RECAP: TRAINING LONG-HORIZON LLM AGENTS WITH CONTEXT COMPACTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Context compaction enables large language model agents to tackle long-horizon tasks within limited context budgets by replacing interaction histories with compressed summaries. However, learning effective compaction jointly with the agent policy remains challenging, as summaries affect task success over many subsequent interactions. We introduce RECAP, a reinforcement learning (RL) framework based on the principle that a summary should preserve the behavior induced by the original interaction history. We formalize this principle with a behavior distortion objective that measures the divergence between the agent's subsequent behavior under compressed and original contexts. We show that the gradient of the distortion objective decomposes into two complementary learning signals: distortion-based reward for summary generation and self-distillation for subsequent actions. Experiments on Qwen3.5-4B show that RECAP consistently outperform the task-reward-only RL baselines on both environments, leading to an absolute improvement of 4.6% and 4.3% on BrowseComp-plus and SWE-bench Verified, respectively. Ablations further highlight the value of jointly supervising summary generation and post-compaction behavior for RECAP.

1 INTRODUCTION

Large language model (LLM) agents have demonstrated strong capabilities in solving complex tasks through reasoning, tool use, and multi-step interactions with their environments (Yao et al., 2022; Wei et al., 2022; Nakano et al., 2021; Smith, 2025). Recent advances in post-training, particularly reinforcement learning (RL), have further improved agent performance in various challenging domains such as deep-research (Jin et al., 2025; Li et al., 2025), software engineering (Jimenez et al., 2024; Yang et al., 2025) and scientific discovery (Huang et al., 2025; Roohani et al., 2025). However, as task horizons increase, LLM agents accumulate growing interaction histories that can exceed the context limit before task completion (Beltagy et al., 2020; Liu et al., 2024). Scaling LLM agents to long-horizon tasks therefore requires not only capable task policies, but also effective management of the information retained in the context for subsequent decisions.

Context compaction provides a straightforward approach to overcome context limits by consolidating an overlong interaction history into a compressed summary (Zhou et al., 2026; Yu et al., 2026). This simple yet effective mechanism allows agents to operate within a fixed context budget and has been widely adopted in various agent harnesses (Smith, 2025). In particular, earlier works have explored context compaction through carefully designed summarization prompts (Kang et al., 2025) or separately optimized compressors (Yi et al., 2026). More recent works focused on jointly training the agent policy and compaction through reinforcement learning (Lu et al., 2025; Sun et al., 2025; Li et al., 2026). However, these approaches typically rely on final task rewards to optimize both compaction and task execution. In long trajectories, the effect of a compaction decision might propagate through many subsequent actions or later compactions, making credit assignment difficult. This raises a fundamental question: *Is there a more direct learning signal for context compaction that can guide training long-horizon LLMs beyond task success?*

In this paper, we introduce RECAP (**RE**inforcement learning for **C**ontext compaction and **A**gent **P**olicy optimization), a framework for training long-horizon agents with context compaction. Our key insight is that a useful summary should retain the information needed for subsequent decisions, thereby *preserving the behavior induced by the original interaction history*. We formalize this prin-

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: **Overview of the behavior distortion objective in RECAP.** We measure the effect of context compaction through the KL divergence between the subsequent actions conditioned on original and compacted contexts, which results in a distortion-based reward for summary generation (green) and self-distillation for post-compaction actions (blue).

ciple with a behavior distortion objective that measures the token-level KL divergence between the agent's responses under compressed and original contexts. Here, the same policy conditioned on the original context serves as a self-teacher, specifying the behavior to be preserved after compaction.

We decompose the gradient of the distortion objective and show that it provides two complementary learning signals: *(i)* distortion-based reward for summary generation, and *(ii)* self-distillation for actions after compaction. The distortion-based reward assigns higher credit to summaries that induce less behavior distortion compared to the original context. The context self-distillation directly trains the subsequent actions using the full-context policy as the self teacher, providing dense token-level supervision to the same policy conditioned on the compressed context (Hübotter et al., 2026; Zhao et al., 2026). We combine these two learning signals with the task-success objective, jointly optimizing context compaction and the agent policy within a unified RL framework, which does not require any additional rollouts or environment interactions.

We evaluate RECAP on two representative long-horizon environments, BrowseComp-plus (Chen et al., 2025b) and SWE-bench Verified (Jimenez et al., 2024), covering deep-research and software engineering tasks. Using Qwen3.5-4B with a 32K working context limit, RECAP outperforms the task-reward-only RL baselines by 4.6% absolute score on the BrowseComp-plus and 4.3% on the SWE-bench Verified, respectively. These gains are accompanied by fewer tool calls and lower token usage in the trajectory, demonstrating improved efficiency of the trained agents. Moreover, we find the advantage of RECAP is more evident under a tighter 16K working-context limit, suggesting that behavior-preserving supervision is particularly useful in the scenario where the agent frequently compact its history to solve the task.

2 Related Work

RL for LLM agent. Reinforcement learning (RL) has become a central component of LLM post-training, improving instruction following and reasoning through human feedback and verifiable rewards (Ouyang et al., 2022; Shao et al., 2024; Schulman et al., 2017; Guo et al., 2025; Yu et al., 2025). Recent work extends RL to multi-turn agents (Chen et al., 2025a; Wang et al., 2025b; 2026b; Jiang et al., 2026), enabling models to learn search and software-engineering behaviors through interactions with tools and environments (Nakano et al., 2021; Jin et al., 2025; Zheng et al., 2025; Jain et al., 2025; Wei et al., 2025).As task horizons increase, sparse and delayed rewards make it difficult to assign credit to individual actions. Recent work addresses this challenge through step-level advantage estimation and turn-level reward assignment, providing finer-grained supervision for agent training (Feng et al., 2025; Wang et al., 2025a; Tan et al., 2026; Dong et al., 2026; Tao et al., 2026). Complementing these efforts, RECAP augments task rewards with a behavior distortion objective that provides localized learning signals for both summary generation and post-compaction actions.

On-policy self-distillation. On-policy distillation trains language models using teacher predictions on student-generated trajectories, providing a dense token-level supervision (Agarwal et al., 2024;

Lu & Lab, 2025; Ma et al., 2026; Ye et al., 2026b). Recent self-distillation methods construct a teacher from the same model by conditioning it on privileged information, such as verified solutions or environment feedback (Hübotter et al., 2026; Zhao et al., 2026; Kim et al., 2026a; Yang et al., 2026; Shenfeld et al., 2026). This approach has also been extended to multi-turn agents, where skill-conditioned self-distillation complements task-reward optimization (Lu et al., 2026; Wang et al., 2026a). Our work uses the original interaction history as privileged context to supervise the agent's post-compaction behavior. We further use the resulting behavioral discrepancy to assign credit to summaries, jointly optimizing context compaction and the agent policy.

Context management for long-horizon task. Existing approaches manage growing interaction histories through context compaction (Wu et al., 2025; Kang et al., 2025; Ye et al., 2026a; Yuan et al., 2025), selective memory retention and retrieval (Packer et al., 2023; Xie et al., 2026; Zhang et al., 2026), or context decomposition across sub-agents and recursive calls (Zhang et al., 2024; 2025). Recent work uses end-to-end RL to jointly optimize context management and task execution (Lu et al., 2025; Sun et al., 2025; Li et al., 2026). For context compaction, however, task outcomes provide only indirect supervision for summaries whose effects unfold over subsequent interactions. RECAP complements task-success rewards with a behavior distortion objective that supervises both summary generation and post-compaction actions.

3 BACKGROUND

In this section, we formalize the agent rollouts with context compaction and provide a preliminary for key concepts we build upon.

Standard agent loop. Let I denote the system prompt and task instruction. A standard LLM-agent trajectory can be represented by:

$$\tau = (I, a_1, o_1, a_2, o_2, \ldots, a_T, o_T), \tag{1}$$

where a_t is the model response and o_t is the environment observation at step $t \in \{1 \ldots T\}$. The context available for LLM agent before action a_t is

$$h_t = (I, a_1, o_1, a_2, o_2, \ldots, a_{t-1}, o_{t-1}). \tag{2}$$

At each step, the agent generates a response from its policy $a_t \sim \pi_\theta(\cdot \mid h_t)$, after which the environment produces an observation through the transition function $o_t \sim P(\cdot \mid h_t, a_t)$. The new action-observation pair (a_t, o_t) is directly appended to the context for the next turn. Together, this process describes the standard agent loop under the case of sufficient context window.

Context compaction. For overlong trajectory, we compact the accumulated context whenever its length exceeds a fixed budget. Specifically, when $|h_t| \geq L_{\text{trigger}}$ at step t with L_{trigger} being the fixed budget, the agent is prompted to generate a compressed summary m_t of the interaction history

$$m_t \sim \pi_\theta(\cdot \mid h_t \oplus q_{\text{comp}}), \tag{3}$$

where q_{comp} is the fixed compression instruction and $\oplus$ denotes the prompt concatenation. The compaction instructs the agent to create a handoff summary on the current progress, key observation and potential next step. The content of the q_{comp} is provided in Appendix D.

After that, the compressed context is constructed as:

$$\tilde{h}_t = (I, m_t, \underbrace{a_{t-k}, o_{t-k}, \ldots, a_{t-1}, o_{t-1}}_{\text{Last } k \text{ interactions}}) \tag{4}$$

where the task instruction and the last k steps are preserved, and the intermediate interactions are replaced by the generated summary m_t. The agent continues to work on task with the compressed context until the task end or the next compaction.

A long-horizon task might contain multiple compactions at steps $t_1 < \cdots < t_K$, which naturally split the entire trajectory into multiple segments:

$$\tilde{\tau} = (I, \sigma_0, m_{t_1}, \sigma_1, \ldots, m_{t_K}, \sigma_K), \tag{5}$$

where each segment σ_k is the collection of action and observation pairs in sequence. At a later compaction, the agent again summarizes its current working context, including the previous summary and subsequent interactions, and applies the same rule to reconstruct the new context.

RL training with compaction. Training an agent with learned context compaction involves optimizing both the task policy and the generation of compression summaries. Recent works typically adopt end-to-end reinforcement learning (RL) to jointly optimize task actions and compression summaries (Lu et al., 2025; Li et al., 2026), using task success as the training objective:

$$J_{\text{task}}(\theta) = \mathbb{E}_{\tilde{\tau}\sim\pi_\theta}\left[R(\tilde{\tau})\right], \tag{6}$$

where $R(\tilde{\tau}) \in \{0, 1\}$ denotes the binary reward of task-success and is assigned to both regular action steps and compaction steps. The policy can be optimized with standard policy-gradient methods such as PPO (Schulman et al., 2017) or GRPO (Shao et al., 2024).

However, the final task reward only evaluates trajectory as a whole, providing only indirect supervision for context compaction via task success. The difficulty is that in a long-horizon trajectory the effect of an individual compaction depends on many subsequent actions and potentially further compactions. As a result, the trajectory-level reward provides a noisy credit assignment for compaction decisions and obscures the effect of any single compaction. This limitation was also reported in recent works (Sun et al., 2025; Xie et al., 2026; Yi et al., 2026). Motivated by this, we propose to use a complementary learning signal that evaluates the effect of each compaction directly.

4 RECAP

We introduce RECAP, an RL framework for jointly optimizing long-horizon agent policies and context compaction. Rather than supervising compaction solely through the final task-success signal, we provide a more direct learning signal based on its effect on the agent's subsequent behavior. Our central idea is that a good compaction should preserve the behavior induced by the original interaction history. Based on this principle, we introduce a *behavior distortion objective* that in addition to the task success provides learning signal for agent training. We first formulate the objective, then derive the corresponding updates for summaries and continuations, and finally integrate them into joint policy optimization.

For clarity. Throughout this section, we present the distortion objective only for a single compaction event in the trajectory. Our formulation naturally extends to the scenarios with multiple compactions, which we defer to Appendix B.2.

4.1 Behavior Distortion Objective

In addition to the final task success, we introduce a behavior distortion objective for jointly training the agent policy and context compaction. The key intuition is that *the context compaction should preserve the agent's behavior in subsequent interactions*. In other words, after a compaction at step t, the behavior induced by the compressed context $\tilde{h}_t$ should remain close to that induced by the original context h_t. At the policy level, this discrepancy can be measured by the KL divergence:

$$D_{\text{KL}}\left(\pi_\theta(\cdot \mid \tilde{h}_t) \,\middle\|\, \pi_\theta(\cdot \mid h_t)\right). \tag{7}$$

We measure the distortion through the subsequent interaction segment in the trajectory. Specifically, let $\sigma_t^+ = (a_t, o_t, \ldots, a_T, o_T)$ denote the action–observation segment after the compaction at step t, ending at the next compaction or at trajectory termination. For a generated summary m_t, we define its behavior distortion as:

$$D_t(m_t) = \mathbb{E}_{\sigma_t^+\sim\pi_\theta(\cdot|\tilde{h}_t)}\left[\sum_{i\in\sigma_t^+}\sum_{j=1}^{|a_i|} D_{\text{KL}}\left(\pi_\theta(\cdot \mid \tilde{h}_t, s_{i,j}) \,\middle\|\, \text{sg}\left[\pi_\theta(\cdot \mid h_t, s_{i,j})\right]\right)\right], \tag{8}$$

where $s_{i,j}$ is the shared action–observation prefix preceding j-th token of action a_i within σ_t^+, and $\text{sg}(\cdot)$ denotes the stop-gradient operator. Thus, $D_t(m_t)$ measures the accumulated token-level KL divergence over the action tokens in the segment σ_t^+. The two policies share the same prefix $s_{i,j}$ and differ only in whether they're conditioned on $\tilde{h}_t$ or h_t. The full-history policy therefore serves as a self-teacher specifying the behavior that should be preserved after compaction.

We define the compaction objective by minimizing this behavioral distortion:

$$J_{\text{comp}}(\theta) = \mathbb{E}_{m_t\sim\pi_\theta(\cdot|h_t\oplus q_{\text{comp}})}\left[-D_t(m_t)\right]. \tag{9}$$

Maximizing $J_{\text{comp}}(\theta)$ therefore encourages the compressed-context behavior to remain close to the corresponding full-context behavior.

Together with the final task success in Eq.(6), the RECAP's final objective is:

$$J_{\text{total}}(\theta) = J_{\text{task}}(\theta) + \lambda J_{\text{comp}}(\theta), \tag{10}$$

where λ controls the relative strength of behavior preservation and task success. Here $J_{\text{comp}}(\theta)$ serves as an auxiliary regularizer for behavior distortion under context compaction, while $J_{\text{task}}(\theta)$ encourages behaviors that are useful for solving the task.

4.2 Learning Summaries and Continuations

Optimizing the behavior distortion objective in Eq.(9) affects the policy through two pathways:

(I) The compression summary $m_t \sim \pi_\theta(\cdot \mid h_t \oplus q_{\text{comp}})$.

(II) The post-compression action-observation segments $\sigma_t^+ \sim \pi_\theta(\cdot \mid \tilde{h}_t)$.

These two pathways are reflected in the gradient of the compaction objective J_{comp}. Conditional on the fixed pre-compaction history h_t, the gradient of J_{comp} in Eq.(9) can be decomposed as

$$\nabla_\theta J_{\text{comp}}(\theta) = \underbrace{\mathbb{E}_{m_t}\left[-D_t(m_t)\nabla_\theta \log \pi_\theta(m_t \mid h_t \oplus q_{\text{comp}})\right]}_{g_{\text{sum}}} + \underbrace{\mathbb{E}_{m_t}\left[-\nabla_\theta D_t(m_t)\right]}_{g_{\text{post}}}, \tag{11}$$

where g_{sum} corresponds to the gradient assigned to the generated summary and g_{post} corresponds to the gradient of the post-compression segments.

Learning the summary from distortion-based reward. The first term g_{sum} corresponds to a standard policy-gradient update for summary generation, where the negative behavior distortion serves as the reward signal. For each generated summary m_t, we sample a single subsequent segment σ_t^+ to estimate the distortion, denoted by $\widehat{D}_t$. The scalar reward $-\widehat{D}_t$ is then assigned to all tokens in m_t, giving higher credit to summaries that induce less distortion in the agent's subsequent behavior.

Learning the continuation through self-distillation. The second term g_{post} provides supervision to the agent's actions after compaction. We treat the policy conditioned on the full context $\pi_\theta(\cdot \mid h_t)$ as the teacher and the policy conditioned on the compressed context $\pi_\theta(\cdot \mid \tilde{h}_t)$ as the student. Specifically, the student generates the continuation σ_t^+, and then the teacher evaluates each token under the same post-compaction prefix $s_{i,j}$, differing only in the context choice of h_t or $\tilde{h}_t$. We therefore optimize this term using an on-policy self-distillation estimator, which provides dense token-level supervision throughout the continuation:

$$g_{\text{post}}^{\text{SD}} = \mathbb{E}_{m_t, \sigma_t^+}\left[\sum_{i \in \sigma_t^+} \sum_{j=1}^{|a_i|} \log \frac{\pi_\theta(a_{i,j} \mid h_t, s_{i,j})}{\pi_\theta(a_{i,j} \mid \tilde{h}_t, s_{i,j})} \cdot \nabla_\theta \log \pi_\theta(a_{i,j} \mid \tilde{h}_t, s_{i,j})\right], \tag{12}$$

where the expectation follows the summary and continuation sampling described above. We provide the derivation in Appendix A. Therefore, for each token in the continuation, the self-distillation objective provides token-level advantage as $A_{i,j}^{\text{SD}} = \log \frac{\pi_\theta(a_{i,j} \mid h_t, s_{i,j})}{\pi_\theta(a_{i,j} \mid \tilde{h}_t, s_{i,j})}$. This token-level advantage can then be combined directly with the summary-level distortion-based reward and the task advantage in the joint policy update.

Both terms can be expressed as policy-gradient updates with token-level weights. We summarize the learning signal induced by the compaction objective as the compaction advantage:

$$A^{\text{comp}} = \begin{cases} -\widehat{D}_t - b_t, & \text{for each summary token in } m_t, \\ A_{i,j}^{\text{SD}}, & \text{for each action token } a_{i,j} \text{ in } \sigma_t^+, \end{cases} \tag{13}$$

where b_t is an optional baseline for the negative-distortion reward. Action tokens outside the post-compaction continuation receive no compaction advantage. Thus, the summary receives a shared distortion-based signal, while the continuation receives token-level self-distillation signals.

4.3 JOINT OPTIMIZATION WITH TASK REWARD

The compaction advantage can be directly combined with the task advantage used by the underlying RL algorithm. Let A^{task} denote the task advantage. We obtain the joint advantage as:

$$A^{\text{total}} = A^{\text{task}} + \lambda A^{\text{comp}}, \tag{14}$$

where the corresponding token-level values are used for each summary or action token. In this way, the task reward supervises the entire rollout, while the behavior distortion objective provides additional supervision to the summary and the actions that follow it.

We optimize A^{total} using the same policy-gradient objective as the task-only RL baseline of GRPO (Shao et al., 2024; Lu et al., 2025). The detailed training process are provided in Appendix B.1. Importantly, RECAP requires *no additional rollouts or environment interactions*, as both the distortion-based reward and self-distillation signal are computed from the same sampled trajectory. The overall training time of RECAP remains roughly on par with the RL baselines.

5 EXPERIMENT

5.1 EXPERIMENTAL SETTING

Datasets. We evaluate our method on two representative long-horizon agent tasks: deep search and software engineering. For deep search, we use BrowseComp-Plus (Chen et al., 2025b), with 300 questions for training and 150 held-out questions for evaluation. The provided tools are `search(query)`, `get_document(docid)` and `finish(answer)`. For software engineering, we select 300 tasks from SWE-smith (Yang et al., 2025) for training and evaluate on all 500 tasks in SWE-bench Verified (Jimenez et al., 2024). The available tools are `bash(command)`, `str_replace_editor(command, path)` and `finish()`. In both environments, agents interact with the environment through the provided tools until task termination, with context compaction triggered when the context reaches the specified budget. All methods use the same training data and configurations within each environment. Details of data selection and environment prompts are provided in Appendix C and D.

Training Details. We conduct our main experiments with Qwen3.5-4B (Team, 2026), with thinking mode disabled. For both environments, we use a 32K working-context limit and allow up to three compaction events per trajectory. After each compaction, the compressed context retains the most recent interaction ($k = 1$). We allow up to 100 interaction steps for BrowseComp-Plus and 200 for SWE-agent, and use the same compaction prompt across both environments. We additionally evaluate a 128K context setting without compaction as a reference. For RECAP, we set λ based on a held out validation set with 100 examples for BrowseComp-Plus and SWE-agent, respectively. RL training is implemented in a fully asynchronous setting using the VeRL framework (Sheng et al., 2024). Across all the training runs, we use a rollout batch size of 16, a group size of 8, a learning rate of 1×10^{-6}, and train for 150 steps. During evaluation, we sample responses with temperature of 1.0, top-p of 0.95, and top-k of 20. We report accuracy averaged over four evaluation runs. Full training and evaluation configurations are provided in Appendix C.

Baselines. We compare RECAP to task-reward-only GRPO (Shao et al., 2024) with the same 32K working-context limit and compaction configuration as RECAP. Following the end-to-end formulation of Lu et al. (2025), this method jointly optimizes the agent policy and compaction summaries using the final task-success rewards. For reference, we also report results for GRPO trained and evaluated with a 128K context window without compaction. We additionally report results from other open models using a 128K context window.

5.2 RECAP PERFORMANCE RESULTS

The results on BrowseComp-plus and SWE-bench Verified are presented in Table 1. Under a 32K working-context limit, RECAP outperforms the GRPO baselines with compaction by 4.6% and 4.3% percentage score in accuracy on the BrowseComp-Plus and SWE-bench Verified, respectively. Moreover, RECAP substantially closes the gap to RL training with a 128K context window. For example, on BrowseComp-Plus, RECAP reaches 62.8% accuracy, only with 1.0% below the 128K GRPO result of 63.8%, while using only one quarter of the peak context length. On the SWE-bench

Table 1: **Main results with and without context compaction.** Peak length denotes the maximum working-context length. Accuracy is averaged over 4 runs. For Qwen3.5-4B, we report mean accuracy ± standard deviation across the evaluation runs.

Method	Peak Length	Compaction	BrowseComp-Plus Acc. (%)	SWE-bench Verified Acc. (%)
Open-model references				
Gemma-4-E4B	128K	✗	24.7	15.4
Qwen3-Coder-30B-A3B	128K	✗	24.7	43.7
Seed-OSS-36B	128K	✗	29.3	50.0
GLM-4.5-Air	128K	✗	44.7	52.8
Qwen3.5-9B	128K	✗	53.0	56.8
Without context compaction				
Qwen3.5-4B	128K	✗	49.2 ± 0.8	49.8 ± 2.1
+ GRPO	128K	✗	63.8 ± 3.0	55.5 ± 1.2
With context compaction				
Qwen3.5-4B	32K	✓	40.7 ± 3.1	45.8 ± 1.3
+ GRPO	32K	✓	58.2 ± 1.5	46.8 ± 1.4
+ RECAP (ours)	32K	✓	**62.8 ± 3.9**	**51.1 ± 1.0**

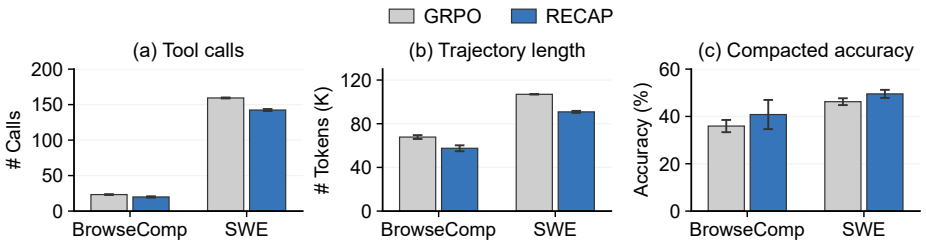

Figure 2: **Interaction steps and accuracy across environments.** Comparison of (a) tool calls per trajectory, (b) cumulative tokens per trajectory, and (c) accuracy on trajectories with at least one compaction. RECAP uses fewer tool calls and tokens on average while achieving higher accuracy on compacted trajectories.

Verified, RECAP achieves 51.1% under context compaction, approaching the 55.5% performance of the 128K setting. These consistent improvements across both environments support the benefit of augmenting task-success optimization with behavior-distortion supervision when jointly training the agent policy and context compaction.

Beyond final task accuracy, we further compare the interaction steps and tokens per trajectory in the trained model in Figure 2 (a) and (b). Across both environments, RECAP uses substantially fewer tool calls and cumulative tokens per task compared to the RL baseline, demonstrating more efficient task execution. We hypothesize that this may result from more effective behavior after context compaction, reducing redundant or repeated interactions. Figure 2(c) further reports accuracy on trajectories that trigger at least one compaction. RECAP consistently achieves higher accuracy on these trajectories, with a particularly pronounced improvement on the BrowseComp-Plus. Together, these results suggest that the benefits of RECAP extend beyond overall task accuracy, improving both interaction efficiency and robustness following context compaction.

5.3 Ablation of Learning Signals

We ablate the two learning signals introduced by the behavior distortion objective in Table 2. We first assess the individual contribution of g_{sum} and g_{post} by augmenting the GRPO baseline with each signal separately. Adding g_{sum} alone provides limited gains,with accuracy slightly decreasing from 58.2% to 56.7% on the BrowseComp-Plus while increasing from 46.8% to 47.9% on the SWE-

bench Verified. In addition, g_{sum} also leads to more tool calls in both environments and produces substantially longer summaries on BrowseComp-Plus. In contrast, adding g_{post} alone consistently improves task performance, reaching 60.3% on BrowseComp-Plus and 47.1% on SWE-bench Verified. The trained model also uses fewer tool calls and shorter summaries in both environments. We hypothesize that this is due to the self-distillation formulation of g_{post}, which uses the full-context policy as a self-teacher to supervise post-compaction actions. Similar efficiency gains from self-distillation have also been reported in recent works (Hübotter et al., 2026; Kim et al., 2026a).

Importantly, the full RECAP that combines the two learning signals yields the best performance in both environments, reaching 62.8% on BrowseComp-Plus and 51.1% on SWE-bench Verified. This indicates that the two signals are complementary: g_{post} regularizes post-compaction behavior, while g_{sum} assigns credit to the summary that induces it. Together, they provide more effective supervision for jointly learning context compaction and the agent policy.

Table 2: **Ablation of the two learning signals in RECAP**. We evaluate the individual and combined effects of g_{sum} and g_{post} in the training. We report accuracy, the average number of tool calls per trajectory, and the average summary length per compaction.

Learning Signals		**BrowseComp-Plus**			**SWE-bench Verified**		
g_{sum}	g_{post}	Acc. (%)	# Tools	Sum. Length	Acc. (%)	# Tools	Sum. Length
✗	✗	58.2	23.4	1146	46.8	158.9	966
✓	✗	56.7	23.5	1788	47.9	180.3	917
✗	✓	60.3	21.2	468	47.1	142.3	557
✓	✓	**62.8**	19.9	709	**51.1**	145.6	1003

5.4 Generalization Across Context Windows

We further evaluate how the trained policies generalize across different context windows. We fix the total interaction budget to 128K tokens and vary the working-context window from 16K to 128K, resulting in increasingly frequent compaction as the context window becomes shorter. We compare RECAP trained at 32K context window with GRPO trained at 32K and 128K context window. The results are shown in Figure 3. Across both environments, RECAP trained at 32K obtains strong performance across different context windows, with particularly clear advantages to the GRPO baselines under shorter contexts that require more frequent compaction to solve a task.

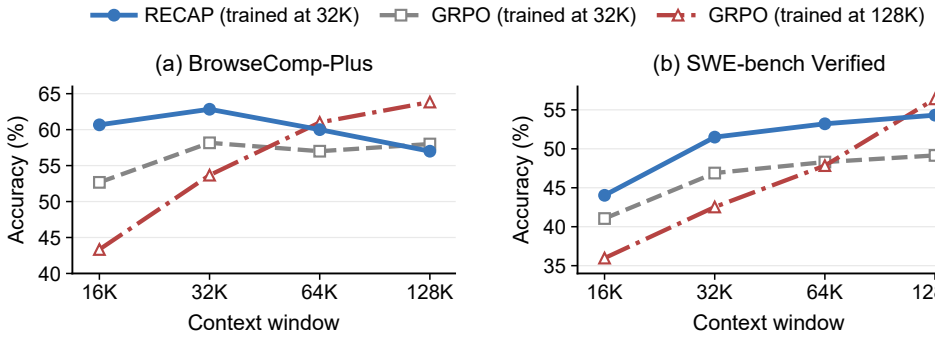

Figure 3: **Performance across context windows**. Evaluation of trained agents on BrowseComp-Plus (left) and SWE-bench Verified (right) with an 128K interaction budget, using 16K $\times$ 8, 32K $\times$ 4, or 64K $\times$ 2 context segments, or 128K without compaction.

On the BrowseComp-Plus, RECAP maintains 60–63% accuracy from 16K to 64K, substantially outperforming both GRPO variants at the shortest 16K window. On the SWE-bench Verified, RECAP outperforms GRPO trained at 32K across all context windows and also surpasses GRPO trained at 128K in all settings that require compaction. In comparison, GRPO trained at 128K performs well in the full context setting but degrades considerably with shorter context windows that require compaction to solve the task.

5.5 Ablation on Relative Weight

Finally, we study the effect of the regularizer λ in the behavior distortion objective in RECAP (Figure 4). On the BrowseComp-Plus, performance improves as λ increases from 0.03 to 0.1 and remains stable around 62% for larger values. With all tested values of the regularizer λ, RECAP substantially outperforms GRPO baseline. On the SWE-bench Verified, performance is more sensitive to the relative weight. The effect of λ is less monotonic, with $\lambda = 0.03$ achieving the highest accuracy of 51.5%. We hypothesize that the difference is due to the higher complexity and longer interaction horizon on the tasks of SWE-bench Verified. As shown in Figure 2 and Table 2, the trajectories in SWE-bench Verified involve substantially more interactions than BrowseComp-Plus, which could make it more challenging for the agent to preserve task-relevant information through compaction. In all experiments we set λ based on the independent validation set which is detailed in Table 3 in Appendix C.

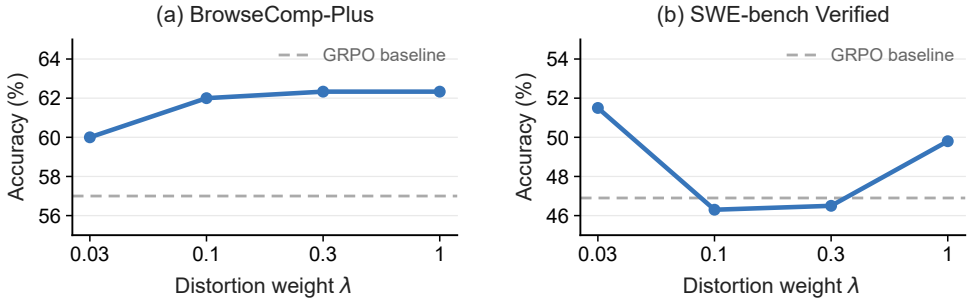

Figure 4: **Ablation of the distortion objective weight λ.** Results with different weight of the behavior distortion objective on BrowseComp-Plus (left) and SWE-bench Verified (right).

6 Conclusion

In this paper, we introduced RECAP, an RL framework for jointly learning context compaction and long-horizon agent policies. Its behavior distortion objective augments task-success optimization with two complementary learning signals: a distortion-based reward for summary generation and self-distillation for post-compaction actions. Experiments on two representative long-horizon benchmarks, BrowseComp-Plus and SWE-bench Verified, demonstrate improved task performance and interaction efficiency over task-reward-only RL. Our experiments focus on Qwen3.5-4B across two task domains, leaving broader evaluation across model scales, task domains, and longer interaction horizons to future work. Real-world tasks might involve diverse information densities and representations, posing additional challenges for effective context compaction over long interaction horizons. This also motivates exploring the benefits of RECAP alongside other non-trivial context management strategies, such as active forgetting and retrieval or task decomposition across recursive sub-agents. It'd also be interesting to integrate RECAP with critic-based RL methods such as PPO to improve credit assignment and sample efficiency in long-horizon training. Overall, our findings suggest that the effects of compaction on subsequent behavior provides a good regularization signal that complements the task reward for training long-horizon agents under limited context budgets.

AI use statement

In this paper, we use generative AI to provide feedback to the research method and experiment design. We additionally use generative AI to polish the manuscript, suggest paper structure, review the related literature and provide feedback on manuscript organization and presentation. Generative AI is also used to generate basic visual elements and icons in the plots, and the final figure were manually designed and assembled by the authors. All AI-assisted suggestions are carefully reviewed and evaluated. Related works are checked and manually included for citations.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, 2024.

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce-style optimization for learning from human feedback in llms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12248–12267, 2024.

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

Kevin Chen, Marco Cusumano-Towner, Brody Huval, Aleksei Petrenko, Jackson Hamburger, Vladlen Koltun, and Philipp Krähenbühl. Reinforcement learning for long-horizon interactive llm agents. *arXiv preprint arXiv:2502.01600*, 2025a.

Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, et al. BrowseComp-Plus: A more fair and transparent evaluation benchmark of deep-research agent. *arXiv preprint arXiv:2508.06600*, 2025b.

Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, et al. Agentic reinforced policy optimization. In *International Conference on Learning Representations*, volume 2026, pp. 16981–17017, 2026.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. In *Advances in Neural Information Processing Systems*, 2025.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-R1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Zhiyuan He, Siwei Zhang, Zhiwen Zhou, Yuqing Yang, Yu Kang, Yuge Zhang, Luna K. Qiu, Tin Yan Tsui, Jiahang Xu, and Chong Luo. Agent lightning v1.0: Towards harnessed agentic rl, 2026. URL `https://arxiv.org/abs/2608.17528`.

Kexin Huang, Serena Zhang, Hanchen Wang, Yuanhao Qu, Yingzhou Lu, Yusuf Roohani, Ryan Li, Lin Qiu, Gavin Li, Junze Zhang, et al. Biomni: A general-purpose biomedical ai agent. *biorxiv*, 2025.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, et al. Reinforcement learning via self-distillation. *arXiv preprint arXiv:2601.20802*, 2026.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. R2e-gym: Procedural environments and hybrid verifiers for scaling open-weights swe agents. *arXiv preprint arXiv:2504.07164*, 2025.

Yulun Jiang, Liangze Jiang, Damien Teney, Michael Moor, and Maria Brbic. Meta-rl induces exploration in language agents. In *International Conference on Learning Representations*, 2026.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, 2024.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

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

Jeonghye Kim, Xufang Luo, Minbeom Kim, Sangmook Lee, Dohyung Kim, Jiwon Jeon, Dongsheng Li, and Yuqing Yang. Why does self-distillation (sometimes) degrade the reasoning capability of llms? *arXiv preprint arXiv:2603.24472*, 2026a.

Minseon Kim, Zhengyan Shi, Emiliano Penaloza, Christopher Cui, Roger Creus Castanyer, Maryam Hashemzadeh, Isadora White, Jonathan Light, Jeonghye Kim, Matheus Pereira, et al. Frognano: Training a 4b coding agent via online task synthesis. *arXiv preprint arXiv:2609.07925*, 2026b.

Xiaoxi Li, Jiajie Jin, Guanting Dong, Hongjin Qian, Yongkang Wu, Ji-Rong Wen, Yutao Zhu, and Zhicheng Dou. Webthinker: Empowering large reasoning models with deep research capability. In *Advances in Neural Information Processing Systems*, 2025.

Yujiang Li, Zhenyu Hou, Yi Jing, Jie Tang, and Yuxiao Dong. CompactionRL: Reinforcement learning with context compaction for long-horizon agents. *arXiv preprint arXiv:2607.05378*, 2026.

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*, 2024.

Kevin Lu and Thinking Machines Lab. On-policy distillation. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20251026. https://thinkingmachines.ai/blog/on-policy-distillation.

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.

Zhengxi Lu, Zhiyuan Yao, Zhuowen Han, Zi-Han Wang, Jinyang Wu, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, et al. Self-distilled agentic reinforcement learning. *arXiv preprint arXiv:2605.15155*, 2026.

Wenhan Ma, Jianyu Wei, Liang Zhao, Hailin Zhang, Bangjun Xiao, Lei Li, Qibin Yang, Bofei Gao, Yudong Wang, Rang Li, et al. Mopd: Multi-teacher on-policy distillation for capability integration in llm post-training. *arXiv preprint arXiv:2606.30406*, 2026.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *Advances in neural information processing systems*, 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.

Yusuf Roohani, Andrew Lee, Qian Huang, Jian Vora, Zachary Steinhart, Kexin Huang, Alexander Marson, Percy Liang, and Jure Leskovec. Biodiscoveryagent: An ai agent for designing genetic perturbation experiments. In *International Conference on Learning Representations*, 2025.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning. *arXiv preprint arXiv:2601.19897*, 2026.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

Calvin Smith. Openhands context condensation for more efficient ai agents, 2025. Blog post.

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.

Hui-Ze Tan, Xiao-Wen Yang, Hao Chen, Jie-Jing Shao, Yi Wen, Yuteng Shen, Weihong Luo, Xiku Du, Lan-Zhe Guo, and Yu-Feng Li. Hindsight credit assignment for long-horizon llm agents. *arXiv preprint arXiv:2603.08754*, 2026.

Leitian Tao, Baolin Peng, Wenlin Yao, Tao Ge, Hao Cheng, Mike Hang Wang, Jianfeng Gao, and Sharon Li. Trace: Turn-level reward assignment via credit estimation for long-horizon agents. *arXiv preprint arXiv:2607.13988*, 2026.

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

Hanlin Wang, Chak Tou Leong, Jiashuo Wang, Jian Wang, and Wenjie Li. Spa-rl: Reinforcing llm agents via stepwise progress attribution. *arXiv preprint arXiv:2505.20732*, 2025a.

Hao Wang, Guozhi Wang, Han Xiao, Yufeng Zhou, Yue Pan, Jichao Wang, Ke Xu, Yafei Wen, Xiaohu Ruan, Xiaoxin Chen, et al. Skill-sd: Skill-conditioned self-distillation for multi-turn llm agents. *arXiv preprint arXiv:2604.10674*, 2026a.

Jiongxiao Wang, Qiaojing Yan, Yawei Wang, Yijun Tian, Soumya Smruti Mishra, Zhichao Xu, Megha Gandhi, Panpan Xu, and Lin Lee Cheong. Reinforcement learning for self-improving agent with skill library. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1529–1550, 2026b.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, et al. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in neural information processing systems*, 2022.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida Wang. SWE-RL: Advancing llm reasoning via reinforcement learning on open software evolution. In *Advances in Neural Information Processing Systems*, 2025.

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Dingchu Zhang, Yong Jiang, et al. Resum: Unlocking long-horizon search intelligence via context summarization. *arXiv preprint arXiv:2509.13313*, 2025.

Zijun Xie, Binbin Zheng, Enlei Gong, Jihua Liu, Yuyang You, Lingfeng Liu, Jiayao Tang, Guanqun Zhao, Aoqi Hu, and Zeyu Chen. ECHO: Prune to act, trace to learn with selective turn memory in agentic rl. *arXiv preprint arXiv:2606.31650*, 2026.

Chenxu Yang, Chuanyu Qin, Qingyi Si, Minghui Chen, Naibin Gu, Dingyu Yao, Zheng Lin, Weiping Wang, Jiaqi Wang, and Nan Duan. Self-distilled rlvr. *arXiv preprint arXiv:2604.03128*, 2026.

John Yang, Kilian Lieret, Carlos Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. SWE-smith: Scaling data for software engineering agents. In *Advances in Neural Information Processing Systems*, 2025.

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

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. In *International Conference on Learning Representations*, 2026a.

Tianzhu Ye, Li Dong, Xun Wu, Shaohan Huang, and Furu Wei. On-policy context distillation for language models. *arXiv preprint arXiv:2602.12275*, 2026b.

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. *arXiv preprint arXiv:2605.30785*, 2026.

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, et al. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent. In *International Conference on Learning Representations*, 2026.

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. In *Advances in Neural Information Processing Systems*, 2025.

Qianhao Yuan, Jie Lou, Zichao Li, Jiawei Chen, Yaojie Lu, Hongyu Lin, Le Sun, Debing Zhang, and Xianpei Han. MemSearcher: Training llms to reason, search and manage memory via end-to-end reinforcement learning. *arXiv preprint arXiv:2511.02805*, 2025. URL https://arxiv.org/abs/2511.02805.

Alex L Zhang, Tim Kraska, and Omar Khattab. Recursive language models. *arXiv preprint arXiv:2512.24601*, 2025.

Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Ö Arık. Chain of agents: Large language models collaborating on long-context tasks. In *Advances in Neural Information Processing Systems*, 2024.

Yuxiang Zhang, Jiangming Shu, Ye Ma, Xueyuan Lin, Shangxi Wu, and Jitao Sang. Memory as action: Autonomous context curation for long-horizon agentic tasks. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*, 2026.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. Deepresearcher: Scaling deep research via reinforcement learning in real-world environments. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 414–431, 2025.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Bryan Kian Hsiang Low, and Paul Liang. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents. In *International Conference on Learning Representations*, 2026.

APPENDIX

A DERIVATION OF RECAP OBJECTIVE

We consider a single compaction event at a fixed history h_t. The policy samples a summary on the current history $m_t \sim \pi_\theta(\cdot \mid h_t \oplus q_{\text{comp}})$, constructs the compressed context $\tilde{h}_t$, and then continue to work on the compressed context with continuation $\sigma_t^+ \sim \pi_\theta(\cdot \mid \tilde{h}_t)$. The continuation ends at the next compaction or task termination. For each action token, let

$$d_{i,j}(\theta) = D_{\text{KL}}\Big(\pi_\theta(\cdot \mid \tilde{h}_t, s_{i,j}) \;\Big\|\; \text{sg}[\pi_\theta(\cdot \mid h_t, s_{i,j})]\Big), \tag{15}$$

where $s_{i,j}$ is the shared continuation prefix and $\text{sg}(\cdot)$ denotes the stop-gradient operator. Writing $\sum_{i,j}$ for the sum over action tokens in σ_t^+, the behavior distortion objective is

$$J_{\text{comp}}(\theta) = \mathbb{E}_{m_t}[-D_t(m_t)], \qquad D_t(m_t) = \mathbb{E}_{\sigma_t^+ \mid m_t}\left[\sum_{i,j} d_{i,j}(\theta)\right]. \tag{16}$$

All expectations below follow these sampling distributions, with h_t held fixed.

A.1 SUMMARY GRADIENT

Applying the product rule and the log-derivative trick $\nabla_\theta \pi_\theta = \pi_\theta \nabla_\theta \log \pi_\theta$ gives the gradient decomposition in the main text:

$$\begin{aligned}
\nabla_\theta J_{\text{comp}}(\theta) &= \nabla_\theta \mathbb{E}_{m_t}[-D_t(m_t)] \\
&= -\nabla_\theta \sum_{m_t} \pi_\theta(m_t \mid h_t \oplus q_{\text{comp}}) D_t(m_t) \\
&= -\sum_{m_t} \Big[D_t(m_t) \nabla_\theta \pi_\theta(m_t \mid h_t \oplus q_{\text{comp}}) \\
&\qquad\qquad + \pi_\theta(m_t \mid h_t \oplus q_{\text{comp}}) \nabla_\theta D_t(m_t) \Big] \\
&= \underbrace{\mathbb{E}_{m_t}[-D_t(m_t) \nabla_\theta \log \pi_\theta(m_t \mid h_t \oplus q_{\text{comp}})]}_{g_{\text{sum}}} \\
&\quad + \underbrace{\mathbb{E}_{m_t}[-\nabla_\theta D_t(m_t)]}_{g_{\text{post}}}.
\end{aligned} \tag{17}$$

The first term g_{sum} is the policy gradient for summary generation, with negative distortion as its scalar reward. In practice, we estimate $D_t(m_t)$ from one sampled continuation and assign the reward $-\widehat{D}_t$ to every summary token generated by the policy.

A.2 POST-COMPACTION SELF-DISTILLATION

For the second term g_{post}, we hold the sampled summary fixed and expand the derivative through the continuation expectation:

$$\begin{aligned}
g_{\text{post}} &= -\mathbb{E}_{m_t}\left[\nabla_\theta \mathbb{E}_{\sigma_t^+ \mid m_t}\left[\sum_{i,j} d_{i,j}(\theta)\right]\right] \\
&= -\mathbb{E}_{m_t, \sigma_t^+}\left[\left(\sum_{i,j} d_{i,j}(\theta)\right) \nabla_\theta \log \pi_\theta(\sigma_t^+ \mid \tilde{h}_t) + \sum_{i,j} \nabla_\theta d_{i,j}(\theta)\right].
\end{aligned} \tag{18}$$

Here, the continuation distribution includes environment transitions, which are independent of θ. The first term accounts for how the policy changes the distribution of visited prefixes; the second differentiates the token-level KL at fixed prefixes.

Following Hübotter et al. (2026, Appendix A.1), we use per-token estimator and apply stop-gradient through continuation sampling process. Specifically, we use the surrogate

$$\overline{D}_t(m_t) = \mathbb{E}_{\sigma_t^+ \sim \mathrm{sg}[\pi_\theta(\cdot|\tilde{h}_t)]}\left[\sum_{i,j} d_{i,j}(\theta)\right], \qquad g_{\mathrm{post}}^{\mathrm{SD}} = -\mathbb{E}_{m_t}\left[\nabla_\theta \overline{D}_t(m_t)\right]. \tag{19}$$

This gives a gradient of the surrogate rather than the full continuation objective. The continuation is still sampled from the current policy but the sampling process is detached during differentiation.

To differentiate each KL, abbreviate the student and stopped teacher distributions at a fixed prefix as $p_\theta(v) = \pi_\theta(v \mid \tilde{h}_t, s_{i,j})$ and $q(v) = \mathrm{sg}[\pi_\theta(v \mid h_t, s_{i,j})]$. Then

$$\begin{aligned} -\nabla_\theta d_{i,j}(\theta) &= -\nabla_\theta \sum_v p_\theta(v) \log \frac{p_\theta(v)}{q(v)} \\ &= -\sum_v \nabla_\theta p_\theta(v) \left(\log \frac{p_\theta(v)}{q(v)} + 1\right) \\ &= \mathbb{E}_{v \sim p_\theta}\left[\log \frac{q(v)}{p_\theta(v)} \nabla_\theta \log p_\theta(v)\right], \end{aligned} \tag{20}$$

where the last equality uses $\sum_v \nabla_\theta p_\theta(v) = \nabla_\theta 1 = 0$. Using the sampled action token $a_{i,j}$ for this inner expectation yields

$$g_{\mathrm{post}}^{\mathrm{SD}} = \mathbb{E}_{m_t, \sigma_t^+}\left[\sum_{i,j} \log \frac{\pi_\theta(a_{i,j} \mid h_t, s_{i,j})}{\pi_\theta(a_{i,j} \mid \tilde{h}_t, s_{i,j})} \nabla_\theta \log \pi_\theta(a_{i,j} \mid \tilde{h}_t, s_{i,j})\right]. \tag{21}$$

A sampled continuation therefore provides a Monte Carlo estimate of this surrogate gradient, with the log-ratio treated as a fixed token-level advantage. Equivalently, this update can be interpreted by using discount factor $\gamma = 0$ for the distillation reward, where each token receives its own teacher–student log-ratio, without rewards from later tokens (Lu & Lab, 2025).

B IMPLEMENTATION OF RECAP

B.1 TRAINING ALGORITHM

Figure 5 summarizes the computation of the RECAP loss.

```python
def recap_loss(batch, weight):
    # Cached rollout scores on the sampled tokens.
    old_logp = batch.rollout_logprobs
    comp_adv = zeros_like(old_logp)
    rewards = []
    for event in batch.compactions:
        idx = event.continuation_tokens
        sampled_kl = old_logp[idx] - batch.teacher_logprobs[idx]
        rewards.append(-sampled_kl.mean())
        comp_adv[idx] = -sampled_kl / len(idx)

    # Leave-one-out baseline within each task and compaction index.
    summary_adv = leave_one_out(rewards, batch.compaction_groups)
    comp_adv[batch.summary_tokens] = broadcast(summary_adv)

    # Combine advantages before the shared clipped policy loss.
    task_adv = grpo(batch.rewards, batch.task_groups)
    advantage = (task_adv + weight * comp_adv).detach()
    logp = pi_theta.log_prob(batch)
    ratio = exp(logp - old_logp)
    loss = clipped_policy_loss(ratio, advantage)
    return aggregate(loss, batch, "trajectory-mean-token-mean")
```

Figure 5: Pseudocode for RECAP.

Implementation details. We estimate A^{task} with GRPO (Shao et al., 2024) by subtracting the task-group mean from each terminal reward and dividing by the group's reward standard deviation. This advantage is shared by all action and summary tokens in the trajectory. For g_{sum}, we estimate the baseline b_t using the leave-one-out mean (Ahmadian et al., 2024) of negative distortion rewards from other rollouts of the same task at the same compaction index. We average token losses within each complete trajectory, including all its segments, and then average over trajectories. We reuse rollout-policy π_{rollout} for the calculation in both the importance-sampling ratio $\frac{\pi_\theta}{\pi_{\text{rollout}}}$ and in the KL estimate between compressed and original contexts. For each token-level KL, we use the log-probability difference at the sampled token instead of the approximation with the top-k tokens in the vocabulary, as we did not observe a clear benefit from the latter in our experiments.

B.2 EXTENSION TO MULTIPLE COMPACTIONS

For multiple compactions, each event assigns advantages to its own summary and the following actions up to the next compaction or task termination, so the affected tokens do not overlap across events. We compute these advantages separately, combine them with the task advantages, and aggregate the resulting token losses together in the trajectory level. We always treat pre-compaction history h_t as fixed for each compaction event. Thus, distortion at later compactions does not provide a learning signal to earlier compactions, even though later histories contain earlier summaries; for example, h_t at the second compaction includes the first summary.

C Experiment Details

For each environment, we select 300 training problems of suitable difficulty using preliminary Qwen3.5-4B rollouts, favoring problems with mixed outcomes while dropping too easy questions, following He et al. (2026) and Kim et al. (2026b). We use BrowseComp-Plus (Chen et al., 2025b) for deep search and SWE-smith (Yang et al., 2025) for software engineering, evaluating on 150 held-out BrowseComp-Plus questions and all 500 SWE-bench Verified tasks (Jimenez et al., 2024). All methods share the same training subsets and environment settings.

BrowseComp-Plus. We use Qwen3-Embedding-8B for dense retrieval over the fixed BrowseComp-Plus corpus. The `search` tool returns the top five documents with their IDs, retrieval scores, and first 512 tokens; `get_document` retrieves a document by ID and returns its first 4,096 tokens. We use Qwen3.8-27B as the LLM-judge for checking semantic equivalence for the remaining answers, assigning binary reward to the agent's answer.

SWE-bench Verified. Each task runs in an isolated sandbox with `bash` for commands and tests, `str_replace_editor` for file inspection and editing, and `finish` for patch submission. File changes persist across turns; one tool call is allowed per turn. Truncated shell output retains its first and last 3,000 characters; editor output is capped at 20,000 characters. Patches are evaluated in a fresh sandbox using task-specific tests and, for SWE-bench Verified, the official grader.

Selection of λ. We select λ separately for each environment using a held-out validation set of 100 problems. We compare $\lambda \in \{0.03, 0.1, 0.3, 1.0\}$ and choose the value with the highest validation accuracy (Table 3), yielding $\lambda = 0.1$ for BrowseComp-Plus and $\lambda = 0.03$ for software engineering.

Table 3: **Selection of λ.** We evaluated RECAP with different choices of λ on the held out validation set and select the best for each environment.

	BrowseComp-Plus		**SWE-agent**	
λ	Val.	Test	Val.	Test
0.03	35.0	60.0	**44.0**	51.5 g
0.1	**40.0**	62.0	39.0	46.3
0.3	36.0	62.3	41.0	46.5
1.0	34.0	62.3	43.0	49.8

Training Details. We train Qwen3.5-4B with thinking disabled using fully asynchronous VeRL. and the hyperparameters in Table 4. Evaluation uses temperature 1.0, top-$p = 0.95$, and top-$k = 20$, with accuracy averaged over four runs. All experiments run on NVIDIA GH200 GPUs.

Table 4: Training configurations of RECAP shared across environments.

Setting	**Value**
Maximum interaction steps (BrowseComp / SWE)	100 / 200
Maximum output tokens per response	8,192
Task groups per training step	16
Rollouts per task (group size)	8
Mini-batch size (task groups)	8
Inference engine	vllm
Training temperature	1.0
Training steps	150
Optimizer	AdamW
Learning rate	1×10^{-6}
Weight decay	0.01
Warmup steps	0
Gradient clipping norm	1.0
PPO clipping $(\epsilon_{\text{low}}, \epsilon_{\text{high}})$	$(0.20, 0.28)$
Entropy	0
Reference-policy KL	0
RECAP weight λ (BrowseComp-Plus / SWE)	0.1 / 0.03

D PROMPTS

D.1 CONTEXT COMPACTION

We use one shared prompt of context compaction for both environments. At each compaction, we append the following instruction to the current conversation.

Context Compaction Prompt

The current task is paused for context compaction. For this turn, write a handoff summary for another agent that will resume the original task.

Include:

- Current progress and key decisions made
- Important context, constraints, or observations
- What remains to be done (clear next steps)
- Any critical data, examples, or references needed to continue
- Be concise, structured, and focused on helping the next agent seamlessly continue the work.

Do not call any tools in this turn. Return only the handoff summary. The entire response will be used as the summary for the next agent.

After compaction, we keep the original task prompt and add the instruction and summary below as a new user message. We also append the last k iterations, each consisting of an agent action and an environment observation. We use $k = 1$ in our experiments. The previous summary and any interactions before these k iterations are removed in the new context.

Context After Compaction

{original_task_prompt}

Another agent started to solve this problem and produced a summary of its thinking process. You also have access to the actions and observations that were obtained by that agent. Use this to build on the work that has already been done and avoid duplicating work. Here is the summary produced by the other agent, use the information in this summary to assist with your own analysis:

{summary}

{previous k iterations}

D.2 BROWSECOMP-PLUS

BrowseComp-Plus Prompt

You are a deep research agent. You need to answer the given question by interacting with a search engine, using the *search* and *get_document* tools provided. Please perform reasoning and use the tools step by step, in an interleaved manner. You may use the search and get_document tools multiple times.

NOTE:
- You should always call one tool at a time. Use a concise, targeted query containing the most discriminative clues for each tool call.
- Before each tool call, provide your reasoning process to explain what you have learned and why this is the next useful call.
- When you have found the answer, call the *finish* tool with your succinct final answer.

Question: {query}

BrowseComp-Plus Tool Schemas

```
{
  "name": "search",
  "description": "Perform a search on a knowledge source. Returns top-5
   hits with docid, score, and snippet. The snippet contains the
   document's contents and may be truncated.",
  "parameters": {
    "type": "object",
    "properties": {
      "query": {"type": "string", "description": "Search query string."}
    },
    "required": ["query"]
  }
}

{
  "name": "get_document",
  "description": "Retrieve the first 4,096 tokens of a document by its
   docid.",
  "parameters": {
    "type": "object",
    "properties": {
      "docid": {"type": "string", "description": "Document ID to
   retrieve."}
    },
    "required": ["docid"]
  }
}

{
  "name": "finish",
  "description": "Submit the succinct final answer and end the episode.",
  "parameters": {
    "type": "object",
    "properties": {
      "answer": {"type": "string", "description": "Succinct final
   answer."}
    },
    "required": ["answer"]
  }
}
```

D.3 SOFTWARE ENGINEERING

Software Engineering Task Prompt

You are an autonomous software engineer working in the repository at /testbed. Your goal is to resolve the issue described by the user with a correct and maintainable patch.

Recommended workflow

Work step by step so you can learn from command output and test results:

1. Explore the repository and read the relevant code.
2. Reproduce or clearly identify the reported problem.
3. Determine the underlying cause.
4. Edit the source code to fix the issue.
5. Run focused tests or a reproduction to verify the fix.
6. Check relevant edge cases and run broader tests when practical.
7. Submit the completed work with the finish tool.

Adapt this workflow when a step is unnecessary or impractical.

Tool-use rules

While actively solving the issue, briefly explain what you learned and why the next action is useful, then call one tool: *bash, str_replace_editor, or finish*. Do not make multiple or parallel tool calls in one response.

Use *bash* to inspect the repository, search code, run programs, and execute tests. Use *str_replace_editor* for precise file viewing, creation, and editing.

The repository filesystem persists across tool calls. Each bash call starts in /testbed, but shell directory changes and environment variables do not persist to later calls.

Avoid commands that produce unnecessarily large output. Prefer targeted file views, searches, and test commands. If a command fails, inspect its output and continue diagnosing the problem.

When the issue is fixed and adequately tested, briefly summarize the verification and call *finish* with empty arguments. After calling finish, you cannot make further changes.

Issue

{problem_statement}

The available tools are `bash(command)`, `str_replace_editor(command, path)` and `finish()` for SWE-agent.

Software Engineering Tool Schemas

```
{
  "name": "bash",
  "description": "Run one shell command in the repository. Each call
   starts a fresh bash shell in /testbed, while filesystem changes
   persist across calls.",
  "parameters": {
    "type": "object",
    "properties": {
      "command": {"type": "string", "description": "The bash command to
   run in /testbed."}
    },
    "required": ["command"],
    "additionalProperties": false
  }
}

{
  "name": "str_replace_editor",
  "description": "Custom editing tool for viewing, creating, and editing
   plain-text files. State persists for this task. File views use cat -n
   style line numbers; directory views list sorted non-hidden entries up
   to two levels. str_replace requires old_str to match exactly once.",
  "parameters": {
    "type": "object",
    "properties": {
      "command": {"type": "string", "enum": ["view", "create",
   "str_replace", "insert"], "description": "The editor operation to
   perform."},
      "path": {"type": "string", "description": "Absolute path below
   /testbed, or a relative path from /testbed (e.g. src/file.py)."},
      "file_text": {"type": "string", "description": "Complete file
   content required by create."},
      "old_str": {"type": "string", "description": "Exact unique text
   required by str_replace."},
      "new_str": {"type": "string", "description": "Replacement text for
   str_replace, or text required by insert. Omit it from str_replace to
   delete old_str."},
      "insert_line": {"type": "integer", "description": "Insert new_str
   after this one-based line. Use 0 to insert before the first line."},
      "view_range": {"type": "array", "items": {"type": "integer"},
   "minItems": 2, "maxItems": 2, "description": "Optional inclusive
   [start_line, end_line] for file view. Use -1 as end_line to read
   through the end."}
    },
    "required": ["command", "path"],
    "additionalProperties": false
  }
}

{
  "name": "finish",
  "description": "Finish the task and submit the current repository
   changes for evaluation.",
  "parameters": {
    "type": "object",
    "properties": {},
    "additionalProperties": false
  }
}
```