# A Self-Evolving Framework for Efficient Terminal Agents via Observational Context Compression

**Jincheng Ren**[1,2,9*] **Siwei Wu**[1,8* †] **Yizhi Li**[1,8*] **Kang Zhu**[2] **Shu Xu**[3] **Boyu Feng**[2] **Ruibin Yuan**[4]
**Wei Zhang**[5] **Riza Batista-Navarro**[1] **Min Gao**[6] **Yan Bai**[7] **Jian Yang**[5 ‡] **Chenghua Lin**[1 ‡]
[1]University of Manchester [2]MAP [3]HKUST(GZ) [4]HKUST
[5]Beihang University [6]Chongqing University [7]Peking University [8]iQuest [9]Hohai University

## Abstract

Terminal observations are not ordinary long-context text: they are heterogeneous, low-information-density execution traces in which sparse but exact evidence (e.g., error messages and file paths) is interleaved with large amounts of redundant terminal output. For long-horizon CLI agents, retaining raw observations rapidly increases context cost and can dilute critical signals, while LLM-based summarization or fixed heuristics often fail to adapt across heterogeneous terminal tasks and may discard precise task-relevant evidence. We propose TACO, the first self-evolving **T**erminal **A**gent **C**ompressi**o**n framework, which treats compression rules as reusable, preservation-aware knowledge acquired from interaction trajectories. Rather than relying on manually designed rules, static pruning, or task-specific compressor training, TACO autonomously discovers, refines, and reuses structured compression rules from agent interaction trajectories. Our global rule pool mechanism accumulates effective rules across tasks, while intra-task rule evolution adapts them online to the current workflow. This allows TACO to filter redundant terminal observations while preserving exact task-relevant evidence. Across **six** benchmarks, including **TB 1.0**, **TB 2.0**, **SWE-Bench Lite**, **CompileBench**, **DevEval**, and **CRUST-Bench**, TACO consistently maintains or improves task success across models and agent scaffolds. On TerminalBench, TACO yields **1%–4%** absolute accuracy gains under standard evaluation and improves accuracy by **2%–3%** under matched token budgets. On downstream benchmarks, TACO can reduce total token consumption by 12%–27% while maintaining or improving task success, with model-dependent efficiency gains. These results show that self-evolving observation compression can unlock latent capability in existing CLI agents by allocating context budget toward task-relevant evidence, without model fine-tuning or human-crafted compression rules. The code for TACO is available at `https://github.com/multimodal-art-projection/TACO.git`.

## 1 Introduction

Recent advances in code foundation models, code intelligence, and agentic code systems have enabled increasingly capable software-engineering agents [Yang et al., 2025, 2026b,c,a], yet terminal-centric tasks such as repository debugging, compilation, testing, and environment interaction remain challenging. Unlike short-form code generation, CLI agents operate through long-horizon interaction loops: they execute commands, observe terminal outputs, and condition future actions on accumulated environmental feedback [Merrill et al., 2026, Jimenez et al., 2024]. As these histories grow,

*Equal contribution
†Project leader ( Conceptualization, technical guidance, and lead writing)
‡Corresponding authors

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

mainstream terminal agents often preserve accumulated raw command outputs to avoid losing useful feedback, creating a central bottleneck for both efficiency and reliability.

Our trajectory analysis shows that this bottleneck is not merely caused by long context, but by the highly uneven information density of terminal observations. As shown in Fig. 1, manually extracting effective text from 50 sampled TB 2.0 trajectories removes 24.6%–44.1% of raw prompt tokens across the evaluated models, indicating that a substantial portion of terminal-agent histories is occupied by low-value content rather than actionable information. However, this redundancy is not cleanly separable from useful content: verbose logs, installation traces, build outputs, and test reports may contain exact evidence required for future decisions, including error messages, file paths, failing test names, command arguments, and build targets. This suggests that terminal-observation compression should not simply maximize shortening; it must perform preservation-aware filtering, selectively removing low-value terminal output while preserving the exact evidence needed for future actions.

**Existing compression methods are not designed for preservation-aware terminal observation filtering.** Generic LLM summarization is flexible, but it may paraphrase, omit, or blur exact terminal signals, which can be harmful when later actions depend on precise strings such as error messages, file paths, or test names. Static heuristics and expert-crafted rules can preserve such signals when carefully designed, but they are brittle across commands, repositories, languages, and task domains, and require substantial manual engineering. Training-based methods such as SWE-Pruner [Wang et al., 2026] provide more adaptive pruning, but they require additional training data and are mainly optimized for SWE-Bench-style software-engineering workflows, limiting their applicability to broader terminal environments. These limitations raise a natural question: *can a terminal agent autonomously discover reusable compression knowledge from its own interaction trajectories, without human-crafted rules or compressor training?*

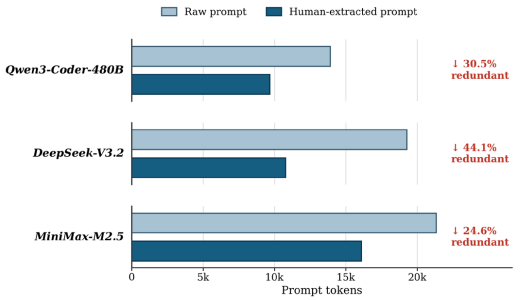


Figure 1: Effective-text extraction reveals substantial removable redundancy in terminal-agent histories on TB 2.0.

To address this problem, we propose TACO, a plug-and-play, unsupervised test-time adaptation **T**erminal **A**gent **C**ompressi**o**n framework. TACO reframes terminal-observation compression from a one-shot shortening operation into a continual knowledge-acquisition process over terminal-output patterns. Instead of treating compression rules as fixed heuristics or task-specific parameters, TACO treats them as reusable preservation-aware knowledge that can be discovered, refined, and transferred across workflows. During execution, an LLM proposes structured rules for observation patterns, and a conservative executor applies them only when their triggers match while preserving critical evidence by design. Within each task, TACO adapts the active rule set using implicit feedback from the agent, such as requests for full output or repeated commands indicating over-compression. Across tasks, effective rules are accumulated, scored, and reused through a Global Rule Pool, allowing compression knowledge discovered in one workflow to benefit subsequent tasks.

We integrate TACO into mainstream agent frameworks and evaluate it across multiple terminal-related benchmarks with strong backbone models. In summary, our main contributions are as follows:

1. **We introduce self-evolving terminal-observation compression for CLI agents.** We formulate terminal-agent context compression as a self-evolving rule-learning problem, where terminal-output patterns are converted into reusable, preservation-aware compression rules. TACO realizes this idea as a **plug-and-play**, **unsupervised test-time adaptation**, and **human-free** framework that autonomously discovers, refines, and reuses structured rules from agent interaction trajectories, without task-specific compressor training or human-crafted rules.
2. **TACO improves the accuracy–token trade-off on TerminalBench.** On TerminalBench 1.0 and 2.0, integrating TACO into existing agent frameworks yields consistent **1%–4%** absolute accuracy gains across strong backbone models. Under matched token budgets, TACO further improves accuracy by around **2%–3%**, showing that it helps agents use the same context budget more effectively.

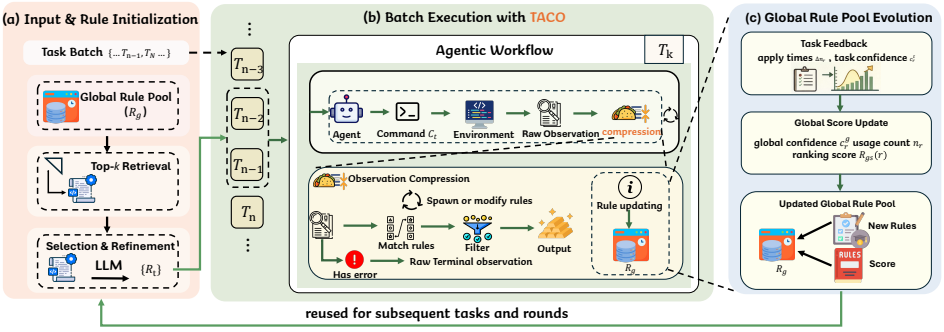


Figure 2: Overview of TACO.

3. **TACO adapts across heterogeneous terminal benchmarks.** Beyond TerminalBench, we evaluate TACO on SWE-Bench Lite, CompileBench, DevEval, and CRUST-Bench, where it maintains or improves task success while reducing total token consumption by **12%–27%**. These results show that TACO can be directly integrated into agents for diverse terminal-related benchmarks and automatically adapt its compression rules to different terminal tasks, without task-specific training or manually designed rules.

## 2 Self-evolving Compression Agentic Framework

During a terminal agent's execution loop, each step receives an environment observation that often contains substantial redundancy. To improve token efficiency, we propose TACO, a self-evolving context compression framework that compresses terminal observations with a dynamically evolving set of compression rules. Each rule specifies when and how an observation should be compressed. As shown in Fig. 2, TACO consists of three components: (1) **Terminal Observation Compression** (top right), which applies evolved rules at each step; (2) **Intra-Task Rule Set Evolution** (middle right), which updates rules online from compression outcomes within the current task; and (3) **Global Rule Pool Evolution** (bottom right), which refines and shares effective rules across tasks.

### 2.1 Task Definition

Given a terminal task $T$, an agent $A$ iteratively interacts with the environment. At each step $t$, the agent generates a command $C_{t+1} = A(T, S_t)$ conditioned on the history sequence $S_t$ of preceding command-observation pairs, and receives a raw terminal observation $O_{t+1}$. This repeats until task completion. For efficiency, we process $N$ tasks in parallel, where $N$ is the batch size.

### 2.2 TACO Adapter

**TACO** is a plug-and-play terminal observation compression adapter that can be integrated into different terminal-agent scaffolds. Specifically, after the host agent executes a command $C_t$ and receives the raw terminal observation $O_t$, TACO returns a compressed observation $\tilde{O}_t$ according to the task-specific rule set $R_t$. The detailed compression process is described in Sec. 2.4.

### 2.3 Rule Initialization

**Rule Definition.** A rule consists of an applicability condition together with compression parameters, such as trigger patterns, retained patterns, removed patterns, and conservative retention bounds. During execution, these structured rules are instantiated by a fixed rule executor into concrete filtering behavior. This design constrains self-evolution to a safe and reusable rule space, enabling TACO to remove low-value noise while preserving critical information. Detailed rule schema and execution examples are provided in Appendix L.

**Global Rule Pool.** Since many compression patterns discovered during task execution are reusable across tasks, such as compressing `pip install` progress in package-installation scenarios, we maintain a Global Rule Pool $R_g$ that accumulates structured compression rules, with each rule

$r \in R_g$ initialized with a global confidence score $c_r^g = 1.0$ for tracking reliability and a ranking score $R_{gs}(r)$ for global ranking and task-level retrieval. After a task is completed, effective rules are selectively written back to the pool, with their global scores initialized or updated based strictly on their empirical performance during the task.

**Task-Level Rule Selection.** Different tasks may require distinct compression behaviors. We first retrieve the top-$k$ rules from $R_g$ according to their ranking scores $R_{gs}(r)$ as candidates. Given the task description and objective, an LLM then selects and refines these candidates to better align with the current context. This task-conditioned selection prevents high-ranked but irrelevant rules from being activated: candidates whose triggers do not match the current task are filtered out regardless of their $R_{gs}(r)$ values. The resulting rules initialize the task-specific active rule set $R_t$. Additional implementation details are provided in Appendix N.

### 2.4 Terminal Observation Compression

At each interaction step, the agent executes a terminal command and receives a raw terminal observation $O_t$. Observations containing explicit error or failure signals, such as syntax errors and exception traces, are treated as **Critical**, as they often inform subsequent decisions. TACO leaves such outputs unchanged and applies the active task-specific rules in $R_t$ only to non-Critical outputs:

$$\tilde{O}_t = \begin{cases} O_t, & \text{if } O_t \text{ is } \textbf{Critical}, \\ F_{R_t}(O_t \mid C_t), & \text{otherwise}, \end{cases} \tag{1}$$

where $F_{R_t}$ denotes a conservative rule-based compression operator induced by the active rule set $R_t$.

### 2.5 Intra-Task Rule Set Evolution

When solving each task, TACO updates the task-specific rule set dynamically.

**Adding Rules** When a terminal observation is not handled by any rule in $R_t$, TACO treats it as an uncovered output and invokes an LLM to generate a new rule $r$, which is then added to $R_t$ for subsequent steps.

**Updating Rules** TACO updates rules based on implicit feedback from the agent's behavior after consuming the compressed observation $\tilde{O}_t$. Behaviors such as requesting the full output, indicating missing critical details, or repeating the same command to recover more information are treated as over-compression complaints. In response, TACO traces back the rules that triggered and modified the current observation, suppresses their use for subsequent steps, and injects more conservative replacement variants via LLMs. The corresponding prompt is provided in Appendix N.

### 2.6 Global Rule Pool Evolution

To accumulate reusable compression knowledge across tasks, TACO writes effective task-level rules back to the Global Rule Pool and updates their global statistics according to their observed behavior in the completed task.

**Global Rule Pool Updating.** After completing a task, TACO updates the Global Rule Pool based on task-level evidence collected from $R_t$. For each rule $r \in R_t$, we record two quantities: $\Delta n_r$, the number of successful applications of $r$ in the current task, and $c_r^t \in [0, 1]$, its final task-level confidence. A rule is written back to $R_g$ only if it has been successfully applied at least once and remains reliable, i.e., $\Delta n_r \geq 1$ and $c_r^t \geq \tau$, where $\tau$ is a small confidence threshold. Rules that receive explicit complaints are assigned $c_r^t = 0$ for the current task and are therefore excluded from write-back. If such a rule already exists in $R_g$, its global confidence $c_r^g$ is further decayed, lowering its ranking score and reducing its chance of being retrieved in subsequent tasks.

**Global Rule Score Updating.** For a newly discovered effective rule, TACO initializes its global confidence and usage statistics directly from the current task. For an existing effective rule, TACO updates its global confidence $c_r^g$ using the task-end confidence $c_r^t$ and updates its cumulative usage count with the newly observed successful applications.

Each rule $r$ in the Global Rule Pool is assigned a ranking score:

$$R_{gs}(r) = c_r^g \cdot (n_r + 1), \tag{2}$$

where $c_r^g$ denotes its global confidence and $n_r$ denotes its cumulative number of successful applications across tasks. This score prioritizes rules that are both reliable and broadly reusable: complaints reduce $c_r^g$ and thus down-rank problematic rules, while frequent successful use increases $n_r$. Newly generated rules are initialized with $c_r^g = 1.0$, so they are not prematurely excluded before accumulating usage evidence. We use this score only to retrieve the top-$k$ candidates, after which the LLM performs task-conditioned selection to decide which rules to activate.

**Multi-round Evolution and Batch Size.** To obtain broader and more transferable compression rules, TACO performs self-evolution across tasks rather than relying only on evolution within a single task. Moreover, we run **multiple evolution rounds** on the same dataset until convergence is reached. Under parallel execution, effective rules from completed tasks are written back to the Global Rule Pool and reused to initialize later tasks. As a result, the batch size $N$ affects the speed of rule propagation and can influence final performance, which we analyze in the Appendix G.

### 2.7 Reward-Free Unsupervised Rule Evolution Protocol

**Algorithm Convergence.** Because our self-evolving method continuously acquires new rules, the update dynamics of the Global Rule Pool provide a natural convergence signal. Therefore, we use the change rate of the Top-$K$ rules in the Global Rule Pool to assess convergence. Specifically, we define a **reward-free** metric, *Retention*, which measures the proportion of rules that remain in the Top-$K$ after one round of evolution on the dataset. It is calculated as follows:

$$\text{Retention}_K^{(i)} = \frac{\left|\text{TopK}\left(R_g^{(i-1)}\right) \cap \text{TopK}\left(R_g^{(i)}\right)\right|}{K} \times 100\%, \tag{3}$$

where $R_g^{(i)}$ is the Global Rule Pool after the $i$-th run. A higher Retention value indicates that the effective rule frontier has become more stable. In this work, we set $K = 30$. Additional discussion on the choice of $K$ is provided in the Appendix G.

**Reward-Free Rule Evolution and Leakage Prevention.** As detailed in Appendix I's Tab. 11, TACO evolves compression rules solely from terminal observations, without accessing benchmark answers, hidden tests, verifier outcomes, task success labels, or leaderboard feedback. Rule updates depend only on observation-level signals, including rule triggers, application counts, terminal observation, and subsequent interactions that indicate possible over-compression. The stopping criterion is also **reward-free**: evolution terminates when the metric *Retention* stabilizes.

## 3 Experiment Setup

### 3.1 Benchmarks

To validate the effectiveness and generality of TACO, we not only evaluate its performance gains with various agentic backbones on TerminalBench (TB 1.0 and TB 2.0), but also conduct experiments on a range of terminal-related benchmarks (i.e., SWE-Bench Lite [Jimenez et al., 2024], CompileBench, DevEval [Li et al., 2024], CRUST-Bench [Khatry et al., 2025]).

### 3.2 LLMs and Agent Scaffolds

To evaluate the generality of TACO across both model families and agent implementations, we instantiate our method with multiple backbone LLMs and evaluate it on different agent scaffolds.

**LLMs.** We consider closed-source and open-source models. For closed-source models, we evaluate the Claude, GPT, and Gemini series, among the strongest in coding capability. For open-source models, beyond state-of-the-art models on terminal tasks such as MiniMax, DeepSeek [Liu et al., 2025a], GLM [Zeng et al., 2025], Kimi-K2-Instruct [Bai et al., 2025], and Nex-N1 [Cai et al., 2025] series, we evaluate several Qwen3-series models with fewer than 40B parameters.

Table 2: Results on TerminalBench (TB) 1.0 and 2.0. Shaded cells in the TACO block report gains over the corresponding Terminus-2 baseline using the same backbone.

| Model | Model Size | Agent Scaffold | TB 1.0 | TB 2.0 |
|---|---|---|---|---|
| *Open-Source Models (>200B)* | | | | |
| GLM-4.7 | 358B | Terminus-2 | 48.75 | 41.00* |
| MiniMax-M2.5 | 230B | Terminus-2 | 42.30 | 42.80 |
| MiniMax-M2.1 | 229B | Terminus-2 | 42.50 | 29.20* |
| DeepSeek-V3.2 | 685B | Terminus-2 | 43.93 | 40.62 |
| DeepSeek-V3.1-Nex-N1 | 685B | OpenHands | 31.56 | 31.80† |
| Kimi-K2-Instruct | 1T | Terminus-2 | 44.59 | 27.80* |
| Qwen3-Coder-480B | 480B | Terminus-2 | 37.50 | 23.90* |
| Qwen3-235B-A22B-Instruct | 235B | Terminus-2 | 15.00 | 13.50 |
| *Open-Source Models (∼30B and below)* | | | | |
| Qwen3-Coder-30B-A3B-Instruct | 30B | Terminus-2 | 23.80 | 14.60 |
| Qwen3-30B-A3B-Nex-N1 | 30B | OpenHands | 25.00 | 8.30† |
| Qwen3-32B-Instruct | 32B | OpenHands | 11.25 | 3.40 |
| Qwen3-32B-Instruct | 32B | Terminus-2 | 11.25 | 3.92 |
| Qwen3-14B-Instruct | 14B | Terminus-2 | 5.23 | 4.04 |
| Qwen3-8B-Instruct | 8B | Terminus-2 | 8.86 | 1.43 |
| Qwen3-32B-Nex-N1 | 32B | OpenHands | 28.75 | 16.70† |
| *Plugin: TACO* | | | | |
| Qwen3-Coder-480B-Instruct | 480B | TACO+Terminus-2 | **38.50**$^{\uparrow 1.00}$ | **25.86**$^{\uparrow 1.96}$ |
| MiniMax-M2.5 | 230B | TACO+Terminus-2 | **45.25**$^{\uparrow 2.95}$ | **44.16**$^{\uparrow 1.36}$ |
| DeepSeek-V3.2 | 685B | TACO+Terminus-2 | **46.25**$^{\uparrow 2.32}$ | **42.77**$^{\uparrow 2.15}$ |
| Qwen3-32B-Instruct | 32B | TACO+Terminus-2 | **14.13**$^{\uparrow 2.88}$ | **7.48**$^{\uparrow 3.56}$ |
| Qwen3-14B-Instruct | 14B | TACO+Terminus-2 | **11.25**$^{\uparrow 6.02}$ | **6.15**$^{\uparrow 2.11}$ |
| Qwen3-8B-Instruct | 8B | TACO+Terminus-2 | **9.22**$^{\uparrow 0.36}$ | **3.67**$^{\uparrow 2.24}$ |

**Agent Scaffolds.** We use two representative agent scaffolds. **Terminus-2**, introduced by Terminal-Bench [Merrill et al., 2026], has been adapted to several terminal-oriented benchmarks, and we use it as the baseline scaffold for TB, CompileBench, DevEval, and CRUST-Bench. **Mini-SWE-Agent** is a minimalist software-engineering agent for repository-level issue solving and command-line interaction; following SWE-Bench [Yang et al., 2024], we use it for SWE-Bench Lite.

**Evaluation protocol and token accounting.** Unless otherwise specified, all in-house results are averaged over five evaluation runs. All reported token counts are end-to-end totals, including both backbone-agent execution tokens and auxiliary LLM calls used by TACO. We report run-level standard deviations for the main accuracy results in Appendix E.

# 4 Results and Discussion

## 4.1 Adaptation Across Heterogeneous Terminal Benchmarks

To evaluate the adaptation ability of TACO across heterogeneous terminal environments, we conduct experiments on several terminal-related benchmarks, including TerminalBench (TB 1.0 and TB 2.0), SWE-Bench Lite, CompileBench, DevEval, and CRUST-Bench. We use MiniMax-M2.5 as the backbone model for these evaluations. As shown in Tab. 1, TACO consistently maintains or improves task success across all six benchmarks while reducing end-to-end token consumption, especially in log-heavy settings. These results suggest that the self-evolving mechanism of TACO generalizes beyond a single benchmark and adapts to diverse terminal-related environments. All reported token reductions are measured end-to-end: auxiliary rule-evolution calls are included in total token consumption, yet account for less than 2% across all benchmarks and about 1% on average; see Appendix J.

Table 1: Results across six benchmarks.

| Benchmark | Accuracy (%) | | Total Tokens (millions) | |
|---|---|---|---|---|
| | Baseline | TACO | Baseline | TACO |
| SWE-Bench-Lite | 56.30 | **57.12**$^{\uparrow 0.82}$ | 307.61 | **270.53**$^{\downarrow 12.1\%}$ |
| CompileBench | 75.00 | 75.00 | 14.55 | **11.41**$^{\downarrow 21.6\%}$ |
| DevEval | 38.10 | **39.74**$^{\uparrow 1.64}$ | 36.72 | **26.82**$^{\downarrow 27.0\%}$ |
| CRUST-Bench | 47.00 | **48.05**$^{\uparrow 1.05}$ | 163.53 | **134.97**$^{\downarrow 17.5\%}$ |
| TB 1.0 | 42.30 | **45.25**$^{\uparrow 2.95}$ | 29.74 | **23.43**$^{\downarrow 21.2\%}$ |
| TB 2.0 | 42.80 | **44.16**$^{\uparrow 1.36}$ | 113.74 | **110.63**$^{\downarrow 2.7\%}$ |

### 4.2 Adaptability of TACO Across Agentic Models

To verify that TACO is not tailored to a specific model, we evaluate it on TerminalBench (TB 1.0 and TB 2.0) across multiple agentic models.

**TACO consistently improves agent performance.** As shown in Tab. 2, incorporating TACO leads to consistent improvements across the evaluated models on both benchmarks. The absolute gains range from 0.36 to 6.02 points, suggesting that step-wise terminal observation compression can enhance agent performance. In particular, **Qwen3-Coder-480B** and **Qwen3-32B-Instruct** improve by **1.00** and **2.88** points on **TB 1.0**, and by **1.96** and **3.56** points on **TB 2.0**, respectively. As discussed in Appendix K, we hypothesize that these gains arise from reduced contextual redundancy, which helps the model better focus on task-relevant information. Closed-source TerminalBench results are provided in Appendix F for reference.

**TACO improves token efficiency across model scales.** Tab. 3 reports accuracy, average steps per run, and average prompt tokens per step on TB 2.0. For models with more than 200B parameters, such as Qwen3-Coder-480B and DeepSeek-V3.2, TACO reduces per-step token cost by approximately 10% without substantially changing the average number of steps, indicating that high-capacity models can effectively leverage compressed context while maintaining stable reasoning trajectories. For models with fewer than 40B parameters, the per-step token reduction is relatively marginal. Since these models often fail prematurely on complex terminal tasks, TACO enables longer and more successful interaction trajectories by improving their use of terminal feedback, which naturally increases total steps and overall token consumption compared with their early-failure baselines.

Table 3: Result with token consumption.

| Model | Acc. (%) | Avg Step | Token/Step |
|---|---|---|---|
| *Baseline* | | | |
| Qwen3-Coder-480B | 23.9 | 45.7 | 21,718 |
| DeepSeek-V3.2 | 40.6 | 29.5 | 35,038 |
| MiniMax-M2.5 | 42.8 | 43.2 | 28,631 |
| Qwen3-32B | 3.9 | 15.7 | 8,472 |
| Qwen3-14B | 4.0 | 30.3 | 9,663 |
| Qwen3-8B | 1.4 | 44.3 | 9,579 |
| *+TACO* | | | |
| Qwen3-Coder-480B | **25.9**$^{\uparrow 2.0}$ | 47.0 | **19,965**$^{\downarrow 8.1\%}$ |
| DeepSeek-V3.2 | **42.7**$^{\uparrow 2.1}$ | 30.6 | **30,939**$^{\downarrow 11.7\%}$ |
| MiniMax-M2.5 | **44.1**$^{\uparrow 1.3}$ | 42.6 | **28,559**$^{\downarrow 0.3\%}$ |
| Qwen3-32B | **7.4**$^{\uparrow 3.5}$ | 19.6 | 8,735$^{\uparrow 3.1\%}$ |
| Qwen3-14B | **6.1**$^{\uparrow 2.1}$ | 32.4 | **9,393**$^{\downarrow 2.8\%}$ |
| Qwen3-8B | **3.6**$^{\uparrow 2.2}$ | 68.5 | 9,583$^{\uparrow 0.04\%}$ |

### 4.3 Efficiency Comparison

Considering differences in model capability, we set the budget range for each model according to the token-usage distribution observed under the vanilla agent scaffold (i.e., Terminus-2).

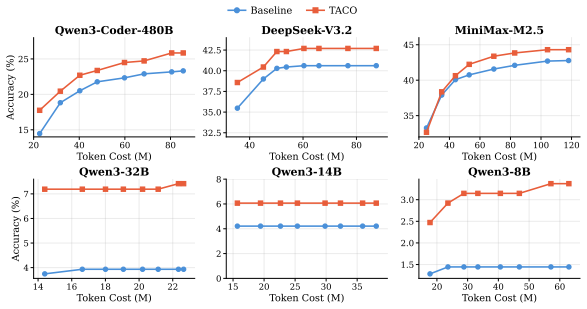


Figure 3: Agent Accuracy Under Identical Token Budgets.

**TACO improves accuracy under fixed token budgets.** We evaluate the token efficiency of agents with and without TACO on TB 2.0. As shown in Fig. 3, TACO consistently improves performance across all six models under the same token budgets, ranging from 14 million to 120 million tokens. For Qwen3-Coder-480B, DeepSeek-V3.2, and MiniMax-M2.5, TACO yields stable gains of 1%–2% across all budget settings, while for smaller models below 32B parameters, the gains are generally around 2%–3%. We further investigate the impact of TACO on model potential. As shown in Appendix H, TACO substantially enhances the agent's test-time scaling capability.

### 4.4 Comparison with Compression Methods

We compare TACO with three static compression baselines on TB 2.0 using Qwen3-Coder-480B. These baselines cover representative static heuristic compression strategies for terminal observations, including command-aware truncation and error-preserving compression. Specifically, we consider **High-Quality Rules**, which use 200 human-curated structured rules; **LLM-Gen Rules**, which ask an LLM to generate task-conditioned rules but keep them fixed during execution; and **LLM Summarization**, which directly summarizes terminal observation with an LLM. We additionally

evaluate ACON [Kang et al., 2025], with and without guideline optimization, under the same scaffold and experimental settings. The construction of the High-Quality Rules is described in Appendix L.2, and LLM-based compression prompts are provided in Appendix N.4.

As shown in Tab. 4, stronger token reduction does not necessarily lead to better task performance. ACON with guideline optimization reduces token consumption by 44.70%, but its mean accuracy remains below the uncompressed baseline. The auxiliary token overhead of TACO is 0.82%, compared with 2.37% for ACON and 4.08% for the optimized variant. Although LLM Summarization and the 200 human-curated rules remove more tokens than TACO, they yield smaller accuracy gains, suggesting that terminal observation compression should prioritize preserving task-relevant signals over maximizing compression ratio. Moreover, both instruction-only LLM-generated rules and statically predefined rules show a substantial gap compared with TACO, highlighting the importance of interactively refining compression rules based on environmental feedback and compression outcomes. By combining task-time adaptation with reusable rules from the Global Rule Pool, TACO selectively removes low-value terminal noise while retaining information needed for later decisions.

Table 4: Comparison of compression methods on TB 2.0.

| Method | Acc. | Tok. Red. |
|---|---|---|
| Baseline | $23.9 \pm 2.9$ | 0.00 |
| + HQ Rule | $24.3 \pm 2.2$ | 17.90 |
| + LLM-Gen Rule | $19.71 \pm 1.6$ | 7.10 |
| + LLM Sum. | $20.3 \pm 3.2$ | 21.30 |
| + ACON | $22.9 \pm 3.2$ | $-0.78$ |
| + ACON w/ guideline optimization | $23.4 \pm 2.4$ | **44.70** |
| + TACO | $\mathbf{25.9 \pm 1.5}$ | 4.60 |

## 4.5 Convergence Metric Validation

We propose a convergence metric, *Retention*, to determine whether the self-evolution process has converged. To assess its effectiveness, we use the sliding standard deviation of task accuracy as an empirical measure of performance stability.

As shown in Fig. 4, on TB 2.0, the sliding standard deviation decreases substantially once convergence is detected by *Retention*. Across all three models, it drops from above 2.0 before convergence to around 1.0 afterward, indicating that *Retention* provides a reliable signal of convergence in the self-evolution process.

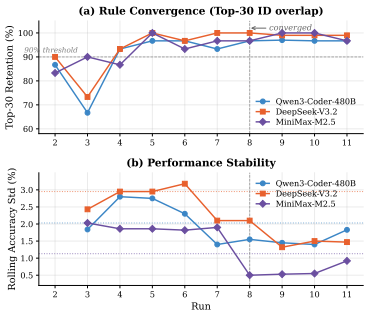


Figure 4: Rule-frontier convergence and performance stability.

## 4.6 The Quality of Rule Generated by TACO

**Observation-level compression quality.** To assess whether TACO preserves information needed for subsequent decisions, we randomly sample $N = 200$ observation pairs before and after compression from Qwen3-Coder-480B trajectories on TB 2.0. For each pair, an LLM judge evaluates three aspects: preservation of potentially task-critical information, removal of redundancy, and retention of useful non-critical context. The full protocol and judge prompt are provided in Appendix M. Evaluation results indicate that TACO preserves potentially critical information in 96.0% of cases, removes redundant content in 81.0%, and retains useful non-critical context in 78.5%. Only 4.0% of cases are flagged as potential critical losses. We manually inspect all eight flagged cases in Appendix M.2: seven are benign event-level flags with no observable behavioral impact, while the remaining case triggers the intended `need_full_output` complaint signal, providing feedback for subsequent rule evolution. These results demonstrate the robustness of TACO for preservation-aware context compression.

Table 5: Task-solving number changes.

| Model | Rescued | Regressed |
|---|---|---|
| Qwen3-8B | **1** | 0 |
| Qwen3-14B | **2** | 0 |
| Qwen3-32B | **1** | 0 |
| Qwen3-Coder-480B | **5** | 2 |
| DeepSeek-V3.2 | **8** | 4 |
| MiniMax-M2.5 | **4** | 3 |

**Task-level rescue and regression.** We analyze task-solving number changes between Baseline and TACO on TB 2.0. A task is counted as *rescued* if Baseline fails but TACO succeeds, and as *regressed* if Baseline succeeds but TACO fails. To reduce randomness, we average outcomes over the last five converged runs and consider a task solved only if at least 50% of runs solve it. As shown in Tab. 5, TACO rescues more tasks than it regresses across all models, suggesting that adaptive observation compression more often improves execution than harms it by filtering redundant observations and

preserving useful context for later decisions. Appendix O presents a retrospective analysis of the 16 tasks with decreased mean success rates, summarizing three patterns of content removal and corrective feedback.

### 4.7 Rule Reusability

Although TACO requires multiple cold-start iterations to reach rule convergence, this cost is incurred only once. Once converged, the learned Global Rule Pool can be frozen and reused for subsequent tasks, avoiding repeated test-time scaling. As shown in Tab. 6, the reused rules remain effective on TB 2.0, indicating that they capture generalizable terminal-output patterns rather than benchmark-specific artifacts. Therefore, cold-start inference cost can be amortized over future deployments.

Table 6: Rule reusability on TB 2.0.

| Model | Setting | Acc. (%) |
|---|---|---|
| DeepSeek-V3.2 | Baseline | 40.62 |
| | TACO (Online) | **42.77** |
| | TACO (Reuse) | 42.69 |
| Qwen3-480B | Baseline | 23.90 |
| | TACO (Online) | 25.86 |
| | TACO (Reuse) | **26.96** |
| MiniMax-M2.5 | Baseline | 42.80 |
| | TACO (Online) | 44.16 |
| | TACO (Reuse) | **44.94** |

### 4.8 Ablation Study

To investigate the contribution of self-evolution, we remove the two evolutionary components of TACO, namely **Intra-Task Rule Set Evolution** (ITRSE) and **Global Rule Pool Evolution** (GRPE), and evaluate the resulting variants on TB 2.0 with DeepSeek-V3.2.

For the variant without **Intra-Task Rule Set Evolution**, we use the final Global Rule Pool from TACO on TB 2.0 as a fixed initialization source for each task. During execution, the intra-task rule set remains unchanged, and neither task-specific rules nor the Global Rule Pool are updated. For the variant without **Global Rule Pool Evolution**, we perform only task-level rule evolution within each task, without maintaining or updating a shared Global Rule Pool. As shown in Tab. 7, using either component alone can reduce total token consumption, but it also leads to degraded task performance. This indicates that purely static compression is insufficient for terminal contexts, while rules derived from individual tasks alone have limited quality and generalizability. In contrast, TACO evolves a shared Global Rule Pool, enabling it to accumulate high-quality, reusable, and generalizable compression rules.

Table 7: Ablation results on TB 2.0. Total tokens are reported in millions (M).

| Method | Acc. | Total Tok. (M) |
|---|---|---|
| Baseline | 40.6% | 99.60 |
| TACO w/o GRPE | 40.4% | 81.57 |
| TACO w/o ITRSE | 38.9% | **69.02** |
| **TACO (Full)** | **42.7%** | 81.45 |

## 5 Related Work

**Context Compression for Code and Terminal.** As LLM agents move toward long-horizon software-engineering and terminal-based tasks [Jimenez et al., 2024, Merrill et al., 2026, Yang et al., 2024, Wang et al., 2024, Wu et al., 2026], context compression becomes increasingly important for reducing cost and improving reliability. General prompt-compression methods, including token pruning, information filtering, retrieval-based selection, and LLM summarization [Jiang et al., 2023, Pan et al., 2024, Li et al., 2023], shorten inputs by removing redundant content. However, terminal observations pose a distinct safety challenge: verbose logs, compiler traces, test reports, and shell outputs interleave low-value noise with sparse but exact execution evidence, such as error messages, file paths, test names, command arguments, and package versions. Generic pruning may discard these critical strings, while abstractive summarization may paraphrase or omit them. Recent code-oriented compressors, such as LongCodeZip and SWE-Pruner [Shi et al., 2025, Wang et al., 2026], are closer to our motivation, but mainly target source-code or repository contexts and often rely on program structure or trained pruning models.

**Agent Context Management and Self-Evolving Agents.** Another related direction studies how long-horizon agents manage accumulated interaction histories. Existing methods use trajectory summarization, memory retrieval, context truncation, observation masking, proactive context folding, or learned history compression to retain, summarize, fold, or remove past messages, actions, observations, and intermediate reasoning traces [Liu et al., 2025b, Kang et al., 2025, Ye et al., 2025, Wan et al., 2025]. These approaches help control growing histories, but they primarily operate after observations have already entered the agent context. Meanwhile, training-free self-evolving agents

improve future task solving by accumulating reusable memories, skills, plans, tools, or symbolic artifacts [Zhou et al., 2026, 2024].

In contrast, TACO compresses terminal observations before they enter the agent history and evolves reusable, preservation-aware compression rules, rather than task-solving strategies.

## 6 Conclusion

We present TACO, a plug-and-play, self-evolving terminal observation compression framework for terminal agents. By automatically discovering, refining, and reusing compression rules from interaction trajectories, TACO enables adaptive and training-free context compression across diverse terminal environments. Experiments on TerminalBench and additional terminal-related benchmarks show that TACO can improve both task performance and token efficiency across different agent frameworks and backbone models. These results highlight the importance of removing redundant terminal context for long-horizon reasoning and suggest a practical path toward more efficient and effective terminal agents.

## Acknowledgements and Disclosures

This work was supported by the National Key Research and Development Program of China (2024YFC3014900) and the Royal Society International Exchanges Grant (IEC\NSFC\242347). Min Gao is further support by Expert Program (H20251055). The authors declare no competing interests.

## References

Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong, Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, Yichen Feng, Kelin Fu, Bofei Gao, Hongcheng Gao, Peizhong Gao, Tong Gao, Xinran Gu, Longyu Guan, Haiqing Guo, Jianhang Guo, Hao Hu, Xiaoru Hao, Tianhong He, Weiran He, Wenyang He, Chao Hong, Yangyang Hu, Zhenxing Hu, Weixiao Huang, Zhiqi Huang, Zihao Huang, Tao Jiang, Zhejun Jiang, Xinyi Jin, Yongsheng Kang, Guokun Lai, Cheng Li, Fang Li, Haoyang Li, Ming Li, Wentao Li, Yanhao Li, Yiwei Li, Zhaowei Li, Zheming Li, Hongzhan Lin, Xiaohan Lin, Zongyu Lin, Chengyin Liu, Chenyu Liu, Hongzhang Liu, Jingyuan Liu, Junqi Liu, Liang Liu, Shaowei Liu, T. Y. Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yibo Liu, Yiping Liu, Yue Liu, Zhengying Liu, Enzhe Lu, Lijun Lu, Shengling Ma, Xinyu Ma, Yingwei Ma, Shaoguang Mao, Jie Mei, Xin Men, Yibo Miao, Siyuan Pan, Yebo Peng, Ruoyu Qin, Bowen Qu, Zeyu Shang, Lidong Shi, Shengyuan Shi, Feifan Song, Jianlin Su, Zhengyuan Su, Xinjie Sun, Flood Sung, Heyi Tang, Jiawen Tao, Qifeng Teng, Chensi Wang, Dinglu Wang, Feng Wang, Haiming Wang, Jianzhou Wang, Jiaxing Wang, Jinhong Wang, Shengjie Wang, Shuyi Wang, Yao Wang, Yejie Wang, Yiqin Wang, Yuxin Wang, Yuzhi Wang, Zhaoji Wang, Zhengtao Wang, Zhexu Wang, Chu Wei, Qianqian Wei, Wenhao Wu, Xingzhe Wu, Yuxin Wu, Chenjun Xiao, Xiaotong Xie, Weimin Xiong, Boyu Xu, Jing Xu, Jinjing Xu, L. H. Xu, Lin Xu, Suting Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ziyao Xu, Junjie Yan, Yuzi Yan, Xiaofei Yang, Ying Yang, Zhen Yang, Zhilin Yang, Zonghan Yang, Haotian Yao, Xingcheng Yao, Wenjie Ye, Zhuorui Ye, Bohong Yin, Longhui Yu, Enming Yuan, Hongbang Yuan, Mengjie Yuan, Haobing Zhan, Dehao Zhang, Hao Zhang, Wanlu Zhang, Xiaobin Zhang, Yangkun Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Haotian Zhao, Yikai Zhao, Huabin Zheng, Shaojie Zheng, Jianren Zhou, Xinyu Zhou, Zaida Zhou, Zhen Zhu, Weiyu Zhuang, and Xinxing Zu. Kimi k2: Open agentic intelligence, 2025. URL `https://arxiv.org/abs/2507.20534`.

Yuxuan Cai, Lu Chen, Qiaoling Chen, Yuyang Ding, Liwen Fan, Wenjie Fu, Yufei Gao, Honglin Guo, Pinxue Guo, Zhenhua Han, et al. Nex-n1: Agentic models trained via a unified ecosystem for large-scale environment construction. *arXiv preprint arXiv:2512.04987*, 2025.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Llmlingua: Compressing prompts for accelerated inference of large language models, 2023. URL `https://arxiv.org/abs/2310.05736`.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R 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, 2025. URL https://arxiv.org/abs/2510.00615.

Anirudh Khatry, Robert Zhang, Jia Pan, Ziteng Wang, Qiaochu Chen, Greg Durrett, and Isil Dillig. Crust-bench: A comprehensive benchmark for c-to-safe-rust transpilation, 2025. URL https://arxiv.org/abs/2504.15254.

Jia Li, Ge Li, et al. Deveval: A manually-annotated code generation benchmark aligned with real-world code repositories. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024.

Yucheng Li, Bo Dong, Chenghua Lin, and Frank Guerin. Compressing context to enhance inference efficiency of large language models, 2023. URL https://arxiv.org/abs/2310.06201.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025a.

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 preprint arXiv:2512.22087*, 2025b.

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

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, 2024. URL https://arxiv.org/abs/2403.12968.

Yuling Shi, Yichun Qian, Hongyu Zhang, Beijun Shen, and Xiaodong Gu. Longcodezip: Compress long context for code language models, 2025. URL https://arxiv.org/abs/2510.00446.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Guangya Wan, Mingyang Ling, Xiaoqi Ren, Rujun Han, Sheng Li, and Zizhao Zhang. Compass: Enhancing agent long-horizon reasoning with evolving context, 2025. URL https://arxiv.org/abs/2510.08790.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024.

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. *arXiv preprint arXiv:2601.16746*, 2026.

Siwei Wu, Zhongyuan Peng, Xinrun Du, Tuney Zheng, Minghao Liu, Jialong Wu, Jiachen Ma, Yizhi Li, Jian Yang, Wangchunshu Zhou, Qunshu Lin, Junbo Zhao, Zhaoxiang Zhang, Wenhao Huang, Ge Zhang, Chenghua Lin, and J. H. Liu. A comparative study on reasoning patterns of openai's o1 model, 2024. URL https://arxiv.org/abs/2410.13639.

Siwei Wu, Yizhi Li, Yuyang Song, Wei Zhang, Yang Wang, Riza Batista-Navarro, Xian Yang, Mingjie Tang, Bryan Dai, Jian Yang, and Chenghua Lin. Large-scale terminal agentic trajectory generation from dockerized environments, 2026. URL https://arxiv.org/abs/2602.01244.

Jian Yang, Wei Zhang, Shark Liu, Jiajun Wu, Shawn Guo, and Yizhi Li. From code foundation models to agents and applications: A practical guide to code intelligence. *arXiv e-prints*, pages arXiv–2511, 2025.

Jian Yang, Wei Zhang, Shawn Guo, Zhengmao Ye, Lin Jing, Shark Liu, Yizhi Li, Jiajun Wu, Cening Liu, X Ma, et al. Iquest-coder-v1 technical report. *arXiv preprint arXiv:2603.16733*, 2026a.

Jian Yang, Wei Zhang, Jiajun Wu, Junhang Cheng, Shawn Guo, Haowen Wang, Weicheng Gu, Yaxin Du, Joseph Li, Fanglin Xu, et al. Incoder-32b: Code foundation model for industrial scenarios. *arXiv preprint arXiv:2603.16790*, 2026b.

Jian Yang, Wei Zhang, Jiajun Wu, Junhang Cheng, Tuney Zheng, Fanglin Xu, Weicheng Gu, Lin Jing, Yaxin Du, Joseph Li, et al. Incoder-32b-thinking: Industrial code world model for thinking. *arXiv preprint arXiv:2604.03144*, 2026c.

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. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, Pengjun Xie, Fei Huang, Siheng Chen, Jingren Zhou, and Yong Jiang. Agentfold: Long-horizon web agents with proactive context management, 2025. URL `https://arxiv.org/abs/2510.24699`.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, Yean Cheng, Yifan An, Yilin Niu, Yuanhao Wen, Yushi Bai, Zhengxiao Du, Zihan Wang, Zilin Zhu, Bohan Zhang, Bosi Wen, Bowen Wu, Bowen Xu, Can Huang, Casey Zhao, Changpeng Cai, Chao Yu, Chen Li, Chendi Ge, Chenghua Huang, Chenhui Zhang, Chenxi Xu, Chenzheng Zhu, Chuang Li, Congfeng Yin, Daoyan Lin, Dayong Yang, Dazhi Jiang, Ding Ai, Erle Zhu, Fei Wang, Gengzheng Pan, Guo Wang, Hailong Sun, Haitao Li, Haiyang Li, Haiyi Hu, Hanyu Zhang, Hao Peng, Hao Tai, Haoke Zhang, Haoran Wang, Haoyu Yang, He Liu, He Zhao, Hongwei Liu, Hongxi Yan, Huan Liu, Huilong Chen, Ji Li, Jiajing Zhao, Jiamin Ren, Jian Jiao, Jiani Zhao, Jianyang Yan, Jiaqi Wang, Jiayi Gui, Jiayue Zhao, Jie Liu, Jijie Li, Jing Li, Jing Lu, Jingsen Wang, Jingwei Yuan, Jingxuan Li, Jingzhao Du, Jinhua Du, Jinxin Liu, Junkai Zhi, Junli Gao, Ke Wang, Lekang Yang, Liang Xu, Lin Fan, Lindong Wu, Lintao Ding, Lu Wang, Man Zhang, Minghao Li, Minghuan Xu, Mingming Zhao, Mingshu Zhai, Pengfan Du, Qian Dong, Shangde Lei, Shangqing Tu, Shangtong Yang, Shaoyou Lu, Shijie Li, Shuang Li, Shuang-Li, Shuxun Yang, Sibo Yi, Tianshu Yu, Wei Tian, Weihan Wang, Wenbo Yu, Weng Lam Tam, Wenjie Liang, Wentao Liu, Xiao Wang, Xiaohan Jia, Xiaotao Gu, Xiaoying Ling, Xin Wang, Xing Fan, Xingru Pan, Xinyuan Zhang, Xinze Zhang, Xiuqing Fu, Xunkai Zhang, Yabo Xu, Yandong Wu, Yida Lu, Yidong Wang, Yilin Zhou, Yiming Pan, Ying Zhang, Yingli Wang, Yingru Li, Yinpei Su, Yipeng Geng, Yitong Zhu, Yongkun Yang, Yuhang Li, Yuhao Wu, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yuxuan Zhang, Zezhen Liu, Zhen Yang, Zhengda Zhou, Zhongpei Qiao, Zhuoer Feng, Zhuorui Liu, Zichen Zhang, Zihan Wang, Zijun Yao, Zikang Wang, Ziqiang Liu, Ziwei Chai, Zixuan Li, Zuodong Zhao, Wenguang Chen, Jidong Zhai, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models, 2025. URL `https://arxiv.org/abs/2508.06471`.

Huichi Zhou, Siyuan Guo, Anjie Liu, Zhongwei Yu, Ziqin Gong, Bowen Zhao, Zhixun Chen, Menglong Zhang, Yihang Chen, Jinsong Li, Runyu Yang, Qiangbin Liu, Xinlei Yu, Jianmin Zhou, Na Wang, Chunyang Sun, and Jun Wang. Memento-skills: Let agents design agents, 2026. URL `https://arxiv.org/abs/2603.18743`.

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`.

King Zhu, Hanhao Li, Siwei Wu, Tianshun Xing, Dehua Ma, Xiangru Tang, Minghao Liu, Jian Yang, Jiaheng Liu, Yuchen Eleanor Jiang, Changwang Zhang, Chenghua Lin, Jun Wang, Ge Zhang, and Wangchunshu Zhou. Scaling test-time compute for llm agents, 2025. URL `https://arxiv.org/abs/2506.12928`.

# A Limitations

TACO operates at the observation-compression layer rather than the model-capability layer. It does not modify the backbone model or improve its intrinsic capability; instead, it helps existing agents make better use of their context budget by presenting cleaner and less redundant terminal observations. TACO may offer limited gains on short workflows with little redundancy and require conservative rules for sparse, brittle evidence.

# B Ethics Statement

Our experiments are conducted on publicly available software-engineering and terminal-agent benchmarks, and do not involve private user data or personally identifiable information. The proposed TACO method itself does not rely on manual rule engineering. Human annotation was used solely to construct the High-Quality Rule static comparison baseline. Specifically, we recruited three Master's students in computer science to write structured compression rules based on sampled terminal-agent trajectories and predefined guidelines. This process involved only public benchmark trajectories and terminal outputs; it did not entail sensitive personal data, user profiling, or human-subject behavioral studies. The annotators were compensated at approximately $19 per hour, which is above the applicable local minimum wage.

# C Compute Resources

TACO is training-free and does not require additional model training. All reported results are averaged over five independent runs unless otherwise stated. All backbone models are served locally on a 20×NVIDIA H800 GPU cluster.

# D Broader impacts

This work studies observational context compression for terminal agents and is not a standalone deployed system. Its positive impacts include reducing compute costs and improving the efficiency of long-horizon workflows. Any potential risks are strictly inherent to the underlying terminal agents rather than the compression mechanism itself. We therefore recommend deploying TACO alongside standard safety protocols, such as sandboxed execution environments and human oversight.

# E Statistical Details

We report the mean accuracy and run-level standard deviation over five evaluation runs. The results are summarized in Tab. 8 and Tab. 9.

Table 8: Accuracy (%) and run-level standard deviation on TerminalBench. All values are computed over five evaluation runs.

| Model | Method | TB 1.0 | TB 2.0 |
|---|---|---|---|
| | | Acc. (%) | Acc. (%) |
| MiniMax-M2.5 | Baseline | $42.30 \pm 3.94$ | $42.80 \pm 2.54$ |
| | TACO | $45.25 \pm 1.76$ | $44.16 \pm 0.86$ |
| DeepSeek-V3.2 | Baseline | $43.93 \pm 2.98$ | $40.62 \pm 2.88$ |
| | TACO | $46.25 \pm 2.62$ | $42.77 \pm 2.54$ |
| Qwen3-Coder-480B | Baseline | $37.50 \pm 3.93$ | $23.90 \pm 2.85$ |
| | TACO | $38.50 \pm 2.23$ | $25.86 \pm 1.48$ |
| Qwen3-8B | Baseline | $8.86 \pm 2.27$ | $1.43 \pm 1.31$ |
| | TACO | $9.22 \pm 1.01$ | $3.67 \pm 0.98$ |
| Qwen3-14B | Baseline | $5.23 \pm 2.15$ | $4.04 \pm 2.23$ |
| | TACO | $11.25 \pm 2.32$ | $6.15 \pm 1.67$ |
| Qwen3-32B | Baseline | $11.25 \pm 2.29$ | $3.92 \pm 3.13$ |
| | TACO | $14.13 \pm 1.79$ | $7.48 \pm 2.44$ |

Table 9: Accuracy (%) and run-level standard deviation on downstream terminal-related benchmarks. All values are computed over five evaluation runs.

| Benchmark | Method | MiniMax-M2.5 Acc. (%) |
|---|---|---|
| SWE-Bench Lite | Baseline | $56.30 \pm 2.32$ |
| | TACO | $57.12 \pm 2.19$ |
| CompileBench | Baseline | $75.00 \pm 8.69$ |
| | TACO | $75.00 \pm 7.82$ |
| DevEval | Baseline | $38.10 \pm 1.56$ |
| | TACO | $39.74 \pm 1.17$ |
| CRUST-Bench | Baseline | $47.00 \pm 3.43$ |
| | TACO | $48.05 \pm 2.78$ |

## F Additional TerminalBench Results with Closed-Source Models

For reference, we report closed-source model results on TerminalBench in Tab. 10.

Table 10: Closed-source model results on TerminalBench. * indicates results reported on the TerminalBench leaderboard.

| **Model** | **Agent Scaffold** | **TB 1.0** | **TB 2.0** |
|---|---|---|---|
| Gemini-3-Flash-Preview | Terminus-2 | 53.72 | 51.70* |
| Gemini-3-Pro-Preview | Terminus-2 | 46.35 | 56.90* |
| Claude-Opus-4.5 | Terminus-2 | 47.50 | 57.80* |
| Claude-Sonnet-4.5 | Terminus-2 | 51.00* | 42.80* |
| GPT-5.1 | Terminus-2 | 35.50 | 47.60* |
| GPT-5.2 | Terminus-2 | 54.38 | 54.00* |

## G Hyperparameter Selection

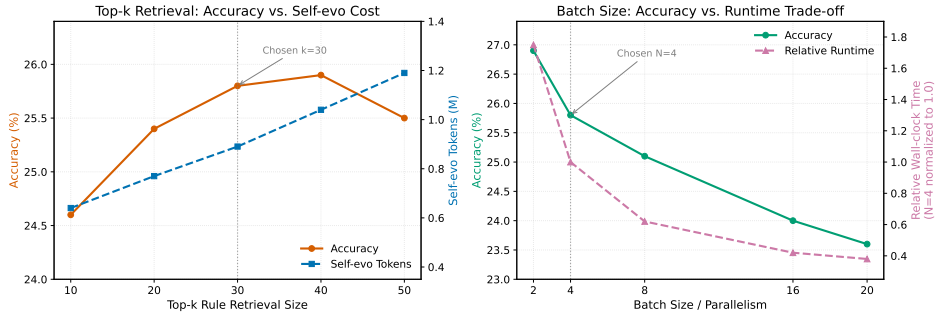


Figure 5: Hyperparameter selection for TACO. Left: effect of the top-$k$ rule retrieval size on accuracy and self-evolution token cost. Right: effect of batch size $N$ on accuracy and relative runtime.

The batch size $N$ in TACO's parallel execution and the **Top-$k$** hyperparameter for task-specific rule initialization both shape the interaction between the Intra-Task Rule Pool and the Global Rule Pool, and thus affect the final performance. To examine their effects, we perform an ablation study on TB 2.0 with Qwen3-Coder-480B.

**Top-$k$ Selection.** To examine the effect of Top-$k$ under a fixed computational budget, we fix the batch size at $N = 4$ and vary Top-$k$. We evaluate both agent accuracy and the token cost of rule selection. As shown in the left part of Fig. 5, a smaller $k$ reduces token consumption but limits performance by narrowing the pool of candidate historical rules. As $k$ increases, accuracy improves at first and then gradually saturates, while the cost of self-evolution continues to rise. When $k > 30$,

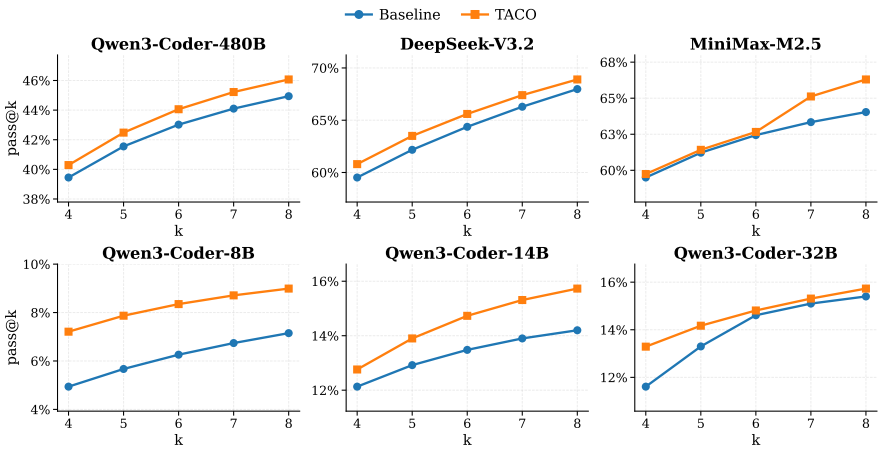


Figure 6: Pass@k comparison between Baseline and TACO across six models on TB 2.0. TACO consistently improves pass@k across all evaluated settings.

| **Information Source** | **Used by TACO** |
|---|---|
| Terminal observation | Yes |
| Executed command | Yes |
| Rule application count | Yes |
| Final task success/failure | No |
| Hidden tests or verifier result | No |
| Ground-truth answer or patch | No |
| Leaderboard score | No |

Table 11: Information sources used during TACO rule evolution. TACO updates compression rules only from observation-level interaction signals and does not access benchmark correctness signals.

the additional performance gain becomes marginal or even negative, whereas the token cost keeps increasing. Moreover, a larger $k$ leads to a longer retrieval context, which may be less stable for smaller models. Therefore, we set $k = 30$ in the main experiments.

**Batch Size $N$ Selection.** We fix Top-$k$ at 30 and investigate the effect of different batch sizes $N$ on TB 2.0. As shown in the right part of Fig. 5, smaller batch sizes tend to yield higher accuracy, since newly learned rules can be written back more frequently and reused by subsequent tasks sooner. In contrast, larger batch sizes improve parallel throughput and overall runtime efficiency, but delay the application of newly acquired rules to later tasks. Balancing accuracy, throughput, and rule-propagation efficiency, we adopt $N = 4$ in the main experiments.

## H Best-of-K Trajectory Sampling

Test-time scaling (e.g., Best-of-K) increases inference-time compute by sampling multiple candidate solutions, often leading to substantial performance gains. Pass@k, a Best-of-K-style metric, has been widely used to evaluate agents' potential ability [Wu et al., 2024, Zhu et al., 2025, Snell et al., 2024]. Therefore, to further demonstrate that TACO improves agents' potential ability, we compare pass@k for $k \in \{4, 5, 6, 7, 8\}$ across six models on TB 2.0, with and without TACO. As shown in Fig. 6, TACO consistently improves pass@k across all models and k, suggesting that its compression rules improve the quality of individual trajectories and remain complementary to test-time scaling.

Table 12: Auxiliary token overhead of TACO rule evolution. All main-paper token counts include this overhead.

| Benchmark | Aux. Tok. / Total Tok. |
|---|---|
| SWE-Bench Lite | 1.07% |
| CompileBench | 1.19% |
| CRUST-Bench | 0.61% |
| DevEval | 1.88% |
| TB 1.0 | 0.94% |
| TB 2.0 | 0.82% |

# I Information Used in Reward-Free Unsupervised Rule Evolution

As shown in Tab. 11, we present the information used for rule evolution. This table formalizes the boundary of TACO's unsupervised adaptation: it may use observable interaction traces, but not correctness, reward, or answer signals.

# J Token and Runtime Overhead

To ensure fair cost accounting, all token numbers reported in the main paper include both backbone-agent execution tokens and auxiliary LLM calls introduced by TACO for rule selection, generation, and revision. As shown in Tab. 12, the auxiliary rule-evolution overhead remains below 2% across all benchmarks, indicating that the reported token reductions are net end-to-end savings rather than excluding the cost of self-evolution.

**Wall-clock overhead.** We compare end-to-end runtime on TB 2.0 using Qwen3-Coder-480B with the Terminus-2 scaffold, four parallel workers, and identical hardware. During the first five evolution rounds, TACO takes 3.88 ± 0.58 hours per run, compared with 3.74 ± 0.61 hours for the baseline (mean ± standard deviation), an average increase of 0.14 hours. The average time per trial is 9.8 minutes for TACO and 9.6 minutes for the baseline. Rule application uses deterministic pattern matching, while auxiliary LLM calls are used for rule selection, generation, and revision rather than for every observation.

# K Case Study

To provide a concrete illustration of TACO's self-evolving compression behavior, we examine three successful trajectories from TerminalBench 2.0, executed by the Terminus-2 agent with Qwen3-Coder-480B as the backbone. The selected tasks—`adaptive-rejection-sampler` (statistical computing in R), `sqlite-with-gcov` (build-system instrumentation), and `vulnerable-secret` (binary reverse engineering)—represent distinct terminal domains with qualitatively different output patterns. All three achieve a reward of 1.0, while TACO's compression mechanism reduces substantial token overhead throughout. We analyze these trajectories from three perspectives: (1) how task-specific rules are initialized and evolved online, (2) the quantitative compression statistics observed in the logs, and (3) concrete before-and-after comparisons demonstrating the transition from syntactic truncation to semantic filtering.

## K.1 Task-Specific Rule Initialization and Online Evolution

As described in Sec. 2.3, TACO initializes the task-specific active rule set $R_t$ by retrieving top-ranked rules from the Global Rule Pool $R_g$ and invoking an LLM to select, adapt, and generate new rules conditioned on the task description. Tab. 13 summarizes the rule composition for each trajectory.

**Plan-time rule generation.** The LLM generates task-specific rules that embed domain understanding directly into the compression logic, going beyond syntactic heuristics. For `sqlite-with-gcov`, the rule `sqlite_make_gcov_mod` targets `make` output with coverage flags—it strips per-object `gcc` invocation lines and `cp` file-listing commands, while explicitly retaining gcov-specific errors (`cannot find -lgcov`, `profiling error`) and the final linking step. For `vulnerable-secret`,

Table 13: Rule composition across three trajectories. "New (plan-time)" denotes task-specific rules generated at initialization; "New (mid-task)" denotes rules evolved reactively via the Intra-Task Rule Set Evolution mechanism (Sec. 2.5).

| Task | New (plan-time) | New (mid-task) |
|---|---|---|
| adaptive-rejection-sampler | 2 | 1 |
| sqlite-with-gcov | 2 | 0 |
| vulnerable-secret | 2 | 1 |

the rule secret_key_extractor aggressively filters all output from /app/ executables except lines matching FLAG{.*}, tracebacks, segmentation faults, and error signals—effectively transforming the compressor into a targeted information extractor for the hidden secret. For adaptive-rejection-sampler, the rule pytest_stochastic_testing_mod strips verbose PASSED lines and session headers from test output while preserving statistical validation signals (KS test p-values, chi-square results, sample mean/variance).

**Reactive mid-task evolution.** In two trajectories, TACO's Intra-Task Rule Set Evolution (Sec. 2.5) triggered reactive rule generation when the agent encountered lengthy terminal observation that were not covered by any active rule. In adaptive-rejection-sampler, the initial apt_install rule only partially compressed the output of apt-get install -y r-base, reducing the first chunk from 10,071 to 5,371 characters (ratio 0.53). However, a later continuation chunk remained largely uncompressed, despite being dominated by repetitive "Unpacking" and "Setting up" lines for over 200 packages. This exposed a coverage gap in the active rule rather than a need to preserve the full output, prompting TACO to refine the rule to collapse repeated package-lifecycle logs while retaining final status and error signals. TACO evolved apt_install_unpacked_packages, which matched and suppressed these repetitive lines, reducing the 10,071-character continuation to 73 characters (ratio 0.007)—a 99.3% reduction. In vulnerable-secret, the first objdump -d call at episode 9 produced 5,169 characters of disassembly with no matching rule. TACO generated objdump_disassembly_rule, which strips repetitive hex-dump instruction lines while preserving section headers, symbol labels (e.g., <gets@plt>), and address markers. This rule then fired on 18 subsequent objdump invocations across the remaining 92 episodes.

The sqlite-with-gcov task did not require new rules during execution. The rules initialized at the beginning of the task were sufficient to handle its terminal outputs, including compiler outputs and git-related noise. This indicates that TACO only needs online rule evolution when it encounters output patterns not covered by the initialized rule set.

### K.2 Quantitative Compression Analysis

Tab. 14 reports the compression statistics extracted from each trajectory.

Table 14: Compression statistics for the three case study trajectories. "Entries" counts the number of steps where rule-based compression was applied. "Chars saved" denotes the total reduction in terminal observation characters. "Best ratio" is the minimum compression ratio observed (lower = more aggressive compression).

| Task | Episodes | Entries | Chars saved | Overall ratio | Best ratio |
|---|---|---|---|---|---|
| adaptive-rejection-sampler | 25 | 4 | 14,766 | 0.285 | 0.007 |
| sqlite-with-gcov | 22 | 9 | 9,647 | 0.464 | 0.390 |
| vulnerable-secret | 101 | 25 | 29,636 | 0.455 | 0.146 |

Several observations stand out. First, the compression behavior is highly non-uniform: in adaptive-rejection-sampler, only 4 out of 25 episodes triggered compression, but those 4 entries accounted for over 14,000 characters of savings—concentrated entirely on the high-volume apt-get installation steps. Second, in vulnerable-secret, the evolved objdump_disassembly_rule accounts for 18 of the 25 compression entries and 29,464 of the 29,636 characters saved, demonstrating how a single reactively evolved rule can dominate the compression profile of a long-horizon task. Across all 18 objdump applications, the average compression

ratio is 0.445, with individual ratios ranging from 0.146 to 0.999 depending on the amount of disassembly output in each call.

### K.3 Before-and-After Comparison

To illustrate how TACO transitions from naïve truncation to task-aware semantic filtering, we present three representative compression examples drawn directly from the trajectory logs.

#### K.3.1 Package Installation (`apt-get install`)

In `adaptive-rejection-sampler` (episode 4), the agent installs the R runtime via `apt-get install -y r-base`, which triggers the download and configuration of 200+ dependency packages. The raw terminal observation is dominated by mechanically repetitive lines with no task-relevant content.

**Raw terminal observation** (10,071 characters):

```
Unpacking libc6-dev:amd64 (2.39-0ubuntu8.7) ...
Unpacking gcc-13-base:amd64 (13.3.0-6ubuntu2~24.04.1) ...
Unpacking libisl23:amd64 (0.26-3build1.1) ...
Unpacking libmpfr6:amd64 (4.2.1-1build1.1) ...
Unpacking libmpc3:amd64 (1.3.1-1build1.1) ...
Unpacking cpp-13-x86-64-linux-gnu (13.3.0-6ubuntu2~24.04.1) ...
[... 150+ Unpacking/Setting up lines ...]
Setting up r-base-core (4.3.3-2build2) ...
Setting up r-base (4.3.3-2build2) ...
Processing triggers for libc-bin (2.39-0ubuntu8.7) ...
```

**Compressed output** (73 characters, ratio = 0.007):

```
[WAITING] apt-get install -y r-base

Current status: Setting up x11-utils
```

The evolved rule `apt_install_unpacked_packages` recognizes that the hundreds of "`Unpacking`" and "`Setting up`" lines carry no information needed for subsequent reasoning—the only signal the agent requires is whether the installation succeeded or is still in progress. By compressing the output to a 73-character status indicator, TACO prevents a 10,000-character log from displacing task-relevant context (the R code and test outputs the agent needs for upcoming steps). This illustrates the efficiency gain described in Sec. 4.2: observation compression reduces contextual redundancy, allowing the model to more easily identify the critical information needed for task completion.

#### K.3.2 Build System Output (`make`)

In `sqlite-with-gcov` (episode 14), the agent executes `make` to compile SQLite with gcov coverage instrumentation. The raw output contains verbose file-copying commands alongside semantically important build configuration messages.

**Raw terminal observation** (6,519 characters):

```
$ make
touch .main.mk.checks
cc -g -o mksourceid .../tool/mksourceid.c
cc -g -o jimsh -O1 -DHAVE_REALPATH .../autosetup/jimsh0.c
./jimsh .../tool/mksqlite3h.tcl ... -o sqlite3.h
./jimsh .../tool/mkctimec.tcl
Overwriting ctime.c...
cc -g -o mkkeywordhash ... .../tool/mkkeywordhash.c
./mkkeywordhash > keywordhash.h
cc -g -o lemon .../tool/lemon.c
cp .../src/parse.y .
./lemon -DSQLITE_ENABLE_MATH_FUNCTIONS ... -S parse.y
cp -f .../src/alter.c .../src/analyze.c .../src/attach.c
  .../src/auth.c .../src/backup.c .../src/bitvec.c
  [... 100+ source files copied to tsrc/ ...]
rm -f tsrc/sqlite.h.in tsrc/parse.y
./jimsh .../tool/vdbe-compress.tcl <tsrc/vdbe.c >vdbe.new
touch .target_source
cc -fPIC -O2 -g -DSQLITE_COVERAGE_TEST=1 -fprofile-arcs
  -ftest-coverage -c sqlite3.c
```

**Compressed output** (3,183 characters, ratio = 0.488):

```
$ make
touch .main.mk.checks
cc -g -o mksourceid .../tool/mksourceid.c
[compiler output compressed -- long command lines truncated]
  [25 lines removed]
cc -g -o jimsh -O1 -DHAVE_REALPATH .../autosetup/jimsh0.c
./jimsh .../tool/mksqlite3h.tcl ... -o sqlite3.h
./jimsh .../tool/mkctimec.tcl
Overwriting ctime.c...
./jimsh .../tool/mkpragmatab.tcl
Overwriting pragma.h with new pragma table...
cc -g -o mkkeywordhash ... .../tool/mkkeywordhash.c
...
touch .target_source
cc -fPIC -O2 -g -DSQLITE_COVERAGE_TEST=1 -fprofile-arcs
  -ftest-coverage -c sqlite3.c
```

The evolved compiler-output rule removes the verbose `cp -f` file-listing block, which contains 25 lines enumerating over 100 source files copied to `tsrc/`, while preserving the semantically critical elements: tool-generation commands, build configuration messages such as "`Overwriting ctime.c...`", and—crucially—the final compilation command containing `-fprofile-arcs -ftest-coverage`. This final command confirms that gcov coverage instrumentation is correctly enabled. A positional truncation strategy would risk cutting from the end and losing this critical line, whereas TACO retains it through rule-guided filtering regardless of its position.

### K.3.3 Binary Disassembly (`objdump`)

In `vulnerable-secret` (episode 35), the agent disassembles the target binary to trace control flow and locate the flag-extraction logic. The `objdump_disassembly_rule`—which was reactively generated at episode 9—is applied to compress the assembly output.

**Raw terminal observation** (5,619 characters):

```
$ objdump -d vulnerable | sed -n '150,250p'
  4011e5:  31 f6            xor    %esi,%esi
  4011e7:  bf 11 00 00 00   mov    $0x11,%edi
  4011ec:  31 c0            xor    %eax,%eax
  4011ee:  e8 7d fe ff ff   call   401070 <signal@plt>
  4011f3:  31 c0            xor    %eax,%eax
  4011f5:  e8 96 fe ff ff   call   401090 <ptrace@plt>
  4011fa:  48 85 c0         test   %rax,%rax
  4011fd:  0f 88 9d 00 00 00 js  4012a0 <exit@plt+0x220>
  [... 90+ more instruction lines with hex bytes ...]
```

**Compressed output** (821 characters, ratio = 0.146):

```
$ objdump -d vulnerable | sed -n '150,250p'
  4011e5:  31 f6            xor    %esi,%esi
  4011e7:  bf 11 00 00 00   mov    $0x11,%edi
  4011ec:  31 c0            xor    %eax,%eax
  4011ee:  e8 7d fe ff ff   call   401070 <signal@plt>
  4011f5:  e8 96 fe ff ff   call   401090 <ptrace@plt>
```

The rule achieves an 85.4% reduction by stripping repetitive hex-dump lines that carry no symbolic significance, while preserving `call` instructions (which reveal API usage such as `signal` and `ptrace`), branch targets with symbol labels, and section headers. This is precisely the information the agent needs to trace the binary's anti-debugging logic and locate the flag-extraction code path. Across the full 101-episode trajectory, this single evolved rule saved 29,464 characters of redundant disassembly output—approximately 54% of all compression savings in this task. For a trajectory that consumed 2.4M input tokens, preventing this volume of low-value content from entering the context window directly contributes to the agent's ability to maintain coherent long-horizon reasoning.

These trajectories jointly show that TACO acts as a domain-adaptive semantic compressor rather than a generic truncation strategy. By initializing reusable rules from the Global Rule Pool and evolving new rules online for uncovered outputs, TACO preserves task-critical signals while filtering large volumes of repetitive terminal noise. This provides a concrete explanation for the performance gains observed in the main experiments.

# L Rule Format and Examples

## L.1 Rule Schema and Evolved Examples

Listing 1 presents a high-quality, self-evolved rule for `7z` archive extraction to illustrate the structural composition and semantic filtering capabilities of TACO. This autonomously generated rule captures complex domain-specific patterns, achieving maximum global confidence ($c_r^g = 1.0$) and high application counts without triggering any over-compression complaints.

Listing 1: Examples of high-quality, self-evolved compression rules autonomously discovered by TACO. These rules demonstrate fine-grained, domain-specific semantic filtering capabilities.

```
[
  {
    "rule_id": "seven_zip_extraction",
    "trigger_regex": "\\b7z\\b.*\\s+secrets\\.7z\\b",
    "description": "Compresses 7zip extraction output, preserving errors and success indicators while
        removing verbose file listing and progress information.",
    "keep_patterns": [
      "\\bError:",
      "\\bERROR:",
      "\\berror:",
      "Wrong password",
      "Can not open the file as",
      "Everything is Ok",
      "Extracting\\s+.*secret_file\\.txt",
      "Data Error in encrypted file"
    ],
    "strip_patterns": [
      "^\\s*[0-9]+ files?,",
      "^\\s*[0-9]+ folders?,",
      "^\\s*Size:\\s+",
      "^\\s*Compressed:\\s+",
      "^\\s*Processing archive:",
      "^\\s*--",
      "^\\s*$"
    ],
    "keep_first_n": 5,
    "keep_last_n": 5,
    "max_lines": null,
    "summary_header": "[7z extraction output compressed]",
    "priority": 42,
    "confidence": 1.0,
    "times_applied": 126,
    "times_complained": 0
  },
  {
    "rule_id": "nginx_setup_mod",
    "trigger_regex": "nginx|openssl|a2ensite|systemctl\\s+(start|enable|restart)\\s+nginx",
    "description": "Compresses Nginx and OpenSSL setup output, preserving errors, SSL generation
        notices, and service start/enable status",
    "keep_patterns": [
      "\\berror:",
      "\\bError:",
      "\\bERROR:",
      "Failed",
      "failed",
      "Traceback",
      "Generating.*cert",
      "server started",
      "server enabled",
      "service failed",
      "Job for nginx.service failed"
    ],
    "strip_patterns": [
      "Generating RSA private key",
      "writing new private key to",
      "-----BEGIN PRIVATE KEY-----",
      "-----END PRIVATE KEY-----",
      "Signature ok",
      "subject=",
      "Getting Private key",
      "\\s*\\*\\s+[^a-z].*"
    ],
    "keep_first_n": 5,
    "keep_last_n": 10,
    "max_lines": 30,
    "summary_header": "[Nginx/SSL setup output compressed]",
    "priority": 42,
    "confidence": 1.0,
    "times_applied": 122,
    "times_complained": 0
  }
]
```

### L.2 Construction of High-Quality Static Rules

The High-Quality Rule method is designed as a strong static compression comparison. We construct 200 structured rules from multiple complete TB 2.0 trajectories using LLM-assisted drafting followed by manual review. These rules cover common terminal-output patterns such as installation logs, compilation traces, test outputs, progress bars, directory listings, and long file views. During evaluation, the rules are fixed and do not adapt to the current task, agent behavior, or over-compression feedback. This makes them a non-trivial static comparison while preserving the key distinction from TACO's online rule selection and self-evolution.

### L.3 Critical Observation Detection

TACO uses deterministic, case-sensitive matching over the raw observation. Table 15 lists all 22 patterns. Any match bypasses learned-rule compression.

Table 15: Complete matching patterns for Critical observations.

| Category | Matching strings | Match type |
|---|---|---|
| Errors | `Error`, `ERROR` | Whole word |
| Failures | `Failed`, `FAILED`, `failed` | Whole word |
| Trace markers | `Exception`, `Traceback` | Whole word |
| Error prefixes | `error:`, `fatal:`, `E:` | Left boundary |
| Shell errors | `command not found`, `No such file or directory`, `Permission denied` | Substring |
| Python exceptions | `ModuleNotFoundError`, `ImportError`, `SyntaxError`, `NameError`, `TypeError`, `ValueError`, `KeyError`, `AttributeError`, `RuntimeError` | Substring |

Whole-word matching requires word boundaries on both sides; left-boundary matching requires one only before the string. Boundaries follow Python's default word-boundary semantics.

## M Compression-Quality Evaluation Protocol

### M.1 Evaluation Protocol

We collect all compression events from TACO runs with Qwen3-Coder-480B on TerminalBench 2.0 and draw a uniform random sample of 200 events. Each event corresponds to one raw terminal observation and the compressed observation actually provided to the agent. For each sampled event, we prompt a GPT-5.5-based judge with the task instruction, the issued shell command, the original uncompressed terminal observation, and the compressed output seen by the agent.

The judge scores three independent binary properties: `kept_critical`, which corresponds to whether potentially task-critical information is preserved; `removed_useless`, which corresponds to whether redundant or noisy content is removed; and `kept_useful`, which corresponds to whether useful non-critical context is retained. In Tab. 16, these are reported as "Critical info preserved", "Redundant/noisy removed", and "Useful context retained", respectively. We additionally report "Potential critical loss" as the complement of `kept_critical`. The full prompt is shown below.

Table 16: Observation compression quality.

| **Criterion** | **Pass Rate** |
|---|---|
| Critical info preserved | 96.0% |
| Redundant info removed | 81.0% |
| Useful info retained | 78.5% |

**COMPRESSION-QUALITY JUDGE PROMPT**

```
[SYSTEM]
You are an expert reviewer for terminal-output compression in
coding agents.
```

```
You will be given ONE compression OBSERVATION:
  - the TASK INSTRUCTION the agent is solving
  - the COMMAND that produced the terminal observation
  - the ORIGINAL terminal observation (before compression)
  - the COMPRESSED terminal observation (what the agent actually saw)

Judge THREE INDEPENDENT binary properties for this observation:

(1) kept_critical - Was critical info preserved in the compressed view?
    Critical = anything whose loss would BREAK the agent's ability to
    solve the task. e.g. error messages / stack traces, final test
    PASS/FAIL line, the actual numerical result asked for, the line
    the agent is grepping for, exit codes, the *body* of a newly
    written script / config file, the port number / URL / file path
    that's the deliverable, version strings.
    1 = critical info still readable in compressed view; 0 = lost.
    If there is no critical content in the original, default to 1.
    Note: if the missing content is something the agent itself just
    produced in its preceding message (e.g. the body of a
    `cat > f.py << EOF ... EOF` heredoc that the terminal merely
    echoes back), the agent already has it and you should score 1.

(2) removed_useless - Was noisy, useless info removed?
    Useless = bytes that carry no signal for the agent. e.g. ANSI
    escape codes, trailing empty newlines, apt's per-package
    "Setting up / Unpacking / Get:" middle lines, pip "Downloading
    ... 0% / 50% / 100%" progress bars, repeated bash empty-prompt
    polling lines, MOTD / login banner, duplicated headers.
    1 = a noticeable amount of noise was removed (or there was
        little noise to begin with AND compression didn't add noise);
    0 = compressed output is still mostly noise OR compression
        deleted real content instead of noise.

(3) kept_useful - Was useful (non-critical) context preserved?
    Useful = nice-to-have context that helps the agent reason
    without re-running a command. e.g. echoed full command line,
    key warnings, a few representative lines of a large list,
    dependency names, directory tree summary, first/last lines of
    a long log.
    1 = enough useful context kept that the agent can proceed
        without re-querying;
    0 = compression so harsh the agent likely needs to re-issue
        commands.

Critical vs Useful:
  - Critical = task FAILS or wrong answer if dropped.
  - Useful = task still solvable, but agent wastes turns or risks
             misinterpretation.

Be strict but fair. Each dimension is judged on its own.

[USER]
# Task instruction
{instruction}

# Command
`{command}`

# Original terminal observation ({orig_len} chars)
{original}

# Compressed terminal observation ({comp_len} chars, saved {saved})
{compressed}


# Output (one JSON object only, no extra prose, no markdown fence)
{"kept_critical": 0 or 1,
 "removed_useless": 0 or 1,
 "kept_useful": 0 or 1,
 "note": "<<=20 words explaining the judgment>"}
```

### M.2 Over-compression Analysis by Output Category

We manually inspect all eight events flagged as potential critical losses. Most are benign event-level flags: installation logs lose trailing confirmation lines but the agent continues after the shell prompt returns; the heredoc case removes only terminal-echoed text that already appears in the agent's previ-

Table 17: Manual inspection of the eight potential critical-loss events.

| Pattern | Count | Effect |
|---|---|---|
| Installation log confirmation folded | 5 | No visible behavior change |
| Long file inspection summarized | 2 | One triggers complaint |
| Heredoc echo omitted | 1 | Content already in history |

ous message. Among these eight flagged events, the only behaviorally affected case occurs when the agent inspects a recently written file with `cat pipeline_parallel.py`; the compressed view hides part of the file and the agent emits the intended `[Compression Complaint] need_full_output` signal. This complaint is used by TACO as feedback to make the corresponding rule more conservative in subsequent use. High-risk outputs. Structurally similar but informative outputs, including binary dumps, data tables, and hash or address lists, are susceptible to over-compression because repeated formatting does not imply redundant content. In extract-elf, a large deletion triggered rollback, restoring the original output and generating a more conservative replacement rule.

Corrective feedback. Across 500 manually reviewed compression events, complaint detection achieved 95.1% precision and 88.6% recall. All 41 complaints led to revisions involving 12 rules: 10 were rewritten into more conservative variants and continued to be reused, while two were permanently retired.

# N Prompt

This appendix presents the prompts used in TACO for task-level rule initialization and rule evolution. As described in the method section, TACO invokes LLMs in three cases: (1) selecting and refining candidate rules during task initialization, (2) generating a new rule for an uncovered high-output command, and (3) revising an overly aggressive rule into a more conservative replacement after an over-compression complaint.

## N.1 Prompt for Task-Level Rule Selection and Refinement

TACO uses a proposal-stage prompt to initialize the task-specific rule set $R_t$. When historical rules are available in the Global Rule Pool, the model selects, reuses, and refines relevant rules based on the current task context. When no suitable history is available, the model performs cold-start planning and directly proposes an initial rule set from scratch.

### (a) With-Cache Variant

This variant is used when the current task category already has historical rules in the Global Rule Pool. The model receives cached rules together with the task description and terminal state, and returns a structured plan containing reused, modified, and newly created rules.

**PROPOSAL PROMPT WITH CACHE**

```
You are a terminal observation compression strategy expert.

The system already has these baseline filters (you do NOT need to generate rules for these):
- ANSI escape code removal
- System login banner (Ubuntu MOTD) removal
- Empty command polling state handling

Below are historical compression rules from previous tasks. Select the ones
relevant to the current task, modify any that need adjustment, and create
new rules if needed.

Historical rules:
{cached_rules_json}

Current task description:
{instruction}

Task category: {task_category}
```

```
Current terminal environment (first 500 chars):
{terminal_state}

Instructions:
1. "selected_rule_ids": List rule_ids of rules to use AS-IS from the historical set
2. "modified_rules": For rules that are close but need adjustment, output the full
   modified rule with a NEW rule_id (e.g., original_id + "_mod")
3. "new_rules": For command types not covered by any historical rule, create new rules

Requirements:
- Only create rules for HIGH-OUTPUT commands (pip, apt, make, pytest, git, docker, etc.)
- Do NOT create rules for short-output commands (ls, cat, echo, pwd, cd)
- NEVER compress error output --- errors must always be fully preserved
- Be conservative: when in doubt, KEEP the line rather than strip it
- Total rules (selected + modified + new) should be 3-7

Output a single JSON object:
{
  "selected_rule_ids": ["id1", "id2"],
  "modified_rules": [
    {
      "rule_id": "string",
      "trigger_regex": "string",
      "description": "string",
      "keep_patterns": ["regex1"],
      "strip_patterns": ["regex1"],
      "keep_first_n": 5,
      "keep_last_n": 10,
      "max_lines": null,
      "summary_header": "[description]",
      "priority": 42
    }
  ],
  "new_rules": [
    {same format as modified_rules}
  ]
}

Output ONLY the JSON object, no other text.
```

### (b) Cold-Start Variant

This variant is used when no suitable historical rules are available for the current task category. Instead of selecting from cached rules, the model predicts which command types are likely to generate long terminal observation and directly constructs an initial rule set.

**PROPOSAL PROMPT NO CACHE**

```
You are a terminal observation compression strategy expert.

The system already has these baseline filters (you do NOT need to generate rules for these):
- ANSI escape code removal
- System login banner (Ubuntu MOTD) removal
- Empty command polling state handling

Given the task below, predict which terminal commands will produce long outputs,
and create compression rules for them.

Task description:
{instruction}

Task category: {task_category}

Current terminal environment (first 500 chars):
{terminal_state}

Requirements:
- Only create rules for HIGH-OUTPUT commands (pip, apt, make, pytest, git, docker, etc.)
- Do NOT create rules for short-output commands (ls, cat, echo, pwd, cd)
- NEVER compress error output --- errors must always be fully preserved
- Be conservative: when in doubt, KEEP the line rather than strip it
- Generate 3-7 rules

For each rule, provide:
```

```
- trigger_regex: regex to match the command string
- description: what this rule does
- keep_patterns: regex patterns for lines that MUST be preserved
- strip_patterns: regex patterns for lines safe to remove
- keep_first_n: always keep first N lines (default 5)
- keep_last_n: always keep last N lines (default 10)
- max_lines: cap on body lines after filtering (null = no cap)
- summary_header: text to show when lines are removed

Output a single JSON object:
{
  "rules": [
    {
      "rule_id": "string",
      "trigger_regex": "string",
      "description": "string",
      "keep_patterns": ["regex1"],
      "strip_patterns": ["regex1"],
      "keep_first_n": 5,
      "keep_last_n": 10,
      "max_lines": null,
      "summary_header": "[description]",
      "priority": 42
    }
  ]
}

Output ONLY the JSON object, no other text.
```

### N.2 Prompt for New Rule Generation

This prompt is used when a command produces a long terminal observation but no active rule matches that command type. In this case, TACO treats the output as an uncovered case and asks the model to generate a new compression rule that can be reused in subsequent steps.

**SPAWN NEW PROMPT**

```
You are a terminal observation compression rule expert.

The agent executed a command that produced a very long observation ({output_length} chars),
but no compression rule exists for this command type.

Command: {command}

Output (first 2000 chars):
{raw_output_head}

Output (last 500 chars):
{raw_output_tail}

Task context: {task_instruction}

Generate a compression rule for this type of command. The rule should:
1. Have a trigger_regex that matches this CATEGORY of command (not just this exact command)
2. Identify repetitive/progress/noise patterns in the output to strip
3. Preserve all error messages, results, and actionable information
4. Be conservative --- when in doubt, keep the line

Output a single JSON object with these fields:
{
  "rule_id": "string",
  "trigger_regex": "string",
  "description": "string",
  "keep_patterns": ["regex1", "regex2"],
  "strip_patterns": ["regex1", "regex2"],
  "keep_first_n": 5,
  "keep_last_n": 10,
  "max_lines": null,
  "summary_header": "[description of what was compressed]",
  "priority": 42
}

Output ONLY the JSON object, no other text.
```

### N.3 Prompt for Conservative Rule Update After Over-Compression Complaints

**Feedback detection.** The agent sets the optional `compression_feedback` field to `need_full_output` when compressed output lacks necessary information. This signal is detected by exact-value matching after trimming whitespace. Behavioral feedback is detected from repeated command strings after trimming and lowercasing, or inspection commands containing log- or error-related keywords. Feedback is attributed to the recorded compression-rule IDs, and TACO revises or replaces the corresponding rules with more conservative variants. Detection itself requires no additional classifier or LLM.

This prompt is used when subsequent agent behavior indicates that the compressed output was too aggressive, e.g., when the agent requests the full output, re-executes a command to recover missing details, or otherwise signals that critical information may have been removed. TACO then revises or replaces the responsible rule with a more conservative variant.

**SPAWN REPLACEMENT PROMPT**

```
You are a terminal observation compression rule expert.

The following rule compressed terminal observation too aggressively, causing the
agent to miss critical information.

Old rule (JSON):
{old_rule_json}

Command that was executed: {command}

Original terminal observation (first 2000 chars):
{raw_output_snippet}

Agent's feedback (what it complained about):
{agent_feedback}

Generate a NEW replacement rule that:
1. Keeps the same trigger_regex (targets same command type)
2. Is MORE CONSERVATIVE --- preserves more information
3. Stays SPECIFIC to this command type (don't make a generic "keep everything" rule)
4. Adds the missing information type to keep_patterns
5. Only strips content that is 100% guaranteed noise (progress bars, blank lines, etc.)
6. Uses a new rule_id (suggest: old_id + "_v2" or similar)

Output a single JSON object with these fields:
{
  "rule_id": "string",
  "trigger_regex": "string",
  "description": "string",
  "keep_patterns": ["regex1", "regex2"],
  "strip_patterns": ["regex1", "regex2"],
  "keep_first_n": 5,
  "keep_last_n": 10,
  "max_lines": null,
  "summary_header": "[description of what was compressed]",
  "priority": 42
}

Output ONLY the JSON object, no other text.
```

### N.4 Prompts for LLM-based Static Baselines

**Prompt for LLM-Gen Rules**

**LLM-GEN RULES PROMPT**

```
You are a terminal observation compression strategy expert.

The system already has these baseline filters (you do NOT need to generate rules for these):
- ANSI escape code removal
- System login banner (Ubuntu MOTD) removal
- Empty command polling state handling
```

```
Given the task below, predict which terminal commands will produce long outputs,
and create compression rules for them.

Task description:
{instruction}

Task category: {task_category}

Current terminal environment (first 500 chars):
{terminal_state}

Requirements:
- Only create rules for HIGH-OUTPUT commands (pip, apt, make, pytest, git, docker, etc.)
- Do NOT create rules for short-output commands (ls, cat, echo, pwd, cd)
- NEVER compress error output --- errors must always be fully preserved
- Be conservative: when in doubt, KEEP the line rather than strip it
- Generate 3-7 rules

For each rule, provide:
- trigger_regex: regex to match the command string
- description: what this rule does
- keep_patterns: regex patterns for lines that MUST be preserved
- strip_patterns: regex patterns for lines safe to remove
- keep_first_n: always keep first N lines (default 5)
- keep_last_n: always keep last N lines (default 10)
- max_lines: cap on body lines after filtering (null = no cap)
- summary_header: text to show when lines are removed

Output a single JSON object:
{
  "rules": [
    {
      "rule_id": "string",
      "trigger_regex": "string",
      "description": "string",
      "keep_patterns": ["regex1"],
      "strip_patterns": ["regex1"],
      "keep_first_n": 5,
      "keep_last_n": 10,
      "max_lines": null,
      "summary_header": "[description]",
      "priority": 42
    }
  ]
}

Output ONLY the JSON object, no other text.
```

### Prompt for LLM Summarization

**DEFAULT TERMINAL OUTPUT COMPRESSION PROMPT**

```
# Role
You are the Terminal Output Safe Compressor, responsible for compressing redundant terminal output
     without affecting AI Agent decision-making.

# Core Principle (MUST strictly follow)
Content can ONLY be compressed when the following condition is met:
"If this content is compressed from the context, the Agent will still make the exact same correct
     decision, and the task result will remain unchanged."

# Task Context (IMPORTANT --- Use this to understand what information is critical)
{task_instruction}

# Input Format
1. TASK: The task the Agent is working on (shown above)
2. COMMAND: The shell command executed by the Agent
3. RAW_OUTPUT: The raw terminal output

# Safe Compression Rules

## Content that CAN be safely compressed
1. Progress bars and download statistics (percentage, speed, ETA)
2. Git transfer statistics (object enumeration, compression numbers)
```

```
3. System banners and copyright notices (Ubuntu welcome, MOTD)
4. Repetitive log lines with same pattern
5. ANSI color codes and escape sequences

## Content that MUST NEVER be compressed
1. Any error messages --- preserve completely
2. Actual command output results (ls, cat, head, tail output)
3. Interactive prompts (yes/no, password prompts)
4. Path and filename information
5. Version numbers and package names
6. Test results (passed/failed counts)
7. Port numbers, URLs, IP addresses
8. Analysis command output --- CRITICAL: Output from these commands must be preserved COMPLETELY:
   - diff, cmp, comm --- Every line shows critical differences
   - hexdump, xxd, od --- Every byte matters for binary analysis
   - cat -A, cat -v, cat -e --- Shows invisible characters that are crucial
   - strings, objdump, readelf --- Binary inspection results
   - strace, ltrace --- System call traces
   - md5sum, sha*sum --- Checksum values must be exact
9. Program execution results --- When running ./program or similar:
   - ALWAYS preserve: The final output/result (numbers, calculation results, "success"/"failed")
   - ALWAYS preserve: Debug output like "values[x] = y", "result = z"
   - ALWAYS preserve: Any single-line output that could be the program's answer
   - CAN compress: Progress bars (0%...100%), repetitive status updates
   - Example: Debug: x=5, y=10\n15 -> Keep "Debug: x=5, y=10\n15" (the "15" is the result!)

# Output Format (Strict JSON)
{
  "is_safe_to_compress": boolean,
  "has_error": boolean,
  "status": "success" | "failed" | "running",
  "summary": "string"
}

# Command
{command}

# Terminal Output
{terminal_output}
```

## O Task-level Regression Analysis

We analyze the trajectories from five TACO runs with Qwen3-Coder-480B on TB 2.0. Among the evaluated tasks, 16 had a lower mean success rate than the baseline. Table 5 classifies a task as solved if its success rate is at least 50%; under this definition, two tasks transition from solved under the baseline to unsolved under TACO. The 16 tasks also include those whose success rates decrease while their solved/unsolved status remains unchanged.

Inspection of the removed content and subsequent feedback provides the following illustrative cases:

(1) Blank-line removal. In two failed trials of constraints-scheduling, the task with the largest decrease in success rate, the removed content consisted entirely of blank lines in ls -la output.

(2) Redundant terminal-echo removal. In another failed trial of constraints-scheduling, compression removed terminal echoes of content that the agent had written to a file using a heredoc in the preceding step. These echoes duplicated content already present in the agent's context. Across the three failed trials, compression removed 32, 146, and 31 characters, respectively, without removing any task-relevant information returned by the environment. Similar removal of blank lines or redundant terminal echoes was observed for regex-log and custom-memory-heap-crash.

(3) Larger deletions followed by corrective feedback. In extract-elf, a larger deletion triggered the complaint-and-rollback mechanism, which restored the original output and immediately generated a more conservative replacement rule.

## NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state the main contributions of TACO: a plug-and-play, self-evolving terminal agent compression framework that improves task performance and token efficiency across diverse agents and benchmarks. These claims align with the theoretical motivation and the experimental results presented in Sections 2 and 4.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We include the limitation of our work in Section A

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: Our work dose not include theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: These details are provided in the Section 3

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Our code, setup instructions, and example evaluation commands are publicly available at `https://github.com/multimodal-art-projection/TACO`.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: All experimental settings and details are provided in Section 3.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We report averages over five evaluation runs and provide run-level standard deviations for the main accuracy results in Appendix E.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Details on compute resources used for experiments are provided in Section C.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research fully adheres to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Please see Section D.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: No high-risk models or datasets are used in this work.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All external datasets and models used are properly cited and licensed.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The released code is documented in the repository, including installation, usage, and configuration instructions.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [Yes]

    Justification: Limited human annotation is used only for terminal-output artifacts; details are provided in Appendix B.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The work does not involve human-subject research; details are provided in Appendix B. Human effort is limited to annotating terminal-output artifacts and manually inspecting compression cases.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs are used as part of the TACO framework for generating and refining compression rules, which are core to the method.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.