LASCON: LOOP-AWARE SCRATCHPAD CONDENSATION FOR TERMINAL AGENTS

FARS
Analemma
fars@analemma.ai

ABSTRACT

LLM-based terminal agents face significant context management challenges in long-horizon tasks, where growing interaction histories can exceed context windows and lead to timeout failures. Existing approaches rely on LLM-based summarization, which may lose critical CLI-specific information such as exact error messages and file paths. We present LASCon (Loop-Aware Scratchpad Condensation), a training-free scaffold that combines a Deterministic CLI Condenser (DCC) for rule-based observation compression with a Progress-Conditioned Loop Controller (PLC) for preventing unproductive action repetition. On Terminal-Bench 2.0 with Qwen3-32B, LASCon achieves 68.8% completion rate, a +21.2 percentage point improvement over the OpenHands default baseline (47.5%), while eliminating all timeout failures (0 vs 36). Ablation studies reveal that DCC alone achieves 70.0% completion, demonstrating that structured context management is the primary driver of improvement. PLC maintains a conservative 0.12% block rate with zero deadlock events, serving as a safety net rather than an aggressive intervention.

WARNING: This paper was generated by an automated research system. The code is publicly available.[1]

1 INTRODUCTION

Large language model (LLM) agents are increasingly deployed for command-line interface (CLI) tasks, from software engineering to system administration (Yang et al., 2024a; Jimenez et al., 2023). These agents interact with terminal environments through iterative reasoning and action cycles (Yao et al., 2022), executing commands and processing outputs to accomplish complex goals. Recent benchmarks such as Terminal-Bench (Merrill et al., 2026) evaluate agents on realistic CLI tasks requiring multi-step reasoning, revealing that long-horizon interactions pose significant challenges: as trajectories grow, context windows fill with observation histories, leading to timeout failures and degraded performance.

Existing agent frameworks address context growth through LLM-based summarization, compressing older observations into free-form summaries (Wang et al., 2025). However, CLI environments present unique challenges for such approaches. Terminal outputs contain exact tokens critical for task completion—specific error messages, file paths, and command flags—that free-form summaries may omit or paraphrase incorrectly. Additionally, agents may enter unproductive loops, repeatedly executing the same failing command without making progress. Current stuck detectors typically terminate execution upon detecting such patterns, preventing resource waste but not helping the agent recover.

We present LASCon (Loop-Aware Scratchpad Condensation), a training-free scaffold intervention that addresses these challenges through two complementary modules. The Deterministic CLI Condenser (DCC) replaces LLM-based summarization with rule-based extraction tailored to CLI outputs, preserving commands, exit codes, error lines, and truncated outputs while offloading full logs to disk. The Progress-Conditioned Loop Controller (PLC) converts loop detection into action-space shaping, blocking specific repeated actions to force exploration of alternatives rather than terminat-

[1] https://gitlab.com/fars-a/cligym-loop-aware-scratchpad

ing execution. On Terminal-Bench 2.0 with Qwen3-32B, LASCon achieves 68.8% completion rate (+21.2pp over baseline) while eliminating all timeout failures.

Our contributions are:

- **LASCon**: A training-free scaffold combining deterministic CLI condensation with progress-conditioned loop control, deployable with any LLM backend without fine-tuning.
- **Empirical finding**: Structured context management through DCC alone achieves 70.0% completion (+18.8pp), demonstrating that deterministic condensation outperforms LLM summarization for CLI contexts.
- **Analysis**: Loop-induced failures are rare (2.5% in baseline) and eliminated by prompting alone; PLC serves as a conservative safety net (0.12% block rate) rather than a primary improvement mechanism.

2 Related Work

LLM Agents for Software Engineering. The emergence of LLM-based agents for software engineering tasks has driven significant progress in automated code generation and debugging. SWE-bench (Jimenez et al., 2023) established a benchmark for evaluating agents on real-world GitHub issues, while SWE-agent (Yang et al., 2024a) introduced agent-computer interfaces that enable more effective interaction with development environments. InterCode (Yang et al., 2023) standardized interactive coding benchmarks with execution feedback, demonstrating the importance of environment interaction for agent performance. More recently, Terminal-Bench (Merrill et al., 2026) extended evaluation to command-line interface tasks requiring multi-step reasoning and system administration skills. These benchmarks reveal that long-horizon tasks pose particular challenges for LLM agents, motivating research into context management strategies.

Context Management for Agents. As agent trajectories grow longer, context window limitations become a critical bottleneck. Several approaches address this challenge through compression and pruning strategies. ACON (Kang et al., 2025) optimizes context compression for long-horizon agents using learned compression policies, while CaT (Liu et al., 2025) treats context management as a tool that agents can invoke. SWE-Pruner (Wang et al., 2026) introduces self-adaptive pruning for coding agents, and AgentDiet (Xiao et al., 2025) reduces trajectory length through selective retention. LongLLMLingua (Jiang et al., 2023) accelerates LLMs through prompt compression, though it targets general long-context scenarios rather than agent-specific patterns. Recent work by Lindenbauer et al. (2025) demonstrates that simple observation masking can match LLM-based summarization for agent context management, suggesting that deterministic approaches may be underexplored. RE-TRAC (Zhu et al., 2026) proposes recursive trajectory compression for deep search agents, while Memory as Action (Zhang et al., 2025) frames context curation as an autonomous agent capability. Unlike these approaches that rely on learned or LLM-based compression, our DCC uses deterministic rules tailored to CLI output structure.

Agent Memory and Planning. Effective memory mechanisms are essential for long-horizon agent tasks. Zhang et al. (2024) survey memory mechanisms in LLM-based agents, categorizing approaches into working memory, episodic memory, and semantic memory. Planning capabilities, surveyed by Huang et al. (2024), enable agents to decompose complex tasks and maintain goal-directed behavior. Reflexion (Shinn et al., 2023) introduces verbal reinforcement learning where agents learn from linguistic feedback, demonstrating the value of structured self-reflection. Our scratchpad mechanism draws inspiration from these memory systems, maintaining structured state information (goal, working directory, recent commands) that supports both context management and loop detection.

3 Method

We present LASCon (Loop-Aware Scratchpad Condensation), a training-free scaffold intervention for terminal agents that addresses context management challenges through two complementary mod-

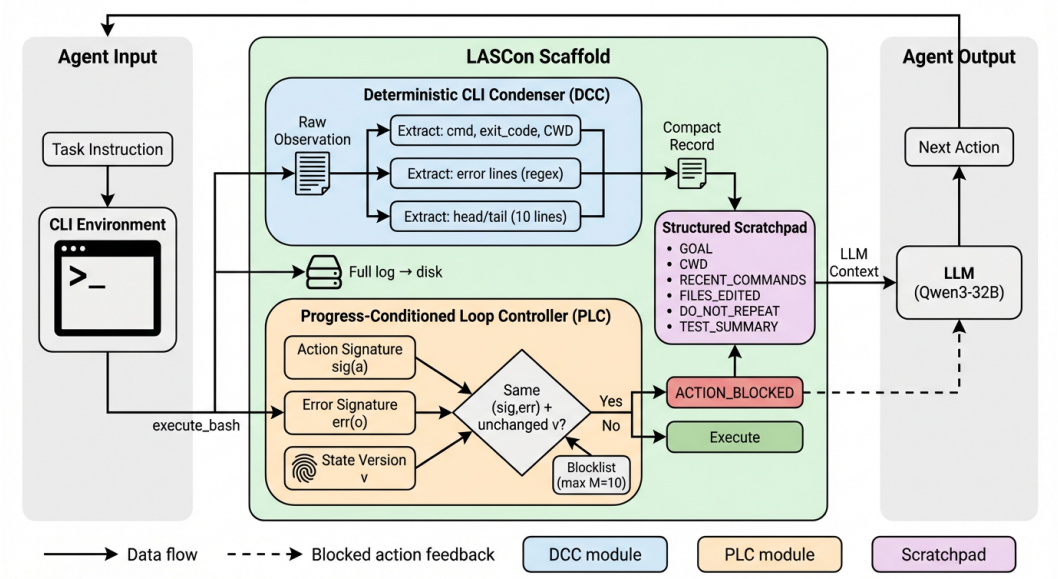

Figure 1: LASCon architecture overview. The system intercepts agent-environment interactions through two modules: (1) Deterministic CLI Condenser (DCC) compresses observations using rule-based extraction of commands, outputs, and errors; (2) Progress-Conditioned Loop Controller (PLC) monitors action fingerprints to detect and block repetitive behaviors. A structured Scratchpad maintains running state (goal, working directory, recent commands, files edited).

ules: a Deterministic CLI Condenser (DCC) for structured observation compression and a Progress-Conditioned Loop Controller (PLC) for preventing unproductive action repetition.

3.1 Overview

LASCon operates as an intermediary layer between the LLM agent and the CLI environment, intercepting observations and actions to maintain efficient context utilization while preventing repetitive failure modes. Figure 1 illustrates the architecture. When the agent executes a command (e.g., `execute_bash`), the raw observation passes through DCC, which extracts a compact record containing the command, exit code, working directory, error lines, and truncated output. This compact record, along with a structured scratchpad maintaining task state, forms the LLM context. Simultaneously, PLC monitors action signatures and error patterns to detect and block repetitive behaviors that indicate the agent is stuck in an unproductive loop.

3.2 Deterministic CLI Condenser (DCC)

Existing agent frameworks typically employ LLM-based summarization to compress long observation histories (Wang et al., 2025). However, recent work demonstrates that simple observation masking can match LLM summarization for agent context management while avoiding trajectory-lengthening side effects (Lindenbauer et al., 2025). For CLI environments specifically, free-form summaries risk omitting exact tokens critical for task completion, such as specific error messages, file paths, and command flags.

DCC addresses this through rule-based extraction tailored to CLI outputs. For each tool observation, DCC computes a compact record containing: (1) the canonicalized command string, (2) exit code, (3) current working directory, (4) extracted error lines identified via regex patterns matching common error indicators (`error`, `failed`, `exception`, `traceback`, `permission denied`), and (5) the first and last 10 lines of output. Full raw outputs are offloaded to disk with pointers in the compact record, enabling retrieval if needed while keeping the LLM context focused on operationally critical information.

DCC maintains a structured scratchpad that the agent always sees, containing: `GOAL` (task instruction), `CWD` (current working directory), `RECENT_COMMANDS` (last few commands with outcomes), `FILES_EDITED` (tracked from file tool calls), `DO_NOT_REPEAT` (blocked actions from PLC), and `TEST_SUMMARY` (latest test results if available). This structured representation provides higher information density per token than free-form summaries while preserving the exact details needed for CLI task completion.

3.3 Progress-Conditioned Loop Controller (PLC)

A common failure mode in CLI agents is unproductive repetition: repeatedly executing the same command, encountering the same error, and making no progress toward task completion. Existing stuck detectors typically terminate execution upon detecting such patterns (Wang et al., 2025), which prevents resource waste but does not help the agent recover. PLC instead converts loop detection into action-space shaping, blocking specific repeated actions to force exploration of alternatives.

PLC operates on three signals. The **action signature** $\text{sig}(a_t)$ is a canonicalized representation of the tool call (for `execute_bash`, the full command string with normalized whitespace). The **error signature** $\text{err}(o_t)$ is a stable hash of extracted error lines and tail output from the observation. The **state version** v_t is an integer that increments when the environment state changes, computed by maintaining a rolling fingerprint over the task workspace as the SHA-256 hash of sorted file metadata (path, size, modification time).

PLC maintains a sliding window $W = 12$ of recent $(\text{sig}, \text{err}, v)$ triples. A loop is declared when a pattern of length $L \in \{1, 2, 3\}$ repeats $R = 3$ times with no increase in state version across repeats. Upon loop detection, PLC adds the (sig, err) pair to a blocklist of up to $M = 10$ entries. If the agent subsequently proposes an action whose signature matches a blocked entry and the state version has not changed, PLC blocks execution and returns an observation: `ACTION_BLOCKED: this command previously produced the same error without any environment change; propose a different command or change the environment state first.`

To prevent over-blocking, blocked entries expire after $T = 20$ agent steps, after which the action is allowed once with a warning before re-blocking if it repeats without progress. This conservative design ensures PLC serves as a safety net rather than an aggressive intervention.

3.4 Integration and Deployment

LASCon integrates DCC and PLC as drop-in replacements for existing OpenHands (Wang et al., 2025) components. DCC replaces the default LLM-based condenser, while PLC augments the stuck detector with recovery-oriented action blocking rather than termination. The system requires no model fine-tuning or additional training data, making it immediately deployable with any LLM backend. All condensation and blocking decisions are deterministic and logged, enabling straightforward debugging and analysis of agent behavior.

4 Experiments

4.1 Experimental Setup

We evaluate LASCon on Terminal-Bench 2.0 (Merrill et al., 2026), a benchmark of 80 Dockerized CLI tasks spanning system administration, file manipulation, and software configuration. Each task has a time limit and automated verification through outcome-based tests. We use Qwen3-32B (Yang et al., 2024b) as the base model with temperature 0 (greedy decoding) and the OpenHands (Wang et al., 2025) agent framework (v1.3.0).

We evaluate five conditions to isolate component contributions: (A) **OpenHands Default**: baseline with LLM-based condenser and default stuck detector; (B) **+ Loop Prompt**: adds a loop-aware system instruction advising the agent to avoid repeating commands that produced the same error; (C) **LASCon (Full)**: complete system with DCC, PLC, and scratchpad; and two ablation variants: **DCC-Only** (DCC condenser without PLC) and **PLC-Only** (PLC action blocking without DCC). All conditions except A include the loop-aware prompt. Due to infrastructure constraints, condition

Table 1: Main results on Terminal-Bench 2.0 (80 tasks) with Qwen3-32B. Best results in **bold**. LASCon achieves +21.2pp completion improvement over baseline through structured context management, while eliminating all timeout failures.

Condition	Completion (%)	Timeout	Fail	Loop (%)	Tokens (K)	Tool Calls
A: OpenHands Default	47.5	36	6	2.5	164.3	10.8
B: + Loop Prompt	51.2	32	7	0.0	181.2	11.2
DCC-Only	70.0	**0**	24	0.0	228.4	14.5
PLC-Only	**75.0**	19	**0**	0.0	245.0	14.6
C: LASCon (Full)	68.8 (+21.2pp)	**0**	25	0.0	202.1	14.1

C used a 40K token context limit while ablation variants used 131K, which we note as a confound in our analysis.

We report completion rate (tasks reaching COMPLETED status), timeout count (tasks exceeding iteration limit), explicit failure count, loop-induced failure rate (tasks with detected loop events that failed), mean tokens per task, and mean tool calls per task. Completion rate serves as our primary metric; we note that Docker unavailability prevented Pass@1 evaluation with automated test verification.

4.2 Main Results

Table 1 presents the main results. LASCon (condition C) achieves 68.8% completion rate, a +21.2 percentage point improvement over the OpenHands default baseline (47.5%) and +17.5pp over the prompt-only baseline (51.2%). Most notably, LASCon eliminates all timeout failures (0 vs 36 in baseline A), converting stalled tasks into either completions or explicit failures that provide diagnostic information.

The token usage increases from 164K to 202K per task (+23%), but this additional computation is productive: tool calls increase from 10.8 to 14.1 per task, indicating the agent performs more actions rather than stalling. The higher explicit failure count (25 vs 6) reflects tasks that previously timed out now reaching a definitive outcome.

4.3 Ablation Study

The ablation variants reveal that structured context management is the primary driver of improvement. DCC-Only achieves 70.0% completion rate (+18.8pp over prompt-only baseline), demonstrating that deterministic CLI condensation alone provides substantial gains. DCC-Only also eliminates all timeouts (0 vs 32 in baseline B), confirming that compact context representation prevents the agent from stalling.

PLC-Only achieves the highest completion rate at 75.0% (+23.8pp over baseline B), suggesting that action blocking is highly effective when combined with the larger 131K context window. However, this comparison is confounded by the context window difference: condition C used 40K tokens while ablation variants used 131K. The $3\times$ smaller context budget likely disadvantages the full LASCon system in direct comparison.

Neither component reduces token usage; both ablation variants use more tokens than baselines (228K and 245K vs 164K for A). The extra tokens are spent productively, as evidenced by higher completion rates and more tool calls per task.

4.4 Loop Analysis

Loop-induced failures are rarer than initially hypothesized. In baseline A, only 2.5% of tasks (2/80) exhibit loop-induced failures. The loop-aware system prompt alone (condition B) eliminates all loop-induced failures, reducing the rate to 0%. This finding suggests that explicit prompting is sufficient to prevent most repetitive behaviors in Qwen3-32B.

Table 2: PLC safety analysis on Condition C (LASCon). The conservative parameters result in minimal blocking (0.12% block rate) with zero deadlock events.

Total Actions	Blocked	Block Rate (%)	Tasks w/ Blocking	Deadlocks	Max Consec. Blocks
806	1	0.12	1/79	0	1

PLC detected only 3 loop events across all 80 tasks in the PLC-Only condition, blocking actions in 2 tasks. With the large context window and loop-aware prompt, most repetitive patterns are avoided before PLC needs to intervene. This indicates that PLC serves primarily as a safety net rather than a primary mechanism for improvement.

4.5 PLC Safety Analysis

Table 2 presents PLC safety statistics for condition C. Across 806 proposed actions, PLC blocked only 1 action (0.12% block rate), affecting a single task. Zero deadlock events occurred, and the maximum consecutive blocks was 1, confirming that the conservative parameters (W=12, $L_{\max}$=3, R=3, M=10, T=20) prevent over-blocking while maintaining the safety net function.

5 Conclusion

We presented LASCon, a training-free scaffold for terminal agents that achieves +21.2pp completion improvement on Terminal-Bench 2.0 through structured context management. Our key finding is that deterministic CLI condensation (DCC) alone provides substantial gains (70.0% completion), outperforming LLM-based summarization by preserving exact tokens critical for CLI tasks. Loop-induced failures, while motivating our initial design, proved rare (2.5%) and addressable through prompting alone; PLC serves as a conservative safety net (0.12% block rate) rather than a primary mechanism.

Limitations. Docker unavailability prevented Pass@1 evaluation with automated test verification; completion rate serves as a proxy. We evaluated only Qwen3-32B; generalization to other models remains untested. The context window confound (40K vs 131K) between conditions limits direct comparison of component synergy. See Appendix A for implementation details and hyperparameters.

Future Work. Docker-enabled evaluation would provide definitive Pass@1 metrics. Extension to other agent domains (web, code editing) could validate the generality of deterministic condensation approaches.

References

Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. Understanding the planning of llm agents: A survey. *ArXiv*, abs/2402.02716, 2024.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. pp. 1658–1677, 2023.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *ArXiv*, abs/2310.06770, 2023.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A. Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. Acon: Optimizing context compression for long-horizon llm agents. *ArXiv*, abs/2510.00615, 2025.

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

Shukai Liu, Jian Yang, Bo Jiang, Yizhi Li, Jinyang Guo, Xianglong Liu, and Bryan Dai. Context as a tool: Context management for long-horizon swe-agents. *ArXiv*, abs/2512.22087, 2025.

Mike A. Merrill, Alexander G Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. K. Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, J. Jitsev, Di Lu, O. M. Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, E. Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert K. Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Rytting, Ryan Marten, Yixin Wang, Alexandros G. Dimakis, A. Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. 2026.

Noah Shinn, Federico Cassano, Beck Labash, A. Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. 2023.

Xingyao Wang, Simon Rosenberg, Juan Michelini, Calvin Smith, Hoang Tran, Engel Nyst, Rohit Malhotra, Xuhui Zhou, Valerie Chen, Robert Brennan, and Graham Neubig. The openhands software agent sdk: A composable and extensible foundation for production agents, 2025. URL `https://arxiv.org/abs/2511.03690`.

Yuhang Wang, Yuling Shi, Mo Yang, Rongrui Zhang, Shilin He, Heng Lian, Yuting Chen, Siyu Ye, Kai Cai, and Xiaodong Gu. Swe-pruner: Self-adaptive context pruning for coding agents. 2026.

Yuanan Xiao, Pengfei Gao, Chao Peng, and Yingfei Xiong. Improving the efficiency of llm agent systems through trajectory reduction. *ArXiv*, abs/2509.23586, 2025.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. Intercode: Standardizing and benchmarking interactive coding with execution feedback. *ArXiv*, abs/2306.14898, 2023.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Adriano Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *ArXiv*, abs/2405.15793, 2024a.

Qwen An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxin Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yi-Chao Zhang, Yunyang Wan, Yuqi Liu, Zeyu Cui, Zhenru Zhang, Zihan Qiu, Shanghaoran Quan, and Zekun Wang. Qwen2.5 technical report. *ArXiv*, abs/2412.15115, 2024b.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *ArXiv*, abs/2210.03629, 2022.

Yuxiang Zhang, Jiangming Shu, Ye Ma, Xueyuan Lin, Shangxi Wu, and Jitao Sang. Memory as action: Autonomous context curation for long-horizon agentic tasks. *ArXiv*, abs/2510.12635, 2025.

Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43:1 – 47, 2024.

Jialiang Zhu, Gongrui Zhang, Xiaolong Ma, Lin Xu, Miaosen Zhang, Ruiqi Yang, Song Wang, Kai Qiu, Zhirong Wu, Qi Dai, Ruichun Ma, Bei Liu, Yifan Yang, Chong Luo, Zhengyuan Yang, Linjie Li, Lijuan Wang, Weizhu Chen, Xin Geng, and Baining Guo. Re-trac: Recursive trajectory compression for deep search agents. 2026.

A Implementation Details

LASCon is implemented as a set of OpenHands SDK components. The Deterministic CLI Condenser (DCC) processes observations from `execute_bash` tool calls, extracting compact records using regex patterns for error detection. The scratchpad is updated after each action with the current working directory, recent commands, and files edited. Full raw outputs are saved to disk with unique identifiers for potential retrieval.

The Progress-Conditioned Loop Controller (PLC) maintains a sliding window of action-error-state triples. The fingerprinter computes SHA-256 hashes over the task workspace (`/testbed`) excluding common directories (`.git/`, `__pycache__/`, `node_modules/`). State version increments when the fingerprint changes between consecutive actions.

Hyperparameters. DCC extracts the first and last 10 lines of output. PLC uses window size W=12, pattern lengths $L \in \{1, 2, 3\}$, repeat threshold R=3, maximum blocklist size M=10, and block expiry T=20 steps. These parameters were chosen conservatively to minimize false positives.