COMPACTIONRL: REINFORCEMENT LEARNING WITH CONTEXT COMPACTION FOR LONG-HORIZON AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon agentic LLMs are increasingly limited by finite context windows, as extended interaction trajectories can exceed the maximum context length before a task is completed. Context compaction offers a natural solution by summarizing previous interaction states and continuing the rollout under a compressed context, but incorporating compaction into reinforcement learning remains underexplored. We propose CompactionRL, a reinforcement learning strategy to train long-horizon agentic LLMs with context compaction. Our approach jointly optimizes *task execution and summary generation* with token-level loss normalization and cross-segment generalized advantage estimation. This design enables the LLM agents to learn from compacted long-horizon trajectories. We train CompactionRL on top of open models and observe consistent performance gains on agentic coding tasks. CompactionRL enables the open GLM-4.5-Air model (106B-A12B) to achieve Pass@1 scores of 66.4% on SWE-bench Verified and 26.2% on Terminal-Bench 2.0, exceeding the base model under inference-time compaction by 6.6 and 4.9 points, respectively. Built upon GLM-4.7-Flash (30B-A3B), CompactionRL improves Pass@1 by 5.5 and 6.7 points against the base model, reaching 56.0% on SWE-bench Verified and 20.2% on Terminal-Bench 2.0.

1 INTRODUCTION

Large language model (LLM) agents are increasingly applied to long-horizon interactive tasks, such as software engineering, terminal-based problem solving, and web interaction (Yao et al., 2023; Nakano et al., 2021; Jimenez et al., 2023). These tasks require agents to repeatedly reason, act, observe environment feedback, and revise their plans over many steps. As the interaction proceeds, the accumulated history, including tool outputs, intermediate reasoning, error messages, and partial solutions, can exceed the model's finite context window. Although longer-context models alleviate this issue, scaling context length alone is costly and does not fully solve degraded utilization over long sequences (Beltagy et al., 2020; Dao et al., 2022; Liu et al., 2024; Hsieh et al., 2024). Thus, long-horizon RL for LLM agents requires mechanisms that allow training to proceed under a fixed context budget.

Context compaction provides a natural way to address this limitation. When the history approaches the context limit, earlier interaction history can be summarized into a shorter state, after which the agent resumes from the compacted summary and a small amount of recent context. Related ideas have been explored in language-agent memory, reflection, and long-horizon interaction systems (Shinn et al., 2023; Park et al., 2023; Sun et al., 2025). However, prior uses of compaction are mostly treated as inference-time heuristics or external memory operations. In RL training, compaction has a more fundamental role: once a summary replaces the original history, it determines what information is available for all subsequent actions. Therefore, task success depends not only on the execution policy, but also on the quality of the compaction policy.

Context compaction introduces additional challenges for asynchronous agentic RL. First, differences in compaction frequency and subsequent execution length increase variation in rollout generation time. Group-relative methods such as GRPO must wait for the slowest rollout in each group to finish before computing advantages, increasing policy lag and off-policy effects (Hou et al., 2026). Second, jointly optimizing task execution and summarization requires including both execution and summary segments in training despite summary segments being substantially shorter, creating

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

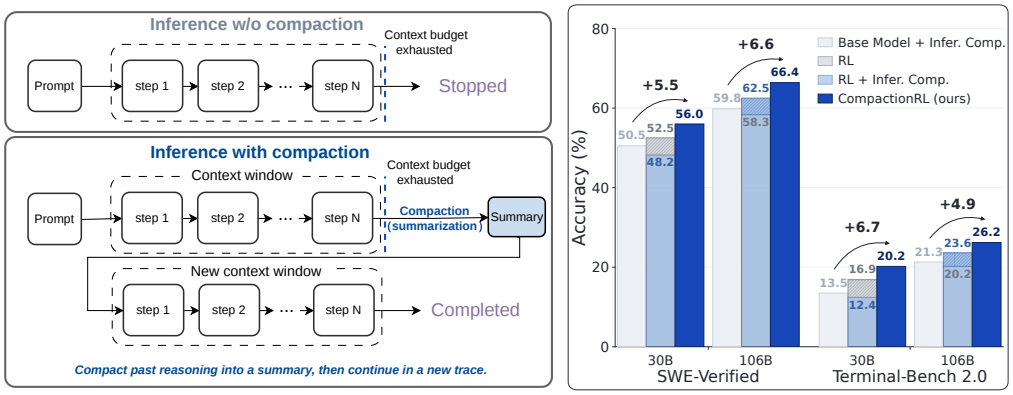


Figure 1: Left: Context compaction allows execution to continue under a fixed context window by initiating a new trace from a compressed summary once the context budget is exhausted. Without compaction, execution terminates upon exhaustion of the context budget. Right: Performance on SWE-bench Verified and Terminal-Bench 2.0 across the base model with inference compaction, standard no-compaction RL, RL with inference compaction, and CompactionRL. "Base Model + Infer. Comp." denotes the base model evaluated with context compaction. "RL" denotes standard RL trained and evaluated without compaction, while "RL + Infer. Comp." uses the same RL checkpoint but enables context compaction during evaluation.

a severe length imbalance that can lead to disproportionate weighting under segment-level loss. Third, computing GAE independently within each segment places the shared terminal reward at local segment boundaries, artificially shortening its temporal distance from earlier actions and summaries and distorting temporal credit assignment.

In this work, we propose CompactionRL, a PPO-based reinforcement learning framework for building long-horizon agentic LLMs with trainable context compaction, as shown in Figure 1. CompactionRL incorporates compaction into rollout sampling, reconstructing the agent context from a summary once context budget is exhausted. Its core strategy is to jointly optimize task execution actions and summarization actions under the same final task reward. This enables RL training horizon to extend beyond the peak context length while keeping the actual working context budget fixed. Our contributions are summarized as follows:

- We bring context compaction into long-horizon agentic RL training. We adopt single-rollout PPO to avoid waiting for rollout groups, which is a problem more severe in compaction-aware training, and reduce the associated policy lag in asynchronous training, while using a learned critic to estimate token-level advantages for execution and summarization segments.
- We develop CompactionRL to jointly train trajectory generation and summary generation with a shared task-level reward, making summarization a learned part of the model rather than an inference-time heuristic. To handle compacted rollouts, we use token-level loss normalization to reduce length-induced weighting bias and cross-segment GAE to preserve temporal credit assignment across compaction boundaries.
- CompactionRL achieves consistent gains on long-horizon agentic coding tasks. Compared with the base model with inference-time compaction, CompactionRL improves GLM-4.7-Flash by 5.5 points on SWE-bench Verified and 6.7 points on Terminal-Bench 2.0. It also improves GLM-4.5-Air by 6.6 and 4.9 points on the two coding benchmarks, respectively. These results show that RL-trained compaction can extend the effective training horizon of agentic LLMs without increasing the maximum working context length.

2 Related Work

Context Compaction for Long-Horizon Agents. Long-horizon language agents must maintain task-relevant state across extended sequences of reasoning, tool calls, and environment feedback. A standard ReAct-style agent appends the full interaction history to the prompt (Yao et al., 2023),

but this makes the context grow with the trajectory and can reduce both efficiency and effective information use in long contexts (Liu et al., 2024). Prior work has addressed this issue through longer-context modeling and evaluation (Bai et al., 2024; Hsieh et al., 2024), prompt or observation compression (Jiang et al., 2023b;a), and explicit memory or reflection mechanisms (Packer et al., 2023; Shinn et al., 2023; Park et al., 2023; Zhou et al., 2025), showing that compressing or externalizing history can extend the effective horizon of LLM agents. Recent work has begun to incorporate context management into agent training. SUPO (Lu et al., 2025) and Memex(RL) (Wang et al., 2026) introduce context management into multi-turn RL and optimize with GRPO, without considering context state differences across compaction segments for advantage calculation; ReSum (Wu et al., 2025) trains on segmented trajectories without optimizing the summarizer while Yi et al. (2026) and Wu et al. (2026) train only the context manager. However, how to jointly train long-horizon coding agents with both execution and compaction segments remains underexplored.

Reinforcement Learning for Language Models. Reinforcement learning has become a central technique for improving language models with outcome feedback(Uesato et al., 2022). Early RLHF systems commonly adopt PPO with an explicit value function (Schulman et al., 2017; Ouyang et al., 2022), while recent reasoning-oriented LLM training often favors critic-free or group-relative objectives such as RLOO, GRPO, and DAPO (Ahmadian et al., 2024; Shao et al., 2024; DeepSeek-AI, 2025; Hou et al., 2025b; Yu et al., 2025; Zheng et al., 2025) that have shown strong performance on verifiable-reward tasks such as mathematical reasoning and coding(Hou et al., 2025a; Wei et al., 2025; Xu et al., 2025). At the same time, recent studies show that critic-based methods can still be highly effective when value learning is carefully designed. VC-PPO (Yuan et al., 2025) improves PPO through value pretraining and modified advantage estimation and VAPO (Yue et al., 2025) demonstrates that a value-based RL framework can achieve strong and stable reasoning performance. SAO (Hou et al., 2026) adopts single-rollout sampling to reduce policy lag caused by waiting for complete rollout groups in asynchronous agentic RL. These studies motivate critic-based optimization for long-horizon agents, supporting both state-dependent advantage estimation and training without group-relative baselines.

3 THE COMPACTIONRL METHOD

We present CompactionRL, a reinforcement learning framework for long-horizon language agents under a fixed context budget. The central idea is to make context compaction part of rollout sampling: when the interaction history approaches the context limit, the agent produces a compact summary of the previous trajectory and then resumes from a reconstructed context containing the summary and a short history tail of recent interaction. The summary is generated by the trainable policy and is optimized together with ordinary task-execution actions under the final task reward. Figure 2 gives an overview of the framework.

3.1 TRAINABLE CONTEXT COMPACTION

Compaction during Rollout Sampling. We represent the interaction history as

$$h_t = (s, u, z_1, \ldots, z_t), \qquad z_i = (a_i, o_i), \tag{1}$$

where s is the system prompt, u is the original user instruction, a_i is the assistant response at step i, and o_i is the corresponding environment observation. We treat each assistant-observation pair z_i as an atomic step, so that tool calls and their feedback are not separated by compaction.

Let C denote the context budget and $|h_t|$ the token length of the current history, Compaction is triggered when the remaining context budget falls below a threshold T_{comp}:

$$C - |h_t| < T_{\text{comp}}. \tag{2}$$

Once triggered, we append a fixed summarization instruction q_{sum} to the current history and sample a summary S_t from the policy, where q_{sum} asks the model to preserve information necessary for continuing the task, including the original goal, completed actions, important observations, unresolved errors, current state, and plausible next steps.

After summary generation, the rollout continues from a reconstructed context

$$\bar{h}_t = s \oplus u_{\text{resume}}(S_t) \oplus (z_{t-k+1}, \ldots, z_t), \tag{3}$$

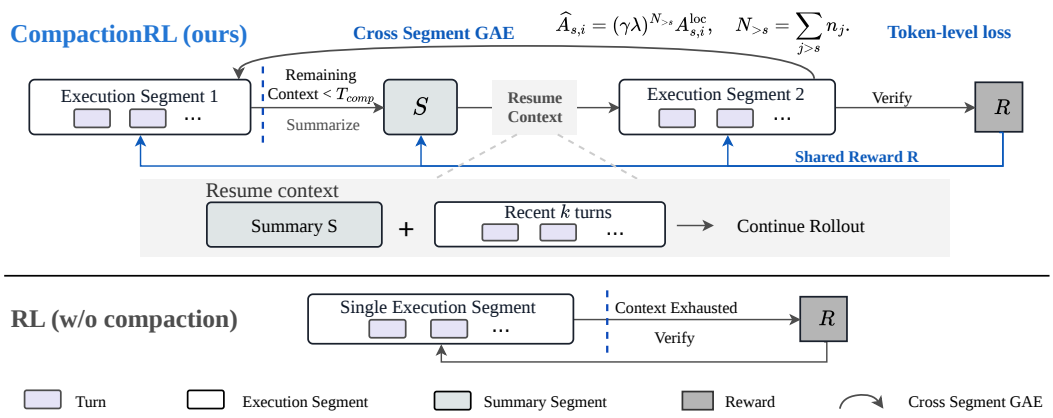


Figure 2: Overview of CompactionRL. During rollout collection, the agent interacts with the environment under a fixed context budget. When the remaining context budget falls below a compaction threshold T_{comp}, the policy generates a summary and resumes from the summary and recent turns. All execution and summary segments are optimized and share the trajectory reward R. During training, we propagate this reward across segment boundaries by correcting the local GAE advantage $A^{\text{loc}}_{s,i}$ with the number of optimized tokens in subsequent segments, $N_{>s}$, yielding the cross-segment advantage $\widehat{A}_{s,i}$. Standard RL without compaction optimizes each rollout as a single execution segment.

Table 1: Effect of summarization quality under context compaction. The execution agent is fixed to GLM-4.7-Flash, while tested with various summary agents including Qwen3.5-27B(Qwen Team, 2026) and Qwen3-30B-A3B(Yang et al., 2025). SWE-Verified Acc. denotes pass@1(%) on SWE-bench Verified; Summary Count / Trace reports the average number of summaries triggered per trace.

Execution Agent	Summary Agent	SWE-Verified Acc.	Summary Count / Trace
GLM-4.7-Flash	Qwen3.5-27B	55.5	1.010
GLM-4.7-Flash	GLM-4.7-Flash	50.5	1.075
GLM-4.7-Flash	Qwen3-30B-A3B	49.0	1.126

where $u_{\text{resume}}(S_t)$ is a fixed template containing the generated summary. The final term keeps the most recent k turns while the summary covers only the history preceding the k turns. We use $k = 2$ by default and reduce k when necessary to ensure that $\bar{h}_t$ fits within the context budget. This reconstruction preserves long-range information through S_t while retaining the most recent environment state exactly.

Training Segments. Compaction naturally partitions a rollout into a sequence of trainable segments,

$$\tau = (\sigma_1, \ldots, \sigma_K), \tag{4}$$

where each segment is either an execution segment or a summarization segment. An execution segment contains tokens produced for solving the task, while a summarization segment contains the summary tokens generated before context reconstruction. Summary tokens are sampled from the same trainable policy and are included in the RL objective.

Each rollout receives a final task reward $R(\tau)$ determined by task correctness. We assign this rollout-level reward to all trainable segments from the same rollout. We do not introduce a separate summary-quality reward, since hand-designed summary metrics may not reflect which details are useful for continuation.

Effect of Summarization Quality. Compaction makes task success sensitive to the quality of the summary. A summary that omits a crucial file path, error message, failed command, or partial solution can make subsequent actions ineffective even when the execution policy itself is strong. To quantify this effect, we fix the execution agent and vary only the summary agent.

Table 1 shows that changing only the summary agent leads to a large difference in final task performance. The best summarizer improves SWE-Verified accuracy from 49.0 to 55.5, a gain of 6.5 absolute points, while also triggering slightly fewer compactions per trace. This result suggests that compaction is a performance-critical decision process rather than a passive preprocessing step. We therefore train summary generation jointly with task execution.

3.2 Optimization under Compacted Rollouts

Single-Rollout Optimization. Compaction increases variation in rollout generation time since rollouts differ in compaction frequency and subsequent execution length. Group-relative advantage estimation requires completed rollouts to wait for the slowest member of the group, while asynchronous policy updates increase the mismatch between rollout and training policies during this waiting period. Following SAO (Hou et al., 2026), we adopt PPO with one rollout per prompt to avoid this additional waiting and reduce the associated off-policy effects.

Moreover, resumed segments condition on different compacted histories. A learned critic provides context-dependent value estimates for token-level advantage estimation, whereas rollout-level group normalization does not explicitly account for these intermediate states differences.

Token-Level Loss for Length Imbalance. Since the length of agent trajectories and summarization varies significantly, we choose to use token-level loss instead of sequence-level loss to overcome the severe imbalance. Let $\mathcal{M}$ denote the set of optimized assistant-token positions in a training batch. For token $y_{s,i}$ in segment σ_s, conditioned on context $x_{s,i}$, let $\rho_{s,i}(\theta)$ denote the importance sampling ratio of the current policy π_θ to the rollout policy $\pi_{\theta_{\text{old}}}$. We optimize the clipped PPO objective

$$\mathcal{L}_\pi = -\frac{1}{|\mathcal{M}|} \sum_{(s,i)\in\mathcal{M}} \min\left(\rho_{s,i}(\theta)\widehat{A}_{s,i}, \text{clip}\left(\rho_{s,i}(\theta), 1-\epsilon, 1+\epsilon\right)\widehat{A}_{s,i}\right), \tag{5}$$

where ϵ is the clipping threshold and $\widehat{A}_{s,i}$ is the advantage estimate described below.

The loss computation is performed over generated tokens rather than over samples. This is important because compaction introduces heterogeneity in both segment count and segment length. Segment-level averaging would make a rollout with more compaction events contribute more strongly than a rollout with fewer segments, even when both receive the same task reward. Token-level normalization removes this segment-count bias and gives each trainable token equal weight.

Cross-Segment Generalized Advantage Estimation. PPO (Schulman et al., 2017) uses a learned critic V_ϕ to estimate expected discounted returns. The temporal-difference (TD) residual measures the discrepancy between the one-step bootstrapped return and the current value estimate:

$$\delta_t = r_t + \gamma V_\phi(x_{t+1}) - V_\phi(x_t), \tag{6}$$

where x_t is the current context, r_t is the reward, and γ is the discount factor. Generalized advantage estimation (GAE) (Schulman et al., 2015) computes token-level advantages by accumulating subsequent TD residuals:

$$A_t^{\text{GAE}} = \sum_{\ell=0}^{T-t} (\gamma\lambda)^\ell \delta_{t+\ell}, \tag{7}$$

where T is the number of optimized tokens and λ is the trace parameter. This accumulation propagates delayed task feedback to earlier tokens while discounting more distant residuals.

Compacted rollouts also create a temporal credit-assignment issue. Since each segment is optimized independently, a naive segment-level GAE computation places the shared terminal reward at the end of every segment. For earlier segments, this makes the final task outcome appear artificially closer than it is in the original rollout, over-crediting actions or summaries that occur far before task completion.

For a segment σ_s with n_s optimized tokens, the local GAE estimator is

$$A_{s,i}^{\text{loc}} = \sum_{\ell=0}^{n_s-i} (\gamma\lambda)^\ell \delta_{s,i+\ell}, \qquad \delta_{s,i} = r_{s,i} + \gamma V_\phi(x_{s,i+1}) - V_\phi(x_{s,i}), \tag{8}$$

where i indexes tokens within the segment. Let $N_{>s} = \sum_{j>s} n_j$ be the number of optimized tokens generated after segment σ_s in the same rollout. We apply a trajectory-position correction

$$\widehat{A}_{s,i} = (\gamma\lambda)^{N_{>s}} A^{\mathrm{loc}}_{s,i}. \tag{9}$$

This correction discounts earlier segments according to the number of subsequent trainable tokens before rollout termination. In particular, if the terminal task reward appears at the last token of each independently optimized segment, then the reward term for token (s, i) receives discount

$$(\gamma\lambda)^{N_{>s}+n_s-i}, \tag{10}$$

which matches its distance to the final outcome in the concatenated compacted rollout.

Note that this is an approximation rather than an exact reconstruction of full-trajectory GAE. Since an exact reconstruction of full-trajectory GAE would require recursively propagating TD residuals from later to earlier compaction segments, introducing cross-segment dependencies that preclude parallel processing within a batch, we use the trajectory-position correction above instead of exact reconstruction to preserve training efficiency.

4 Experiments

4.1 Experimental Setup

Training Details. We conduct experiments with two models at different parameter scales: GLM-4.7-Flash and GLM-4.5-Air-SFT. The latter is obtained by supervised fine-tuning GLM-4.5-Air (GLM et al., 2025) on trajectories generated by GLM-4.7. For each model, we initialize the corresponding critic from the same model checkpoint and perform 50 steps of value pretraining on the training dataset before reinforcement learning. We use open-source training data from SWE-Dev (Wang et al., 2025). For the training of CompactionRL, we use slime (Zhu et al., 2025), an open-source asynchronous RL framework, for reinforcement learning training.

For RL training, we employ a global batch size of 128, sample one rollout per prompt, and use context budgets of 64k for GLM-4.7-Flash and 80k for GLM-4.5-Air-SFT. The policy is optimized with Adam using a learning rate of 2×10^{-6}, while the critic is trained with a learning rate of 3×10^{-6}. For advantage estimation, we set the reward discount factor $\gamma = 1$ and adopt length-adaptive GAE (Yue et al., 2025) with $\lambda = 1 - \frac{1}{\alpha l}$ and $\alpha = 1.5$, where l denotes the response length. To ensure that the critic keeps pace with policy updates, we perform two value model updates and one policy update for each batch, facilitating accurate advantage estimation. Each assistant response is limited to 10,240 tokens and compaction is triggered when the remaining context budget falls below 10,240 tokens, with at most three compaction operations per rollout. Detailed model configurations and compute resources are provided in Appendix A.2.

Evaluation Setting. We evaluate CompactionRL on SWE-bench Verified (Jimenez et al., 2023) and Terminal-Bench 2.0 (Merrill et al., 2026), using 200 randomly sampled tasks from SWE-bench Verified and the full Terminal-Bench 2.0 set, and report Pass@1 accuracy. The complete list of sampled tasks is provided in Appendix A.1. All evaluations are conducted in the Harbor (Harbor Framework Team, 2026) environment with the Terminus-KIRA (KRAFTON AI & Ludo Robotics, 2026) agent scaffold. Evaluation hyperparameters include: top-$p = 1.0$, temperature $= 1.0$, up to 250 interaction turns and at most three compaction operations per trajectory.

4.2 Main Results

Table 2 compares the base models, standard no-compaction PPO , CompactionRL (w/o sum. loss), and the full CompactionRL under single-window or compacted evaluation settings on SWE-bench Verified and Terminal-Bench 2.0. CompactionRL (w/o sum. loss) enables compaction during RL but masks the loss on summary-response turns, separating the effect of training on compacted histories from that of directly optimizing the summarizer.

CompactionRL achieves the best performance across both model scales and benchmarks. Inference-time compaction generally improves the base models by extending their effective context budget. Standard PPO improves single-window execution, but this benefit does not transfer consistently

Table 2: Main results on SWE-bench Verified and Terminal-Bench 2.0. We report Pass@1 accuracy (%). Results are reported as mean ± standard deviation over three evaluation runs. Comp. Training and Comp. Infer. indicate whether context compaction is enabled during RL training and evaluation, respectively. For each model, "w/ Inference Compaction" uses the same checkpoint as the preceding row but enables compaction during evaluation. CompactionRL (w/o sum. loss) retains compaction during both training and inference but masks summary-response tokens in the RL loss. Compacted evaluation retains the peak working context length and allows up to three compactions, yielding an effective budget of up to $4\times$ the peak length.

	Peak Len.	Comp. Training	Comp. Infer.	SWE-bench Verified	Terminal-Bench 2.0
Public reports / baselines					
gpt-oss-120b	128k	–	✗	62.0	18.7
Qwen3.5-35B-A3B	64k	–	✓	58.0	27.0
Qwen3-Coder-480B-A35B-Instruct	256k	–	✗	66.5	23.9
Qwen3-Coder-30B-A3B-Instruct	256k	–	✗	51.9	14.6
Qwen3-235B-A22B-Instruct-2507	256k	–	✗	45.2	13.5
Qwen3-30B-A3B-Instruct-2507	256k	–	✗	25.2	5.34
Terminus-KIRA scaffold					
GLM-4.7-Flash (30B-A3B)	64k	–	✗	$47.3_{\pm 2.7}$	$14.6_{\pm 1.6}$
w/ Inference Compaction	64k	–	✓	$50.5_{\pm 1.2}$	$13.5_{\pm 0.9}$
+ RL (w/o compaction)	64k	✗	✗	$52.5_{\pm 2.5}$	$16.9_{\pm 2.4}$
w/ Inference Compaction	64k	✗	✓	$48.2_{\pm 0.5}$	$12.4_{\pm 0.9}$
+ CompactionRL (w/o sum. loss)	64k	✓	✓	$54.5_{\pm 1.2}$	$12.4_{\pm 0.0}$
+ CompactionRL (ours)	64k	✓	✓	$\mathbf{56.0}_{\pm \mathbf{0.4}}$	$\mathbf{20.2}_{\pm \mathbf{0.9}}$
GLM-4.5-Air (106B-A12B)	80k	–	✗	$57.8_{\pm 2.1}$	$18.0_{\pm 2.8}$
w/ Inference Compaction	80k	–	✓	$59.8_{\pm 1.9}$	$21.3_{\pm 1.6}$
+ RL (w/o compaction)	80k	✗	✗	$58.3_{\pm 0.6}$	$20.2_{\pm 3.7}$
w/ Inference Compaction	80k	✗	✓	$62.5_{\pm 1.1}$	$23.6_{\pm 0.9}$
+ CompactionRL (w/o sum. loss)	80k	✓	✓	$64.5_{\pm 2.4}$	$21.7_{\pm 2.8}$
+ CompactionRL (ours)	80k	✓	✓	$\mathbf{66.4}_{\pm \mathbf{0.7}}$	$\mathbf{26.2}_{\pm \mathbf{2.1}}$

Note. In all our results, SWE-bench Verified is evaluated on a random 200-instance subset, while public baselines are reported on the full benchmark. The public baselines are taken from model cards or public reports and are included for reference only, as their agent scaffolds may differ from ours.

to compacted inference, where it either degrades performance or yields smaller gains than CompactionRL. Moreover, the consistent improvements of CompactionRL over CompactionRL (w/o sum. loss) across both model scales and benchmarks demonstrate the benefit of directly optimizing summarization during compaction-aware training. Overall, these results indicate that CompactionRL effectively extends the usable context horizon through trainable compaction.

4.3 Ablation Studies

Effect of PPO. To evaluate the role of PPO in compaction-aware training, we compare CompactionRL with a GRPO-based baseline implemented following SUPO (Lu et al., 2025). Both methods jointly optimize task execution and summary generation, but SUPO computes group-relative advantages at the rollout level and assigns them uniformly to all compaction segments, whereas CompactionRL uses a learned critic to estimate token-level advantages. As shown in Table 3, CompactionRL outperforms the SUPO baseline by 4.6 and 4.9 percentage points on SWE-bench Verified and Terminal-Bench 2.0, respectively, demonstrating the effectiveness of critic-based PPO training.

Effect of Token-Level Loss and Cross-Segment GAE. We further ablate the two optimization components designed for compacted trajectories using GLM-4.5-Air-SFT: token-level loss normalization and cross-segment GAE. As shown in Table 3, removing either component degrades

Table 3: Ablation study on PPO based RL, token-level loss normalization, and cross-segment GAE using GLM-4.5-Air-SFT. All variants are evaluated under the compaction enabled 80k×4 setting. We report Pass@1 accuracy as mean ± standard deviation on SWE-bench Verified and Terminal-Bench 2.0.

	SWE-bench Verified	Terminal-Bench 2.0
GLM-4.5-Air (106B-A12B)	59.8±1.9	21.3±1.6
+ SUPO	61.8±1.8	21.3±0.9
+ CompactionRL	**66.4±0.7**	**26.2±2.1**
− w/o token-level loss	60.0±1.2	21.3±2.4
− w/o cross-segment GAE	63.0±1.4	22.5±2.4

Table 4: Comparison with extended-context no-compaction RL using GLM-4.5-Air (106B-A12B). We report Pass@1 accuracy (%) as mean ± standard deviation over three evaluation runs. As inference time strategies, Comp.(×4) enables up to three compaction operations with an 80k peak context, while Long uses a 160k non-compacted context window as a sufficiently long context reference.

	Train Budget	SWE-bench Verified		Terminal-Bench 2.0	
		Comp.(×4)	Long	Comp.(×4)	Long
GLM-4.5-Air (106B-A12B)	–	59.8±1.9	59.5±1.6	21.3±1.6	20.6±2.1
+ RL (w/o compaction)-160k	160k	62.8±2.8	64.0±0.8	23.2±2.8	22.1±1.4
+ CompactionRL (ours)	80k×4	**66.4±0.7**	61.8±1.5	**26.2±2.1**	24.2±1.4

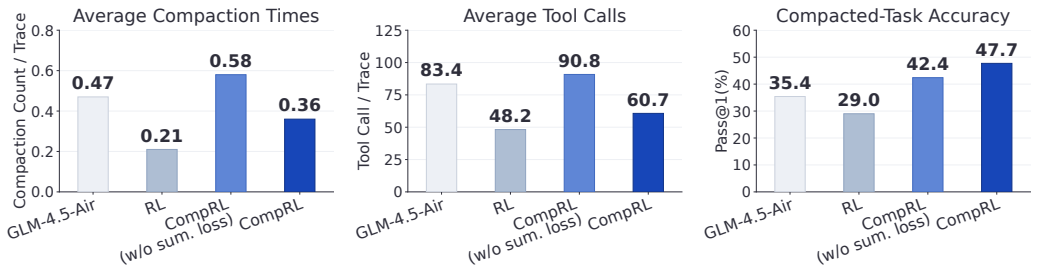
performance, confirming that both loss weighting and cross-segment credit assignment are important for compaction-aware RL. A larger degradation is observed when token-level loss normalization is removed, suggesting that correcting the optimization bias induced by variable segment counts and lengths is particularly critical.

Effect of Extended-Context. We further compare CompactionRL with standard no-compaction RL using an extended context window of 160k, which simulates access to a sufficiently long context that can retain the full trajectory without compaction-induced information loss. As shown in Table 4, under 80k×4 compacted evaluation, CompactionRL outperforms the extended-context baseline evaluated with either 80k×4 compaction or a 160k non-compacted context on both benchmarks. This demonstrates the effectiveness of trainable compaction relative to directly increasing the training context window.

4.4 Analysis

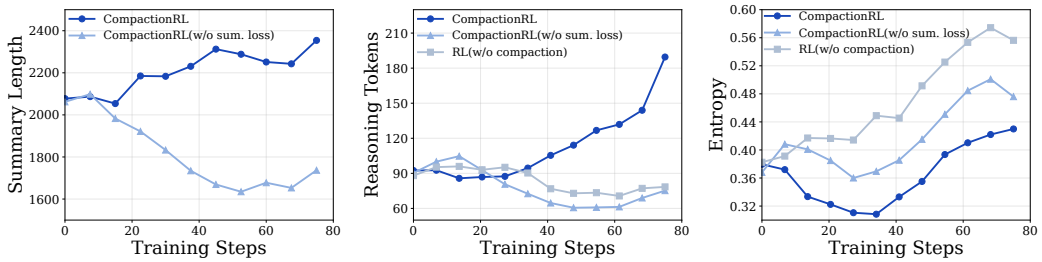
Compaction Behavior. Figure 3 analyzes compaction behavior in the GLM-4.5-Air setting under 80k×4 evaluation. Figure 3(a,b) shows that CompactionRL requires fewer compactions and tool calls than GLM-4.5-Air and CompactionRL (w/o sum. loss) but more than standard no-compaction RL. This indicates that CompactionRL learns both to exploit the extended context horizon enabled by compaction and a higher interaction efficiency, suggesting that the improvement is not attributable to longer interaction traces alone; rather, trained summaries help retain task-relevant information and reduce redundant re-exploration after compaction. Figure 3(c) reports pass@1 on tasks that trigger compaction. CompactionRL achieves the highest accuracy on this subset, indicating that jointly training execution and summarization improves reliable continuation from compacted histories.

Training Dynamics. Figure 4(a) analyzes the change of summary length during GLM-4.5-Air CompactionRL training. Compared with CompactionRL (w/o sum. loss), whose summary length decreases during training, CompactionRL produces increasingly longer and more detailed summaries when summary responses are included in the RL objective. This results in more complete and actionable compacted states that better preserve implementation-relevant context and continuation state. This suggests that optimizing summary responses encourages the model to retain useful detailed information for post-compaction continuation, rather than merely recording high-level task progress.



(a) Average compactions (b) Average tool call counts (c) Compacted-task accuracy

Figure 3: Behavior comparison on SWE-bench Verified in the GLM-4.5-Air setting under 80k×4 compacted evaluation. We compare the base model(GLM-4.5-Air-SFT), standard no-compaction RL, CompactionRL (w/o sum. loss), and CompactionRL. Here, we denote a trace as all compaction segments generated to solve a single task. (a) Average number of compaction operations per trace. (b) Average number of tool calls per trace. (c) Pass@1 accuracy on compaction triggering tasks in each evaluation run.



(a) Summary length (b) Reasoning tokens per turn (c) Entropy

Figure 4: Training dynamics of GLM-4.5-Air CompactionRL. (a) Summary optimization leads to increasingly longer and more detailed summaries. (b) CompactionRL increases reasoning tokens per turn. (c) Policy entropy.

We further quantify summary faithfulness and continuation usefulness in Appendix C, supporting that CompactionRL learns to generate summaries of higher quality. Figure 4(b) further shows that the reasoning tokens of CompactionRL increase throughout training, whereas those of CompactionRL (w/o sum. loss) and standard no-compaction RL decrease. This indicates that learned compaction effectively expands the usable context window, allowing for more reasoning budget. Figure 4(c) shows that CompactionRL also exhibits a slower entropy increase, suggesting more controlled policy optimization.

Additional analysis of hyperparameter sensitivity is provided in Appendix B.

5 Conclusion

We presented CompactionRL, a PPO-based framework that trains long-horizon agents with context compaction by optimizing execution and summary tokens under the same task reward. With token-level loss normalization and cross-segment GAE, CompactionRL improves SWE-bench Verified and Terminal-Bench 2.0 performance under a fixed peak context budget, while further ablations show that summary training produces more detailed compacted states and is important for these gains. Overall, our results demonstrate the effectiveness of incorporating context compaction into RL training for long-horizon agents, rather than treating it solely as an inference-time heuristic.

Limitations. CompactionRL is designed for settings where context compaction is available during both training and inference. Extending its benefits to single-window inference requires further study, as the gains observed under compaction-enabled evaluation do not consistently transfer to this setting. In addition, cross-segment GAE remains an approximation to full-trajectory credit assignment, motivating further investigation into more precise reconstruction.

AI USE STATEMENT

We did not use generative AI tools for any tasks with required disclosure. We used generative AI tools only for language polishing and grammar correction. All research ideas, methods, experiments, and analyses were conceived, designed, implemented, and validated by the authors. We reviewed all AI-assisted text and revised it as necessary. We take responsibility for the final content of this work, including text produced with the aid of generative AI.

REFERENCES

Arash Ahmadian et al. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. *arXiv preprint arXiv:2402.14740*, 2024.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172/.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems*, volume 35, pp. 16344–16359, 2022.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Team GLM, 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.

Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments, 2026. URL https://doi.org/10.5281/zenodo.20953922.

Zhenyu Hou, Ziniu Hu, Yujiang Li, Rui Lu, Jie Tang, and Yuxiao Dong. Treerl: Llm reinforcement learning with on-policy tree search, 2025a. URL https://arxiv.org/abs/2506.11902.

Zhenyu Hou, Xin Lv, Rui Lu, Jiajie Zhang, Yujiang Li, Zijun Yao, Juanzi Li, Jie Tang, and Yuxiao Dong. T1: Advancing language model reasoning through reinforcement learning and inference scaling, 2025b. URL https://arxiv.org/abs/2501.11651.

Zhenyu Hou, Yujiang Li, Jie Tang, and Yuxiao Dong. Single-rollout asynchronous optimization for agentic reinforcement learning, 2026. URL https://arxiv.org/abs/2607.07508.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024. URL https://arxiv.org/abs/2404.06654.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. *arXiv preprint arXiv:2310.06839*, 2023a.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Llmlingua: Compressing prompts for accelerated inference of large language models. *arXiv preprint arXiv:2310.05736*, 2023b.

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

KRAFTON AI and Ludo Robotics. Terminus-kira: Boosting frontier model performance on terminal-bench with minimal harness, 2026. URL https://github.com/krafton-ai/kira.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

Miao Lu, Weiwei Sun, Weihua Du, Zhan Ling, Xuesong Yao, Kang Liu, and Jiecao Chen. Scaling LLM multi-turn RL with end-to-end summarization-based context management. *arXiv preprint arXiv:2510.06727*, 2025. doi: 10.48550/arXiv.2510.06727. URL https://arxiv.org/abs/2510.06727.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces, 2026. URL https://arxiv.org/abs/2601.11868.

Reiichiro Nakano et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

Long Ouyang et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744, 2022.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. URL https://arxiv.org/abs/2310.08560.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023.

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

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. doi: 10.48550/arXiv.1707.06347. URL https://arxiv.org/abs/1707.06347.

Zhihong Shao et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

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

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback, 2022. URL https://arxiv.org/abs/2211.14275.

Haoran Wang, Zhenyu Hou, Yao Wei, Jie Tang, and Yuxiao Dong. Swe-dev: Building software engineering agents with training and inference scaling. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 3742–3761, 2025.

Zhenting Wang, Huancheng Chen, Jiayun Wang, and Wei Wei. Memex(rl): Scaling long-horizon llm agents via indexed experience memory, 2026. URL https://arxiv.org/abs/2603.04257.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. Swe-rl: Advancing llm reasoning via reinforcement learning on open software evolution, 2025. URL https://arxiv.org/abs/2502.18449.

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Dingchu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, Minhao Cheng, Shuai Wang, Hong Cheng, and Jingren Zhou. ReSum: Unlocking long-horizon search intelligence via context summarization. *arXiv preprint arXiv:2509.13313*, 2025. doi: 10.48550/arXiv.2509.13313. URL https://arxiv.org/abs/2509.13313.

Yifan Wu, Lizhu Zhang, Yuhang Zhou, Mingyi Wang, Bo Peng, Serena Li, Xiangjun Fan, and Zhuokai Zhao. Remember when it matters: Proactive memory agent for long-horizon agents, 2026. URL https://arxiv.org/abs/2607.08716.

Yifan Xu, Xiao Liu, Xinghan Liu, Jiaqi Fu, Hanchen Zhang, Bohao Jing, Shudan Zhang, Yuting Wang, Wenyi Zhao, and Yuxiao Dong. Mobilerl: Online agentic reinforcement learning for mobile gui agents, 2025. URL https://arxiv.org/abs/2509.18119.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Lu Yi, Runlin Lei, Liuyi Yao, Yuexiang Xie, Yuyang Li, Wenhao Zhang, Zhewei Wei, Yaliang Li, and Jian-Yun Nie. Learning agent-compatible context management for long-horizon tasks, 2026. URL https://arxiv.org/abs/2605.30785.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Yufeng Yuan, Yu Yue, Ruofei Zhu, Tiantian Fan, and Lin Yan. What's behind PPO's collapse in long-CoT? value optimization holds the secret. *arXiv preprint arXiv:2503.01491*, 2025. doi: 10.48550/arXiv.2503.01491. URL https://arxiv.org/abs/2503.01491.

Yu Yue, Yufeng Yuan, Qiying Yu, Xiaochen Zuo, Ruofei Zhu, Wenyuan Xu, Jiaze Chen, Chengyi Wang, TianTian Fan, Zhengyin Du, et al. Vapo: Efficient and reliable reinforcement learning for advanced reasoning tasks. *arXiv preprint arXiv:2504.05118*, 2025.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, et al. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents, 2025. URL https://arxiv.org/abs/2506.15841.

Zilin Zhu, Chengxing Xie, Xin Lv, and slime Contributors. slime: An llm post-training framework for rl scaling. https://github.com/THUDM/slime, 2025. GitHub repository. Corresponding author: Xin Lv.

A Experimental Details

A.1 SWE-bench Verified Subset

All SWE-bench Verified results in our experiments use the same 200-task subset spanning 10 repositories. For reproducibility, Table 5 lists the complete subset grouped by repository. We report each repository prefix once and list the corresponding numeric instance IDs; for example, `astropy__astropy` with instance ID 7671 denotes `astropy__astropy-7671`.

Table 5: The 200-task SWE-bench Verified evaluation subset, grouped by repository.

Repository	Instance IDs
astropy__astropy	7671, 8707, 13579, 13977, 14365, 14369, 14995
django__django	9296, 10097, 10914, 10999, 11095, 11133, 11138, 11163, 11179, 11211, 11276, 11292, 11299, 11400, 11433, 11477, 11490, 11532, 11728, 11734, 11749, 11815, 11880, 11951, 11964, 11999, 12125, 12155, 12193, 12209, 12262, 12273, 12276, 12419, 12708, 13012, 13112, 13128, 13158, 13297, 13343, 13344, 13346, 13363, 13449, 13516, 13551, 13569, 13658, 13741, 13786, 13809, 13810, 13820, 13821, 13837, 13933, 14007, 14089, 14155, 14170, 14238, 14404, 14493, 14539, 14559, 14580, 14672, 14725, 14765, 14771, 14787, 14792, 14999, 15022, 15037, 15098, 15128, 15252, 15278, 15368, 15382, 15467, 15499, 15554, 15563, 15572, 15731, 15741, 15863, 16032, 16100, 16136, 16256, 16263, 16333, 16429, 16485, 16502, 16595, 16612, 16631, 16661, 16662, 16819, 16901
matplotlib__matplotlib	20676, 22719, 24149, 24627, 24970, 25311, 25775, 26342
psf__requests	1142, 1766, 1921, 2317, 5414
pydata__xarray	2905, 3151, 3305, 4094, 4356, 4629, 6461, 7229
pylint-dev__pylint	4604, 6903, 7277, 8898
pytest-dev__pytest	5631, 5840, 7205, 7236, 7521, 7982, 10051, 10081
scikit-learn__scikit-learn	9288, 10297, 11310, 13124, 13439, 14141, 14496, 15100
sphinx-doc__sphinx	7440, 7462, 7757, 7889, 7985, 8035, 8056, 8120, 8265, 8459, 8475, 8548, 9258, 9591, 9658, 10435, 10466
sympy__sympy	11618, 12096, 12419, 13091, 13372, 13480, 13551, 13757, 13852, 13974, 14248, 14976, 15349, 15875, 15976, 16450, 16597, 17318, 18211, 18698, 19346, 19495, 19637, 22714, 22914, 23262, 23534, 24066, 24539

A.2 Training and Compute Details

Model and compute configurations. The policy and critic are initialized from the same backbone, with the critic augmented by a scalar value head. All hyperparameter settings are identical for both the 30B and 106B configurations. Table 6 summarizes the training configurations and compute resources for both models. Table 7 compares the training-resource usage of CompactionRL and the GRPO baseline under GLM-4.5-Air(106B-A12B). We additionally repeated CompactionRL training with two random seeds under the same configuration and observed similar performance across seeds.

Table 6: Training configurations and compute resources for CompactionRL.

Item	Configuration
GLM-4.7-Flash (30B-A3B)	
Model architecture	47 layers, hidden size 2,048, 20 attention heads
MoE configuration	64 routed experts, top-4 routing, 1 shared expert
Runtime / compute	180 training steps in 15 hours; 960 GPU-hours
GLM-4.5-Air (106B-A12B)	
Model architecture	46 layers, hidden size 4,096, 96 attention heads
MoE configuration	128 routed experts, top-8 routing, 1 shared expert
Runtime / compute	100 training steps in 23 hours; 1,472 GPU-hours
Shared training configuration	
Policy optimization	Adam, constant LR 2×10^{-6}, $\beta = (0.9, 0.98)$, weight decay 0.1
Critic optimization	LR 3×10^{-6}, two critic updates per policy update
Advantage estimation	$\gamma = 1$, length-adaptive GAE with $\lambda = 1 - \frac{1}{\alpha l}$ and $\alpha = 1.5$
KL	KL coefficient 0
Hardware	64 NVIDIA H800 GPUs: 32 shared by actor and critic, and 32 for rollouts

Table 7: Training-resource comparison between CompactionRL and the GRPO baseline.

Method	Total prompts	Batch size (segments)	Total rollouts	Generated tokens	Wall-clock time	GPU-hours
CompactionRL	1,405	128	11,092	442M	23 h	1,472
SUPO	1,405	128	11,484	448M	25 h	1,600

A.3 Compaction Prompt Templates

We use fixed templates for summary generation and post-compaction resumption. These correspond to q_{sum} and $u_{\text{resume}}(S_t)$ in Section 3.1. The original task instruction is explicitly included in the summary-generation prompt, while the most recent k assistant–tool-result pairs are retained verbatim after compaction.

Summarization Prompt q_{sum}

Your job is to create a detailed summary of the conversation so far, paying close attention to the user's explicit requests and the work already completed.

Original task:
{ORIGINAL_INSTRUCTION}

You are summarizing the earlier portion of the conversation only. {RECENT_MESSAGE_NOTE} will remain in the conversation verbatim, so do not repeat them in detail unless they are required for continuity.

Do not assume you can read external files or use tools. Work only from the conversation provided in this context.

Before providing the summary, think through what matters most to preserve. Write the response using exactly these tags:

<analysis> Brief internal reasoning about what to preserve. </analysis>
<summary> Your summary here. </summary>

Inside <summary>, organize the content so the next model can continue without reading any external files. Include:
1. Primary Request and Intent
2. Key Technical Concepts
3. Files and Code Sections examined or changed
4. Errors encountered and how they were addressed
5. Problem Solving and important decisions
6. All user messages and explicit asks from the summarized portion
7. Pending Tasks
8. Current Work
9. Optional Next Step

Be concrete. Preserve exact commands, outputs, code behavior, and error text when those details matter for the work to continue. Do not output JSON.

Post-Compaction Resume Prompt $u_{\text{resume}}(S_t)$

This session is being continued from a previous conversation that ran out of context. The summary below covers the earlier portion of the conversation.

{SUMMARY_TEXT}

Recent messages are preserved verbatim below and should be treated as the source of truth for the latest state.

Continue the conversation from where it left off without asking any further questions. Resume directly from this summary and the preserved recent messages.

B Analysis on Hyperparameter Sensitivity

We evaluate the hyperparameter sensitivity of CompactionRL on the 200-task SWE-bench Verified subset using the trained CompactionRL checkpoint of GLM-4.5-Air (106B-A12B). We vary the context budget, maximum number of compactions, and retained history length at inference time. The context budget denotes the maximum working context length C, and k denotes the number of recent agent–environment interaction turns retained after compaction.

Table 8: Sensitivity to inference-time compaction settings on SWE-bench Verified. Results are Pass@1 accuracy (%) reported as mean $\pm$ standard deviation over evaluation runs.

Factor	Setting	SWE-bench Verified
Context budget	32k	63.0 ± 0.5
	64k	65.5 ± 1.7
	80k	66.4 ± 0.7
Maximum compactions	0	59.1 ± 2.2
	1	65.3 ± 1.5
	3	66.4 ± 0.7
Retained history length	$k = 0$	66.0 ± 0.5
	$k = 2$	66.4 ± 0.7

Table 8 shows that CompactionRL maintains stable performance within practical hyperparameter ranges, indicating limited sensitivity to these settings. Removing the recent-history tail causes only a minor performance drop, suggesting that the learned summaries retain most task-relevant state.

C Summary Quality Analysis

We evaluate summary quality on SWE-bench Verified trajectories where at least one of CompactionRL and RL trained without compaction, when evaluated with inference-time compaction, triggers at least one compaction. The analysis covers 79 samples from the 200-task SWE-bench Verified subset. When multiple compaction summaries are available within the same trajectory, we first average their scores. To reduce uncertainty from the LLM judge, each evaluation is repeated three times. As shown in Table 9, CompactionRL obtains a higher average score than RL without compaction training, with an overall mean difference of $+0.22$ and a 95% bootstrap confidence interval of $[0.020, 0.420]$.

Table 9: Summary-quality evaluation of RL without compaction training and CompactionRL. Scores use a 0–5 scale.

Dimension	Evaluation criterion	RL (w/o compaction training)	CompactionRL
Requirement Fidelity	Whether the summary accurately preserves task requirements, constraints, and acceptance criteria	3.67	**4.00**
Current-State Fidelity	Whether the summary accurately records the current repository, patch, test, and completion state	3.28	**3.53**
Failure-History Fidelity	Whether the summary preserves key failures, diagnoses, and rejected approaches	3.38	**3.48**
Next-Step Validity	Whether the pending work and next steps are correct, specific, executable, and non-misleading	3.14	**3.43**
Continuation Usefulness	Whether the summary reduces redundant exploration caused by missing or misleading state and promotes new, verifiable progress	3.05	**3.15**
Total	–	3.30	**3.52**

The score distribution also shifts away from low-quality summaries: the fraction of scores in the 0–2 range decreases from 22.2% for RL without compaction training to 11.9% for CompactionRL. On the subset where both methods compact, CompactionRL also achieves a higher task success rate, 38.10% versus 28.57%, consistent with its higher summary-quality scores.

Judge setup. The judge uses `claude-opus-5` with `reasoning_effort=medium`. The evaluation prompt is shown below.

Summary-Quality Judge Prompt

Role. You are a trajectory summary evaluator. Assess a compaction summary for **Faithfulness** and **Continuation Usefulness**.

The trace contains:
- **Segment 1:** original prompt, pre-summary execution, summary prompt, and candidate summary.
- **Segment 2:** original prompt, candidate summary, and post-summary execution.

Rules.
- Evaluate whether the summary accurately preserves continuation-critical requirements, state, failures, blockers, and next steps.
- Separate summary quality from continuation-policy, execution, and environment quality.
- Cite concrete trace evidence for every score.
- Judge relevant completeness and information density, not length or detail count.
- The final two pre-summary assistant/tool-result pairs form the retained tail. Do not penalize facts already clear there.
- Use post-summary execution only for continuation usefulness, and penalize only summary-caused recovery errors, repetition, or detours.

Scores. Use 0–5: 5 nearly perfect; 4 high quality; 3 usable with clear defects; 2 important defects; 1 core state missing or wrong; 0 unusable; "N/A" if there is insufficient evidence.

Dimensions.
- **F1 Requirement Fidelity:** Preserves required constraints and acceptance criteria.

- **F2 Current-State Fidelity:** Accurately preserves repository, file, patch, test, environment, and completion state.
- **F3 Failure-History Fidelity:** Preserves failures, diagnoses, rejected hypotheses, and costly ineffective approaches.
- **F4 Next-Step Validity:** Gives correct, specific, executable next steps consistent with the current state.
- **U Continuation Efficiency and Progress:** Helps recover state, avoid summary-induced repetition or detours, and make verifiable progress.

Output. Output exactly one valid JSON object and no other text.

```
{
  "faithfulness": {
    "F1_requirement_fidelity":
      {"score": 0, "evidence": [], "rationale": "", "key_issues": []},
    "F2_current_state_fidelity":
      {"score": 0, "evidence": [], "rationale": "", "key_issues": []},
    "F3_failure_history_fidelity":
      {"score": 0, "evidence": [], "rationale": "", "key_issues": []},
    "F4_next_step_validity":
      {"score": 0, "evidence": [], "rationale": "", "key_issues": []}
  },
  "usefulness": {
    "U_continuation_efficiency_and_progress":
      {"score": 0, "evidence": [], "rationale": "", "key_issues": []}
  }
}
```

Trace.

```
<segment_1>
{{SYSTEM_AND_USER_PROMPTS}}{{PRE_SUMMARY_EXECUTION}}{{SUMMARY_PROMPT}}
<candidate_summary>{{CANDIDATE_SUMMARY}}</candidate_summary>
</segment_1>
<segment_2>
{{INITIAL_PROMPT_AND_CANDIDATE_SUMMARY}}{{POST_SUMMARY_EXECUTION}}
</segment_2>
```

Failure modes. We identify three recurring failure modes in lower-quality summaries: (1) omitted or misdiagnosed failures, where the summary records that tests failed but preserves an incorrect diagnosis; (2) stale or misrepresented repository states, where temporary debugging choices are mistaken for the final solution; and (3) premature completion claims, where passing tests are treated as sufficient even though the implementation still violates an explicit task requirement. In a representative comparison, RL (w/o compaction training) treats an earlier database failure as the current test status despite a later successful test run, causing the post-compaction agent to continue debugging an obsolete issue and ultimately fail. By contrast, CompactionRL preserves a more detailed account of the execution state, including which file-writing attempts failed, how these failures corrupted the source file, and why the task remained incomplete. These details guide the post-compaction agent to recover the file, complete the required modification, and verify the solution.