# REDUCE: RISK-AWARE DUAL-LEVEL CONTEXT COMPRESSION FOR EFFICIENT TERMINAL AGENTS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

The performance of large language model (LLM) agents is increasingly constrained by the accumulation of verbose context. Existing context management methods typically treat rule-based observation filtering and model-based history compression as isolated stages, neglecting the cross-stage risk coupling: Aggressive filtering may remove evidence that downstream summarization cannot recover, whereas conservative filtering passes excessive history to the summarizer, increasing its cost and making it harder to preserve task-relevant details. To bridge this gap, we propose a training-free, self-evolving framework REDUCE, **R**isk-aware **D**ual-level Context **C**ompression for **E**fficient Terminal Agents. It features a risk-aware controller that dynamically governs rule activation, history retention, and compression thresholds by synthesizing online signals, such as context token pressure, summary grounding quality, and rule complaint feedback. Furthermore, it maintains dual timescale persistent knowledge: a global observation rule pool accumulates reusable filtering behaviors, while a statistical memory derives protective priors from historical validation outcomes. Experiments across multiple terminal-agent benchmarks and backbone models show that REDUCE reduces end to end token consumption by 16.1%–63.0% while maintaining or improving task accuracy.

## 1 INTRODUCTION

Large language model (LLM) agents have shown promising capabilities on extended software engineering tasks that require repeated interaction with terminals and repositories (Merrill et al., 2026; Zan et al., 2025). However, these interactions often generate large amounts of verbose context, which increases inference cost and makes important evidence harder to identify. Context management involves two related decisions: filtering individual terminal observations and compressing the interaction history accumulated over time. These decisions are coupled. Aggressive filtering may remove evidence that later summarization cannot recover, whereas conservative filtering increases the cost of history compression and makes it harder to retain important details. Effective context management should therefore coordinate both levels while balancing context usage and information preservation.

Existing methods usually focus on only one part of this process. General prompt compression methods reduce input length through truncation, pruning, or retrieval (Jiang et al., 2023; Pan et al., 2024). Agent oriented methods such as SWE-Pruner and TACO select task relevant information from individual terminal outputs (Wang et al., 2026; Ren et al., 2026), while methods such as ReSum, Context Folding, and AgentFold summarize interaction histories over multiple turns (Wu et al., 2026; Sun et al., 2026; Ye et al., 2026). ACON jointly compresses observations and interaction histories, but its two compression operations are triggered by separate fixed length thresholds and are not dynamically coordinated using cross-stage feedback (Kang et al., 2026). Existing approaches also lack persistent knowledge that can accumulate reusable filtering behaviors and risk estimates from previous validation results. These limitations motivate a unified framework that uses feedback to coordinate observation filtering and history compression over time.

Figure 1 illustrates this tension using a successful, uncompressed Terminal-Bench 2 trajectory (*custom-memory-heap-crash*, 17 steps). As shown in Figure 1(a), six near-identical Valgrind reruns account for 32.7% of the observation text, while the decisive evidence consists of only two



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.

lines buried in a 185-line patch dump at $t_3$ and is not reused until the final decision, thirteen steps later. Thus, a single trajectory can contain both substantial local redundancy and long-range evidence dependencies. As shown in Figure 1(b), the two compression levels are coupled through both information flow and context pressure: aggressive observation filtering may irreversibly remove evidence needed by later summarization, whereas conservative filtering leaves excessive history and reduces the space available for recent interactions. The appropriate compression intensity therefore depends on the current task and the state of both levels, rather than on fixed thresholds or independent decisions. This motivates a risk-aware controller that jointly adapts observation-rule activation, history retention, and compression thresholds using online signals from token pressure, summary quality, and filtering failures.

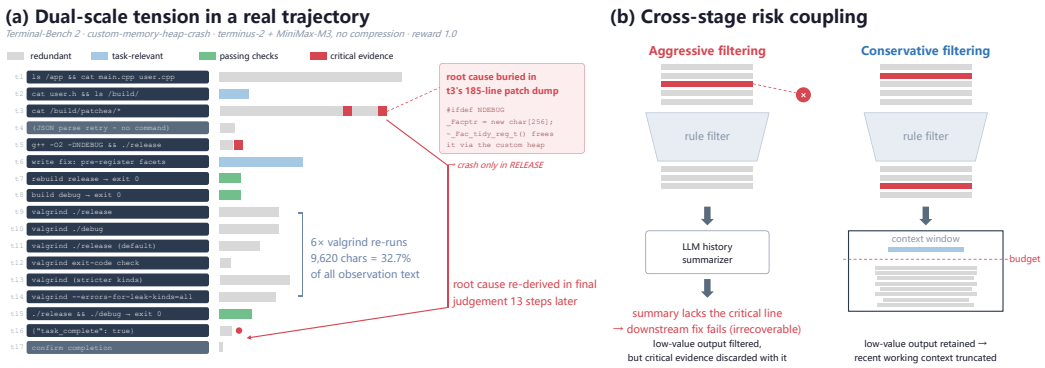



Figure 1: A successful Terminal-Bench 2 trajectory illustrating the coupled challenges of context compression. (a) Six near-identical Valgrind reruns account for 32.7% of the observation text, while the decisive evidence is a two-line fragment buried in a 185-line patch dump and reused thirteen steps later. (b) Aggressive observation filtering may cause irreversible evidence loss, whereas conservative filtering leaves less space for recent interactions. These contrasting risks motivate joint, state-dependent control of observation filtering and history compression.

To address these challenges, we propose **R**isk-aware **D**ual-level Context **C**ompression for **E**fficient Terminal Agents (REDUCE), a training-free framework. REDUCE combines a lineage-aware rule engine for filtering individual observations with a grounded progressive compressor for interaction histories. The history compressor uses structured summarization, source-grounded validation, deferred retry, and extractive fallback to reduce the risk of information loss. A risk-aware controller uses token pressure, summary-validation outcomes, and rule complaints to jointly adjust both compression levels. Across tasks, REDUCE maintains a persistent pool of observation rules and content-free history-compression statistics, enabling adaptive control within tasks and knowledge reuse across tasks.

The main contributions of this work are as follows:

- We propose REDUCE, a training-free framework for context compression in long-horizon terminal agents. REDUCE formulates context compression as a coupled, dual-level control problem, where irreversible observation filtering and history consolidation are jointly adjusted through shared risk-aware control.
- We design a continual compression mechanism that evolves observation rules using lineage and reliability feedback, while compressing histories through grounded progressive summarization, validation, deferred retry, and extractive fallback. A persistent dual-memory design further enables reusable observation rules and statistics of history compression without content to transfer across tasks.
- We evaluate REDUCE across multiple terminal-agent benchmarks and backbone models. Across all reported settings, REDUCE reduces end-to-end token usage by 16.1%–63.0% while achieving 1%–3% absolute accuracy gains over the baseline, demonstrating improvements in token efficiency, compression robustness, and cross-task transfer.

## 2 Problem Definition

Given a terminal task $x$, an agent performs a sequence of action–observation interactions,

$$\tau = (a_1, o_1, \ldots, a_T, o_T), \tag{1}$$

where $a_t$ is the command at turn $t$ and $o_t$ is the resulting observation. The uncompressed history is $H_t = \{(a_i, o_i)\}_{i<t}$, from which the agent selects the next action.

We consider two compression levels:

$$\widetilde{o}_t = F_{\text{obs}}(o_t; u_t^{\text{obs}}), \qquad \widetilde{H}_t = F_{\text{hist}}\left(\{(a_i, \widetilde{o}_i)\}_{i<t}; u_t^{\text{hist}}\right), \tag{2}$$

where $u_t^{\text{obs}}$ and $u_t^{\text{hist}}$ denote the compression intensities at the observation and history levels, respectively. The agent acts on the compressed context

$$\widetilde{C}_t = [x, \widetilde{H}_t]. \tag{3}$$

The two levels are sequentially coupled because history compression cannot recover information discarded from observations. Since tasks differ in redundancy, evidence dependencies, and token pressure, the two compression intensities should be selected jointly rather than by fixed independent rules. We therefore optimize

$$\max \ \mathbb{E}\left[R(\tau) - \lambda \sum_{t=1}^{T} \ell(\widetilde{C}_t)\right], \tag{4}$$

where $R(\tau)$ is the task reward, $\ell(\widetilde{C}_t)$ is the context length, and $\lambda$ controls the trade-off between task performance and context cost.

## 3 Method

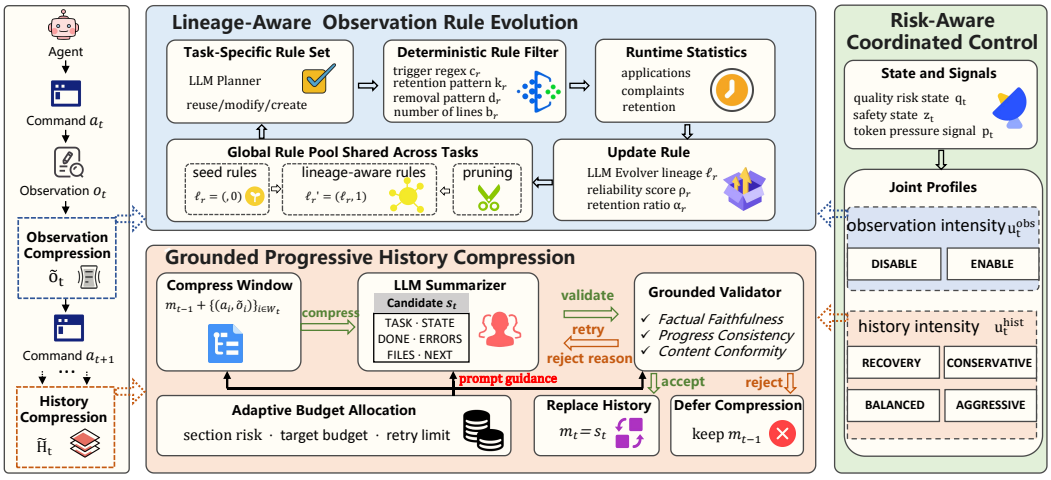



Figure 2: Overview of REDUCE. The framework coordinates three components: (1) lineage-aware observation rule evolution, which selects, applies, updates, and prunes reusable filtering rules; (2) grounded progressive history compression, which summarizes older interactions and validates candidate summaries before updating the accepted memory; (3) risk-aware coordinated control, which uses quality risk and token pressure to select joint operating profiles for observation filtering and history compression. The resulting feedback loop supports adaptive compression within a task and knowledge reuse across tasks.

REDUCE is a training-free framework for risk-aware, dual-level context compression with cross-task adaptation. As shown in Figure 2, it consists of three components: (1) **Lineage-Aware Observation Rule Evolution**, which filters terminal outputs using structured rules and continually refines

them across tasks; (2) **Grounded Progressive History Compression**, which incrementally summarizes older interactions while validating summary quality and retaining recent context; and (3) **Risk-Aware Coordinated Control**, which jointly adjusts observation filtering and history compression according to the pressure of tokens and the feedback on compression quality.

### 3.1 Lineage-Aware Observation Rule Evolution

The observation compressor $F_{\text{obs}}$ applies structured rules from a global pool to individual terminal outputs. The pool is initialized with six handcrafted seed rules and is progressively refined across tasks. Each rule is represented as

$$r = (c_r, k_r, d_r, b_r, s_r, \rho_r, \alpha_r, \ell_r), \tag{5}$$

where $c_r$ is the command trigger, $k_r$ and $d_r$ specify retained and removed line patterns, $b_r = (b^{\text{head}}, b^{\text{tail}}, b^{\text{cap}})$ contains the head, tail, and body budgets, $s_r$ is the rule summary, $\rho_r$ is its reliability, $\alpha_r$ is its retention ratio, and $\ell_r = (\text{parent}, \text{gen})$ records its lineage. A concrete example of the rule format is provided in Appendix A.1.1.

**Rule filtering.** At each turn, the controller determines whether observation filtering is enabled. If a selected rule matches the command and no error signal is present, a deterministic executor preserves the head and tail, filters the body according to $k_r$ and $d_r$, and truncates it to the rule budget:

$$\widetilde{o}_t = \text{head}_{b_r^{\text{head}}}(o_t) \oplus s_r \oplus \text{cap}_{b_r^{\text{cap}}}\left(\text{filter}\left(\text{body}(o_t); k_r, d_r\right)\right) \oplus \text{tail}_{b_r^{\text{tail}}}(o_t). \tag{6}$$

If no rule matches or an error signal is detected, the observation is passed through unchanged.

**Runtime statistics.** Rule reliability is updated from its application and complaint history:

$$\rho_r = \max\left(0,\ 1 - 2\frac{n_r^{\text{cmp}}}{n_r^{\text{app}}}\right), \qquad n_r^{\text{app}} > 0, \tag{7}$$

where $n_r^{\text{cmp}}$ and $n_r^{\text{app}}$ denote the numbers of complaints and applications. Rules with no applications are treated as unverified and are not ranked by reliability. The retention ratio $\alpha_r$ is the average value of $|\widetilde{o}_t|/|o_t|$ over the rule's applications. Together, $\rho_r$ and $\alpha_r$ measure the reliability and compression strength of a rule.

**Task-specific rule adaptation.** At task initialization, an LLM planner uses the task description, terminal state, and rule statistics to retrieve rules from the global pool and form a task-specific rule set. Unmodified reuse preserves a rule's identity, whereas changing its trigger or filtering configuration creates a child rule with the original rule as its parent and an incremented generation in $\ell_r$. New rules can also be created when the task has no suitable coverage.

During execution, complaints, requests for missing information, and repeated command executions serve as over-compression signals. When a rule becomes unreliable, an LLM generates a more conservative child rule based on the original rule and the observed feedback. The child then accumulates its own application and complaint statistics.

**Lineage-aware cross-task evolution.** After a task, task-specific rules and their statistics are merged into the shared pool. For a new rule without an explicit parent, the system links it to the most similar existing rule when their command triggers overlap sufficiently; otherwise, it starts a new lineage. As the pool grows, lineage-aware pruning removes untested or redundant variants while preserving seed rules, verified rules, and their relevant ancestors. The number of unapplied variants is also bounded to prevent untested rules from accumulating within a lineage. Thus, task-specific adaptation creates and tests rule variants, whereas cross-task evolution integrates, reuses, and consolidates them. An example is provided in Appendix A.1.2.

### 3.2 Grounded Progressive History Compression

The history compressor $F_{\text{hist}}$ complements $F_{\text{obs}}$ by summarizing accumulated interaction history. It retains recent rounds and compresses older history when the token count of the compressible portion exceeds a controller-selected soft threshold. Since compression may be repeated within a task, each generated summary is treated as a candidate and validated before replacing the accepted memory.

**Progressive structured memory.** Each summary follows a six-section schema covering the task objective, terminal state, completed commands, errors, relevant file paths, and the next action. At compression event $t$, the summarizer directly generates a structured candidate memory:

$$s_t = G\left(m_{t-1}, \{(a_i, \widetilde{o}_i)\}_{i \in W_t}\right), \tag{8}$$

where $G$ denotes the LLM-based summarizer, $m_{t-1}$ is the previously accepted memory, and $W_t$ denotes the older rounds selected for compression. The candidate has the form

$$s_t = (\textsc{task}, \textsc{state}, \textsc{done}, \textsc{errors}, \textsc{files}, \textsc{next}). \tag{9}$$

If $s_t$ passes validation, it becomes the new accepted memory, $m_t = s_t$. The agent then receives

$$\widetilde{H}_t = [m_t,\ \{(a_i, \widetilde{o}_i)\}_{i \notin W_t}], \tag{10}$$

which combines the accepted memory with recent uncompressed rounds.

**Grounded validation with deferred retry.** A candidate $s_t$ is accepted only if it passes three checks. *Factual faithfulness* verifies that important entities, such as paths, filenames, errors, and identifiers, are supported by the source rounds. *Progress consistency* checks that the summary does not deny executed commands or regress the accepted state. *Content conformity* rejects incomplete or non-content responses. The validator also checks schema completeness and the retention of task-specified output paths. Failures are associated with the affected memory sections.

If validation fails, the previous accepted memory and relevant uncompressed history remain in use. Compression is deferred, and the failure reasons are returned to the summarizer for a bounded number of retries. If all retries fail, compression is postponed until the hard token limit requires intervention, after which the system uses a deterministic extractive summary from the source rounds. This fallback introduces no LLM-generated content and therefore cannot fabricate facts. Deferred retries and fallback events are reported to the coordinator. A trace-level example is provided in Appendix B.

**Adaptive budget allocation.** Across tasks, the system stores content-free validation statistics grouped by compression stage and the number of required output paths. These statistics record validation outcomes, retries, fallbacks, and rejection reasons, and are converted into priors for protected sections and summary budgets. For a new task, the matching prior initializes section risks and the target budget. Rejections increase the risk and budget of affected sections, while successful summaries gradually restore their defaults. The resulting protected sections, per-section budgets, and retry limits are provided to the summarizer as explicit guidance.

### 3.3 Risk-Aware Coordinated Control

The coordinator jointly controls observation filtering $F_{\text{obs}}$ and history compression $F_{\text{hist}}$. At each step $t$, it selects an action from the following action space:

$$\begin{aligned} u_t^{\text{obs}} &\in \{\texttt{enable}, \texttt{disable}\}, \\ u_t^{\text{hist}} &\in \{\texttt{aggressive}, \texttt{balanced}, \texttt{conservative}, \texttt{recovery}\}. \end{aligned} \tag{11}$$

**State and signal.** The selection is based on the current token pressure $p_t$, quality-risk state $q_t$, and discrete safety state $z_t$. Let $N_t^{\text{old}}$ denote the token count of older history eligible for compression, and let $B^{\text{bal}}$ denote the `balanced` compression budget. We define normalized token pressure as

$$p_t = \frac{N_t^{\text{old}}}{B^{\text{bal}}}, \tag{12}$$

where larger values indicate a stronger need for compression. The quality-risk state summarizes recent memory-validation outcomes. Let $\sigma_t \in [0, \sigma_{\max}]$ denote the severity of the most recent validation outcome, with validated LLM summaries assigned the lowest severity and extractive fallbacks the highest severity. The risk state is updated as

$$q_t = \begin{cases} \max\{0, q_{t-1} - \delta_t^{\text{s}}\}, & \text{if the latest candidate is validated,} \\ \min\{1, \max(q_{t-1}, \sigma_t) + \delta_t^{\text{f}}\}, & \text{otherwise,} \end{cases} \tag{13}$$

where $\delta_t^{\text{s}}$ and $\delta_t^{\text{f}}$ are bounded non-decreasing updates in the streak lengths of consecutive successes and failures. The discrete safety state $z_t$ collects safety-relevant events, including severe validation failures, extractive fallback, and detected information loss.

**Joint profiles.** At the beginning of control interval $t$, the coordinator observes $(p_t, q_t, z_t)$. Let $\tau_p$ denote the token-pressure threshold, and let $\tau_q^{\text{low}} < \tau_q^{\text{high}}$ denote the lower and upper quality-risk thresholds. The coordinator selects the history profile according to the following priority:

1. If $z_t$ contains an emergency event, select `recovery`.
2. Otherwise, if $q_t \geq \tau_q^{\text{high}}$, select `conservative`.
3. Otherwise, if $p_t \geq \tau_p$, $q_t \leq \tau_q^{\text{low}}$, select `aggressive`.
4. Otherwise, select `balanced`.

The selected profile sets four hyperparameters of the compression mechanism. The recent-history window specifies the number of most recent rounds that remain uncompressed. The compression budget specifies the maximum number of older-history tokens allowed before compression is triggered. The cooldown period specifies the minimum interval between two compression attempts, and the retry budget specifies the maximum number of failed attempts tolerated before switching to extractive fallback.

The `balanced` profile uses the task-default values. The `aggressive` profile shortens the recent-history window and lowers the compression budget, thereby triggering compression earlier. The `conservative` profile enlarges the window and raises the budget, thereby preserving more recent history. The `recovery` profile further enlarges the window, extends the cooldown period, increases the retry budget, and disables observation filtering.

Observation filtering is enabled during normal operation and disabled during recovery:

$$u_t^{\text{obs}} = \begin{cases} \texttt{disable}, & z_t \text{ contains an emergency event,} \\ \texttt{enable}, & \text{otherwise.} \end{cases}$$

This priority ordering favors information preservation over token efficiency. Aggressive compression is therefore used only when token pressure is high, quality risk is low. Emergency events take precedence because history compression cannot reconstruct information removed by observation filtering.

# 4 Experiments

## 4.1 Experimental Setup

**Benchmarks.** To demonstrate both the efficacy and broad applicability of our approach, we evaluate its performance across diverse agentic backbones using TerminalBench 2.0 (Wang et al., 2025), while also extending our evaluation to multiple terminal benchmarks, including CompileBench and DevEval(Li et al., 2024).

**LLM and Agent Scaffolds.** We adopt Terminus-2 (Merrill et al., 2026) as the agent scaffold. Since its official implementation has been adapted to various terminal-related benchmarks, we use it as the baseline for evaluation on TerminalBench, CompileBench, and DevEval. We evaluate three representative LLMs as the underlying backbones: MiniMax-M3 (428B MoE)(MiniMax, 2026), Qwen3.8-27B, and Qwen3.5-9B(Qwen Team, 2026).

### 4.1.1 Performance and Efficiency

**Evaluation across Backbone Models.** As shown in Table 1, REDUCE achieves the highest accuracy across all three backbone models, improving over Baseline by 2.2, 3.3, and 3.4 percentage points, respectively. Meanwhile, its E2E token consumption is reduced by 63.0%, 16.1%, and 61.3%, respectively, decreasing from 374.5M to 180.1M tokens overall—a reduction of 51.9%. REDUCE also substantially reduces the peak context length, demonstrating its ability to improve task performance while effectively reducing inference overhead.

Table 1: Main results across backbone models on TerminalBench 2.0. Acc. denotes task accuracy, E2E Tok. includes input and output tokens from both the backbone agent and auxiliary compression models. Peak Ctx. denotes peak context length. ↑ and ↓ indicate that higher and lower values are better, respectively. respectively.

| Model | Method | Acc. (%) ↑ | E2E Tok. (M) ↓ | Peak Ctx. (K) ↓ |
|---|---|---|---|---|
| MiniMax M3 | Baseline | 50.6 | 159.215 | 362.9 |
| | TACO | 49.4 | 122.242 | 163.1 |
| | **REDUCE** | **52.8** | **58.918** | **124.9** |
| Qwen3.8-27B | Baseline | 41.6 | 83.664 | 130.6 |
| | TACO | 39.3 | 61.232 | 127.9 |
| | **REDUCE** | **44.9** | **70.204** | **125.3** |
| Qwen3.5-9B | Baseline | 19.1 | 131.629 | 116.0 |
| | TACO | 18.0 | 129.678 | 117.2 |
| | **REDUCE** | **22.5** | **50.955** | **33.8** |

Table 3: Component ablation on TerminalBench 2.0 with Qwen3.5-9B. Token usage includes both main-agent and compression-model tokens.

| Method | Acc. (%) ↑ | Main Tok. (M) | Comp. Tok. (M) | Total Tok. (M) ↓ |
|---|---|---|---|---|
| Baseline | 19.1 | 131.629 | – | 131.629 |
| Observation-only | 21.3 | 130.014 | 0.057 | 130.071 |
| History-only | **22.5** | 66.679 | 3.205 | 69.884 |
| Dual-Level w/o Coordination | 21.3 | 66.432 | 3.362 | 69.794 |
| **Full** | **22.5** | 46.698 | 4.256 | **50.954** |

**Evaluation across Datasets.** To evaluate the cross-benchmark generalization ability of REDUCE, we conduct experiments on TerminalBench 2.0, CompileBench, and DevEval using MiniMax-M3 as the backbone model. As shown in Table 2, REDUCE improves accuracy on TerminalBench 2.0 and DevEval by 2.2 and 1.6 percentage points, respectively, while preserving the original accuracy on CompileBench. At the same time, it reduces end-to-end token consumption by 63.0%, 25.4%, and 33.4% on the three benchmarks, respectively. These results demonstrate that REDUCE generalizes effectively across heterogeneous terminal-agent benchmarks while providing substantial inference-cost savings.

Table 2: Benchmark results.

| Benchmark | Acc. (%) ↑ | E2E Tok. (M) ↓ |
|---|---|---|
| *Baseline* | | |
| TB 2.0 | 50.56 | 159.21 |
| CompileBench | 86.7 | 7.980 |
| DevEval | 53.97 | 53.41 |
| *+REDUCE* | | |
| TB 2.0 | 52.8 (+2.2) | 58.91 (-63.0%) |
| CompileBench | 86.7 (+0.0) | 5.955 (-25.4%) |
| DevEval | 55.6 (+1.6) | 35.60 (-33.4%) |

### 4.2 Ablation Studies and Analysis

**Component Ablation.** We study compression at the observation and history levels, as well as the effect of coordinating the two, on TerminalBench 2.0 with Qwen3.5-9B. As shown in Table 3, history-level compression provides a strong efficiency–accuracy trade-off, improving accuracy from 19.1% to 22.5% while reducing total token usage by 47.0%. In contrast, observation-level compression alone yields a smaller accuracy improvement to 21.3% and reduces token usage by only 1.2%. Coordinating the two compression levels preserves the higher accuracy of history-level compression while further reducing total token usage to 50.955M, corresponding to a 61.3% reduction relative to the baseline and a 27.1% reduction relative to history-only compression. This result demonstrates that coordination is essential for realizing the full efficiency benefits of combining the two compression levels.

**Evolution of Compression Knowledge** Figure 3 visualizes the compression knowledge accumulated by Qwen3.5-9B over 17 rounds on TerminalBench 2.0. The observation-rule memory grows through the creation, revision, and repeated reuse of rule lineages, with several lineages receiv-

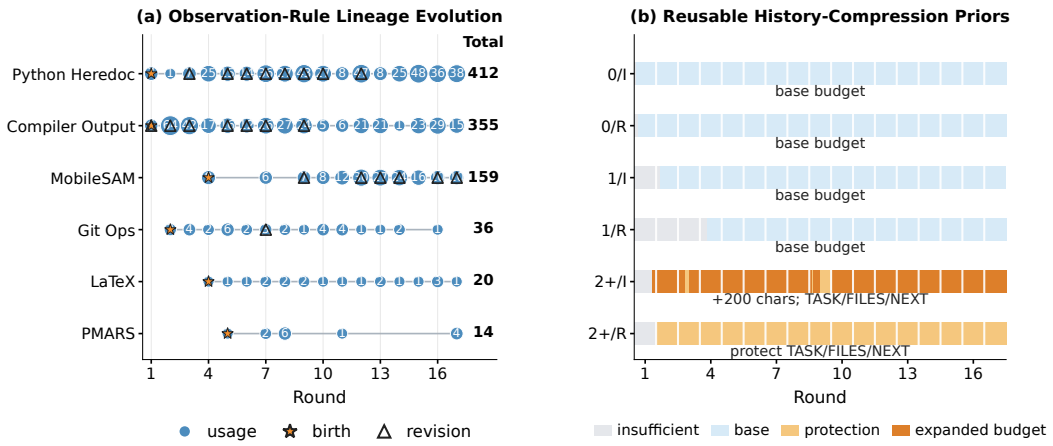



Figure 3: Compression knowledge accumulated by Qwen3.5-9B over 17 rounds on TerminalBench 2.0. (a) Observation-rule lineage evolution. The six most frequently used lineages are shown; circles indicate usage, stars creation, triangles revision, and rightmost values total applications. (b) Reusable history-compression priors. 0, 1, and 2+ denote the number of required output paths, while I and R denote initial and recursive compression. Colors indicate compression-budget states.

Table 4: Cross-task transfer of dual compression memory from TerminalBench 2.0 to DevEval. All settings use the full dual-level architecture without target-time updates. E2E Tok. includes backbone-agent, rule-evolution, and history-compression tokens.

| Setting | Observation Memory | History Memory | Acc. (%) ↑ | E2E Tok. (M) ↓ |
|---|---|---|---|---|
| Cold start | Seed rules | Default policy | 14.3 | 28.446 |
| Rule-only transfer | Warm rule pool | Default policy | 15.9 | 25.263 |
| History-only transfer | Seed rules | Warm history prior | 15.9 | 30.183 |
| Dual-memory transfer | Warm rule pool | Warm history prior | 17.5 | 29.363 |

ing hundreds of applications. Meanwhile, the history-compression memory progressively activates reusable policies and refines their compression budgets. In particular, multi-path outputs trigger stronger budget allocation and section protection, indicating that the learned policies adapt their compression behavior to the structural demands of the task. Overall, the figure shows that REDUCE accumulates reusable compression knowledge at both the observation and history levels rather than relying solely on task-specific decisions.

**Cross-Task Memory Transfer.** We evaluate cross-task transfer using Qwen3.5-9B, where the memories are first accumulated over 17 rounds on TerminalBench 2.0 and then transferred to DevEval without target-time updates. As shown in Table 4, transferring either the rule memory or the history memory improves accuracy from 14.3% to 15.9%. Combining both memories achieves the best accuracy of 17.5%, a 3.2-point gain over the cold start setting, while using only 29.363M tokens. These results suggest that the two memory types provide complementary benefits for cross-task transfer.

## 5 Related Work

### 5.1 Context Compression for Long-Horizon Agents

Code agents solve repository-level and terminal-based software engineering tasks through long command–observation trajectories (Jimenez et al., 2024; Zan et al., 2025; Merrill et al., 2026; Yang et al., 2024; Wang et al., 2024). These trajectories contain verbose outputs as well as critical evidence, including file paths, error messages, and test results.General prompt compression reduces input length at the token or sentence level (Jiang et al., 2023; Pan et al., 2024). Agent-oriented

methods operate more directly on interaction data: SWE-Pruner and AGORA select task-relevant context, Context Folding compresses histories, ContextTool manages context through an agent tool, and TACO learns terminal-output filtering rules (Wang et al., 2026; Zhang & Sun, 2026; Sun et al., 2026; Liu et al., 2026; Ren et al., 2026). These methods generally treat observation filtering and history compression separately. However, aggressive filtering may discard evidence that later summarization cannot recover, while conservative filtering increases cost and may obscure task-relevant details. REDUCE jointly controls both levels with shared risk signals.

### 5.2 Self-Evolving Agents and Persistent Compression Knowledge

Self-evolving agents improve through accumulated experience. Some methods update model parameters through reinforcement learning or fine-tuning (Xia et al., 2025; Jiang et al., 2026; Xia et al., 2026), while others maintain external reflections, skills, or policies (Zhou et al., 2024; Liang et al., 2025; Zhou et al., 2026). The latter avoids retraining, but the accumulated knowledge may become redundant, inconsistent, and expensive to use. Recent work has also explored learning compression policies. ACON learns natural-language compression guidelines offline and keeps them fixed during deployment, while TACO updates a collection of terminal filtering rules during task execution (Kang et al., 2026; Ren et al., 2026). These methods do not consider how persistent knowledge can support both observation filtering and history compression. REDUCE formulates context compression as a continual adaptation problem with two interacting components. It updates both compression levels and manages their accumulated knowledge through rule lineage, consolidation and statistics that do not contain task-specific text.

## 6 Limitations.

REDUCE is designed for terminal-agent scenarios and its components—including the line-level observation rules and the six-section structured memory schema—are tailored to the characteristics of command-line interactions. Direct application to other modalities (e.g., web navigation, API interactions) may require adaptation. The grounded validation mechanism provides robustness against information loss but introduces extra LLM calls per compression event; while this overhead is offset by token savings in long-horizon tasks, it may become a bottleneck in shorter episodes. Finally, REDUCE assumes synchronous single-agent execution; extending it to asynchronous or multi-agent settings would demand changes to the compression trigger and state management logic.

## 7 Conclusion

We presented REDUCE, a dual-level compression framework for long-horizon terminal agents. REDUCE combines observation-level rule evolution with history-level compression and coordinates the two mechanisms through reusable compression memories. These memories accumulate compression knowledge across tasks and can be transferred to unseen benchmarks without target-time updates. Experiments across multiple backbone models and terminal-agent benchmarks show that REDUCE consistently maintains or improves task accuracy while substantially reducing end-to-end token consumption and peak context length.

## Ethics Statement

This work uses no human subjects, personal data, or crowdsourcing. All benchmarks (Terminal-Bench 2.0, CompileBench, DevEval) are publicly released research artifacts used under their licenses. We mitigate dual-use risk by only releasing compression code and configuration files (no model weights or executable payloads) and by noting that REDUCE improves efficiency without expanding the underlying agent's capabilities. All experiments run inside the benchmark-provided sandboxed Docker images with no external network access and no persistent state across tasks. The summarizer, rule planner, and rule evolver share the same temperature setting as the main agent; every LLM-generated summary is verified by a deterministic grounded validator before replacing the accepted memory. We report token consumption alongside accuracy so that downstream users can estimate inference energy use.

## AI USE DISCLOSURE

OpenAI ChatGPT was used during the preparation of this work to assist with software implementation, code debugging, code editing, and improving the grammar and readability of the manuscript. The core research idea, problem formulation, methodological framework, and overall organization of the paper were developed by the authors.

OpenAI ChatGPT was not used to generate synthetic datasets, formulate or prove mathematical claims, propose the main hypotheses, design the core methodology, or interpret the experimental results. All AI-assisted code was inspected, tested, and verified by the authors, and all AI-assisted text was reviewed and revised by the authors. We take full responsibility for the final content of this work, including its claims, code, analyses, and conclusions.

## REPRODUCIBILITY STATEMENT

We will release the full implementation, all configuration files, the final cross-task rule cache, and the history-compression priors at a public repository upon acceptance. Our experiments use publicly released benchmark artifacts (TerminalBench 2.0, CompileBench, DevEval) and three backbones: MiniMax-M3, Qwen3.8-27B, and Qwen3.5-9B. Model identifiers, scaffold version, and benchmark Docker image tags will be pinned in the released repository. The compression framework follows the algorithm described in §3. We do not pin random seeds; reported numbers are single-pass values and a fresh run may show small deviations. Token consumption is measured end-to-end on the full task suite of each benchmark and includes both backbone-agent and compression-model tokens. For double-blind review, the implementation is not currently publicly available. Reviewers may request access to the code and intermediate artifacts via the OpenReview discussion channel.

## REFERENCES

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 13358–13376. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.825. URL https://aclanthology.org/2023.emnlp-main.825/.

Minqi Jiang, Andrei Lupu, and Yoram Bachrach. Bootstrapping task spaces for self-improvement. *Transactions on Machine Learning Research*, 2026. URL https://openreview.net/forum?id=k2VsgUxC6X.

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? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

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, 2026. URL https://openreview.net/forum?id=7JbSwX6bNL.

Jia Li, Ge Li, Yunfei Zhao, Yongmin Li, Huanyu Liu, Hao Zhu, Lecheng Wang, Kaibo Liu, Zheng Fang, Lanshen Wang, Jiazheng Ding, Xuanming Zhang, Yuqi Zhu, Yihong Dong, Zhi Jin, Binhua Li, Fei Huang, Yongbin Li, Bin Gu, and Mengfei Yang. Deveval: A manually-annotated code generation benchmark aligned with real-world code repositories, 2024. URL https://arxiv.org/abs/2405.13971.

Xuechen Liang, Meiling Tao, Yinghui Xia, Jianhui Wang, Kun Li, Yijin Wang, Yangfan He, Jingsong Yang, Tianyu Shi, Yuantao Wang, Miao Zhang, and Xueqian Wang. SAGE: Self-evolving agents with reflective and memory-augmented abilities. *Neurocomputing*, 647:130470, 2025. doi: 10.1016/j.neucom.2025.130470. URL https://doi.org/10.1016/j.neucom.2025.130470.

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. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.1032. URL https://aclanthology.org/2026.findings-acl.1032/.

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. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=a7Qa4CcHak.

MiniMax. MiniMax-M3. https://huggingface.co/MiniMaxAI/MiniMax-M3, 2026. Model card.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. LLMLingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 963–981. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.57. URL https://aclanthology.org/2024.findings-acl.57/.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Jincheng Ren, Siwei Wu, Yizhi Li, Kang Zhu, Shu Xu, Boyu Feng, Ruibin Yuan, Wei Zhang, Riza Batista-Navarro, Jian Yang, and Chenghua Lin. A self-evolving framework for efficient terminal agents via observational context compression, 2026. URL https://arxiv.org/abs/2604.19572.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. Scaling long-horizon agent via context folding, 2026. URL https://openreview.net/forum?id=JaLXQnA2wi.

Feng Wang, Dong Chen, Shaoxin Lin, Muhan Zeng, Guanzheng Chen, Xin Liu, Sai Zhang, Bo Duan, Zijie Zhao, Linchao Zhu, Lijun Wu, Jie Tang, and Enhong Chen. Terminal-bench 2.0: From command accuracy to agentic problem solving, 2025. URL https://arxiv.org/abs/2510.13769.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024. URL https://arxiv.org/abs/2407.16741.

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. URL https://arxiv.org/abs/2601.16746.

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Ding-Chu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, Minhao Cheng, Shuai Wang, Hong Cheng, and Jingren Zhou. Resum: Unlocking long-horizon search intelligence via context summarization, 2026. URL https://openreview.net/forum?id=PjIK38mwKm.

Peng Xia, Kaide Zeng, Jiaqi Liu, Can Qin, Fang Wu, Yiyang Zhou, Caiming Xiong, and Huaxiu Yao. Agent0: Unleashing self-evolving agents from zero data via tool-integrated reasoning, 2025. URL https://arxiv.org/abs/2511.16043.

Peng Xia, Jianwen Chen, Xinyu Yang, Haoqin Tu, Jiaqi Liu, Kaiwen Xiong, Siwei Han, Shi Qiu, Haonian Ji, Yuyin Zhou, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. Metaclaw: Just talk an agent that meta-learns and evolves in the wild. *arXiv preprint arXiv:2603.17187*, 2026.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652, 2024. URL https://papers.nips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, Pengjun Xie, Fei Huang, Jingren Zhou, Siheng Chen, and Yong Jiang. Agentfold: Long-horizon web agents with proactive context folding. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=IuZoTgsUws.

Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Shulin Xin, Linhao Zhang, Qi Liu, Aoyan Li, Lu Chen, Xiaojian Zhong, Siyao Liu, Yongsheng Xiao, Liangqiang Chen, Yuyu Zhang, Jing Su, Tianyu Liu, Rui Long, Ming Ding, and Liang Xiang. Multi-SWE-Bench: A multilingual benchmark for issue resolving. In *Advances in Neural Information Processing Systems*, 2025. URL https://mlanthology.org/neurips/2025/zan2025neurips-multiswebench/.

Haoran Zhang and Zhaohua Sun. AGORA: Adapter-grounded observation-action retention for inference-free prompt compression in LLM agents, 2026. URL https://arxiv.org/abs/2605.26596.

Huichi Zhou, Siyuan Guo, Anjie Liu, Zhongwei Yu, Ziqin Gong, Bowen Zhao, Zhixun Chen, Menglong Zhang, Yihang Chen, Jinsong Li, et al. Memento-skills: Let agents design agents. *arXiv preprint arXiv:2603.18743*, 2026.

Wangchunshu Zhou, Yixin Ou, Shengwei Ding, Long Li, Jialong Wu, Tiannan Wang, Jiamin Chen, Shuai Wang, Xiaohua Xu, Ningyu Zhang, Huajun Chen, and Yuchen Eleanor Jiang. Symbolic learning enables self-evolving agents, 2024. URL https://arxiv.org/abs/2406.18532.