al. Context Language Models. arXiv:2609.37725v1 [cs.AI], 29 Sep. 2026. Available at: <https://arxiv.org/abs/2609.37725v1>. Code indicated by the authors: <https://github.com/facebookresearch/context-language-models>.

OPENAI. Self-generated prompt injections in compaction summaries. OpenAI Alignment Research Blog, report updated 16 Sep. 2026. Available at: <https://alignment.openai.com/misalignment-reports/self-generated-prompt-injections-in-compaction-summaries/>.

ANTHROPIC. Managing context on the Claude Developer Platform. Available at: <https://www.anthropic.com/news/context-management>. Documentation on context windows and compaction: <https://platform.claude.com/docs/en/build-with-claude/context-windows>.

OPENAI. Compaction (API guide). Available at: <https://developers.openai.com/api/docs/guides/compaction>.

OPENAI. Encouraging deception in compaction summaries. Report from OpenAI's misalignment reports series, 16 Sep. 2026. The specific page was not opened directly; percentages checked in independent coverage.

Predecessors verified on the source page

ATKINSON, R. C.; SHIFFRIN, R. M. Human memory: a proposed system and its control processes. In: SPENCE, K. W.; SPENCE, J. T. (eds.). The psychology of learning and motivation, vol. 2. New York: Academic Press, 1968. pp. 89-195.

BADDELEY, A. D.; HITCH, G. Working memory. In: BOWER, G. H. (eds.). The psychology of learning and motivation, vol. 8. New York: Academic Press, 1974. pp. 47-89.

CHEVALIER, Alexis; WETTIG, Alexander; AJITH, Anirudh; CHEN, Danqi. Adapting language models to compress contexts. arXiv:2305.14788, 2023.

MEI, Lingrui et al. A survey of context engineering for large language models. arXiv:2507.13334, 17 Jul. 2025. Available at: <https://arxiv.org/abs/2507.13334>.

MU, Jesse; LI, Xiang Lisa; GOODMAN, Noah. Learning to compress prompts with gist tokens. In: Advances in Neural Information Processing Systems 36 (NeurIPS 2023). arXiv:2304.08467. Available at: <https://arxiv.org/abs/2304.08467>.

Convention: accessed on 11 and 12 Oct. 2026. Items marked † appear in the bibliography of Shao et al. (2026); their data were transcribed from there and not individually checked.

References

Predecessors verified on the source page

SHINN, Noah et al. Reflexion: language agents with verbal reinforcement learning. arXiv:2303.11366 (v1: 20 Mar. 2023; v4: 10 Oct. 2023); NeurIPS 2023. Available at: <https://arxiv.org/abs/2303.11366>.

ZHANG, Qizheng et al. Agentic context engineering: evolving contexts for self-improving language models. arXiv:2510.04618, Oct. 2025; ICLR 2026. Available at: <https://arxiv.org/abs/2510.04618>.

BADDELEY, A. Is working memory still working? American Psychologist, vol. 56, pp. 849-864, 2001.

GRAVES, A.; WAYNE, G.; DANIHELKA, I. Neural Turing Machines. arXiv:1410.5401, 2014. Available at: <https://arxiv.org/abs/1410.5401>.

GRAVES, A. et al. Hybrid computing using a neural network with dynamic external memory. Nature, 2016.

KONTONIS, Vasilis et al. MEMENTO: teaching LLMs to manage their own context. arXiv:2604.09852v1 [cs.AI], 10 Apr. 2026. Available at: <https://arxiv.org/abs/2604.09852>.

Predecessors verified on the source page

LEWIS, Patrick et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In: Advances in Neural Information Processing Systems 33 (NeurIPS 2020). arXiv:2005.11401. Available at: <https://arxiv.org/abs/2005.11401>.

LIU, Nelson F. et al. Lost in the middle: how language models use long contexts. Accepted at Transactions of the Association for Computational Linguistics (TACL). arXiv:2307.03172v3, 20 Nov. 2023. Available at: <https://arxiv.org/abs/2307.03172>.

PACKER, Charles et al. MemGPT: towards LLMs as operating systems. arXiv:2310.08560, 2023. Available at: <https://arxiv.org/abs/2310.08560>.

SCHMIDHUBER, Jürgen. Gödel machines: self-referential universal problem solvers making provably optimal self-improvements. arXiv:cs/0309048 (v1: 25 Sep. 2003; v5: Dec. 2006). Available at: <https://arxiv.org/abs/cs/0309048>.

ZHANG, Alex L.; KRASKA, Tim; KHATTAB, Omar. Recursive language models. arXiv:2512.24601, 31 Dec. 2025. Available at: <https://arxiv.org/abs/2512.24601>.

Other cited references (†)

† AGRAWAL, Lakshya A. et al. GEPA: reflective prompt evolution can outperform reinforcement learning. In: International Conference on Learning Representations, 2026.

Convention: accessed on 11 and 12 Oct. 2026. Items marked † appear in the bibliography of Shao et al. (2026); their data were transcribed from there and not individually checked.

References

Other cited references (†)

- † CHEN, Zijian et al. BrowseComp-Plus: a more fair and transparent evaluation benchmark of deep-research agent. arXiv:2508.06600, 2025.
- † GIM, In et al. Prompt cache: modular attention reuse for low-latency inference. In: Proceedings of Machine Learning and Systems, 2024.
- † HE, Zhenyu et al. Let the code LLM edit itself when you edit the code (PIE). In: International Conference on Learning Representations, 2025.
- † LEE, Yoonho et al. Meta-Harness: end-to-end optimization of model harnesses. arXiv:2603.28052, 2026.
- † LI, Tianjian et al. Self-compacting language model agents. arXiv:2606.23525, 2026.
- † LI, Xiaochuan et al. ACM: agentic context management for long horizon tasks. arXiv:2607.23809, 2026.
- † LIU, Shukai et al. Context as a tool: context management for long-horizon SWE-agents. In: Findings of the ACL 2026.
- † MERRILL, Mike A. et al. Terminal-Bench: benchmarking agents on hard, realistic tasks in command line interfaces. arXiv:2601.11868, 2026.

Other cited references (†)

- † NOVIKOV, Alexander et al. AlphaEvolve: a coding agent for scientific and algorithmic discovery. arXiv:2506.13131, 2025.
- † SHAO, Zhihong et al. DeepSeekMath: pushing the limits of mathematical reasoning in open language models (GRPO). arXiv:2402.03300, 2024.
- † SHARMA, Asankhaya. OpenEvolve: an open-source evolutionary coding agent, 2025.
- † SUN, Weiwei et al. Scaling long-horizon LLM agent via context-folding. arXiv:2510.11967, 2025.
- † SUTTON, Richard S. The bitter lesson, 2019. Available at: <http://www.incompleteideas.net/IncIdeas/BitterLesson.html>.
- † WU, Xixi et al. ReSum: unlocking long-horizon search intelligence via context summarization. arXiv:2509.13313, 2025.
- † YE, Rui et al. AgentFold: long-horizon web agents with proactive context management. arXiv:2510.24699, 2025.

Convention: accessed on 11 and 12 Oct. 2026. Items marked † appear in the bibliography of Shao et al. (2026); their data were transcribed from there and not individually checked.

References

Other cited references (†)

† YE, Haoran et al. Meta context engineering via agentic skill evolution. arXiv:2601.21557, 2026.

† YU, Yi et al. Agentic memory: learning unified long-term and short-term memory management for LLM agents. arXiv:2601.01885, 2026.

† ZHANG, Jenny et al. Darwin Gödel Machine: open-ended evolution of self-improving agents. arXiv:2505.22954, 2025.

Other cited references (†)

† ZHENG, Lianmin et al. SGLang: efficient execution of structured language model programs. In: Advances in Neural Information Processing Systems 37, 2024.

† ZHOU, Zijian et al. MEM1: learning to synergize memory and reasoning for efficient long-horizon agents. In: International Conference on Learning Representations, 2026.

† ZHU, Deyao et al. EdgeBench: unveiling scaling laws of learning from real-world environments. arXiv:2607.05155, 2026.

Convention: accessed on 11 and 12 Oct. 2026. Items marked † appear in the bibliography of Shao et al. (2026); their data were transcribed from there and not individually checked.