SCRUBAGENT: CONTEXT SCRUBBING FOR SCALABLE LONG-HORIZON LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model agents trained with reinforcement learning have achieved strong results on complex tasks requiring extended interaction. Recent work addresses the challenge of growing context by compressing or folding interaction history. These methods reduce context length while leaving the location of critical information unspecified. Valuable discoveries remain mixed with failed attempts and repeated searches. Agents face disorganized history when making decisions. This motivates us to examine how humans process long content such as movies. Viewers actively construct structured memory by noting key plot points and skimming routine scenes. Important moments are highlighted and become easier to recall later. To bridge this gap, we introduce `ScrubAgent`, which brings this principle to LLM agents as a learnable memory-writing policy for long-horizon interaction. Specifically, at each step, the agent jointly decides how to act and how to record the resulting experience: routine steps are compressed, important exploration is preserved at higher resolution, and task-critical discoveries are written as addressable keyframes for later recall. We further propose ScrubGRPO, a reinforcement learning algorithm whose process rewards guide compression and memory decisions. The resulting memory state keeps useful evidence accessible under a fixed working budget while allowing later decisions to retrieve discoveries without replaying the full trajectory. Experiments on BrowseComp-Plus and SWE-Bench Verified show that `ScrubAgent` is **(1) accurate**, reaching 54.8% and 53.6% pass@1 at 7B and 56.4% and 53.8% at 14B, and **(2) context-efficient**, reducing context by 65% at 7B. We also report trajectory-length and retrieval analyses.

1 INTRODUCTION

Large language model agents have demonstrated strong capabilities in solving complex tasks through extensive interaction with external environments Yao et al. (2023); Wei et al. (2025); Li et al. (2025). In domains such as deep research Chen et al. (2025); Wei et al. (2025) and software engineering Jimenez et al. (2024); Pan et al. (2025a), agents must execute dozens to hundreds of steps. Each step involves reasoning, tool invocation, and observation processing. The accumulated trajectory serves as working memory for subsequent decisions. However, long trajectories introduce two intertwined challenges. Computationally, attention costs grow quadratically with sequence length Vaswani et al. (2017). From a reasoning perspective, relevant information becomes increasingly difficult to locate as context expands Liu et al. (2023); Shi et al. (2023).

To make long-horizon agents practical, previous work has explored several directions for context and memory management. The first category retains the complete interaction history in the context window OpenAI (2025); Anthropic (2025). This preserves all information but incurs substantial computational overhead Xiao et al. (2024). Performance also degrades as models struggle to attend to relevant details in lengthy contexts Liu et al. (2023). The second category applies summarization to compress the history when context limits are reached Pan et al. (2025b); Zhong et al. (2023). These methods reduce context length but often compress at coarse or arbitrary boundaries, which can disrupt the agent's reasoning flow. The third category uses hierarchical structures to separate main tasks from subtasks through branching mechanisms Sun et al. (2025); Ye et al. (2025). Agents can isolate token-intensive operations into temporary branches and collapse them upon completion, providing effective subtask-level context management.

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.

Subtask boundaries need not coincide with evidence boundaries. A single branch may contain failed attempts, routine exploration, partial clues, and decisive findings, while an important clue may emerge outside an explicitly planned subtask. Existing methods reduce or restructure long histories, yet long-horizon exploration also requires a finer-grained policy for deciding which observations remain high-resolution, which interactions are procedural, and which discoveries become directly accessible evidence for later decisions.

We identify this failure mode as *salience-blind trajectory accumulation*. Standard agents record interaction histories at nearly uniform resolution. A failed search query, a redundant page opening, a partial clue, and a decisive discovery are all appended as ordinary turns in the same chronological stream. This creates a *recording and reasoning mismatch*: execution produces a time-ordered transcript, whereas later decision making requires a compact set of task-critical evidence organized by relevance, reliability, and future utility. As trajectories grow, valuable discoveries become difficult to recover because the context is long and the memory layout provides no direct handle to important evidence. Long-horizon agents require memory layouts in which important evidence is compact, salient, and directly addressable.

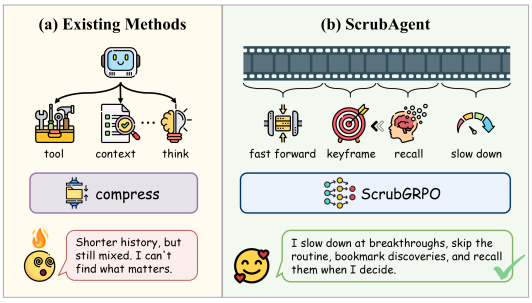


Figure 1: **Context management paradigms for long-horizon agents.** (a) Existing methods reduce or restructure long histories through compression, segmentation, or folding, but important discoveries may still remain hidden inside routine operations. (b) Humans watch movies by skimming filler scenes, attending to plot-critical moments, and bookmarking key scenes for later recall. `ScrubAgent` brings this principle to long-horizon agents through adaptive resolution and addressable keyframe memory.

This perspective suggests a different abstraction for agent memory. Humans do not process long experiences as uniform transcripts. When watching a two-hour movie, viewers do not give equal attention to every scene. They skim routine dialogue, slow down at plot-critical moments, and later recall bookmarked scenes without replaying the entire film. The key property combines compression with *adaptive resolution and addressability*: routine content is summarized, important segments are preserved at high resolution, and decisive moments are indexed for later retrieval. Current agents lack this organizational capability. They passively accumulate, summarize, or fold histories without explicitly learning how to write task-critical evidence into memory so that it remains easy to retrieve when needed.

Based on this insight, we introduce `ScrubAgent`, a framework that treats context construction as a learnable memory-writing policy. At each step, the agent decides what action to take and how the resulting experience should be recorded for future reasoning. `ScrubAgent` combines two complementary mechanisms. The first is *resolution control*: the agent can enter a focused mode to preserve full details for important exploration, and it can compress recent routine steps once their low information value becomes clear. The second is *keyframe memory*: when the agent makes a valuable discovery, it can crystallize the discovery into a written insight and store it as a retrievable keyframe. When making later decisions, the agent can recall relevant keyframes by describing what information it needs. The retrieved insights provide direct access to past discoveries without searching through the entire trajectory.

To train these capabilities, we develop **ScrubGRPO**, a reinforcement learning algorithm that extends standard GRPO with process rewards for context scrubbing. The rewards encourage timely compression of routine content, discourage unnecessary keyframe creation, and penalize compression that loses critical information. Through this training, the agent learns when to preserve high-resolution context, when to compress routine interactions, and what evidence deserves keyframe storage. `ScrubAgent` therefore transforms long-horizon context management from passive history reduction into active memory organization.

Our contributions are as follows.

❶ ***Bottleneck Identification.*** We identify *salience-blind trajectory accumulation* as a key bottleneck in long-horizon agents. Standard agents record routine exploration and decisive discoveries in the

same chronological stream, creating a recording and reasoning mismatch that makes task-critical evidence difficult to recover.

❷ ***Practical Solution.*** We propose `ScrubAgent`, a framework that formulates context management as a learnable memory-writing policy with adaptive resolution and addressable keyframe memory.

❸ ***Training Algorithm.*** We introduce **ScrubGRPO**, a reinforcement learning algorithm with process rewards that teach agents when to compress, when to preserve high-resolution context, and what evidence to index for later retrieval.

❹ ***Experimental Validation.*** Experiments on BrowseComp-Plus and SWE-Bench Verified show that `ScrubAgent` achieves 54.8% and 53.6% pass@1 with 32K active context in the 7B setting, and 56.4% and 53.8% in the 14B setting. Additional analyses cover longer trajectories, retrieval quality, and matched training and inference cost.

2 Related Work

2.1 Long-Horizon LLM Agents

Long-horizon task execution requires agents to maintain coherent reasoning over extended interaction sequences. Early approaches employed chain-of-thought prompting Wei et al. (2022) to decompose complex problems into intermediate steps. ReAct Yao et al. (2023) combined reasoning traces with tool-based actions to enable dynamic planning and environmental interaction. Multi-agent frameworks emerged to distribute task complexity across specialized roles. MetaGPT Hong et al. (2024) encoded standardized operating procedures into collaborative workflows. Reflexion Shinn et al. (2023) enhanced iterative improvement through verbal self-reflection on past trajectories. Recent work has further explored hierarchical decomposition and context management strategies. Methods like AgentGym Xi et al. (2025) provide diverse training environments for evolving agent capabilities, while branching-based approaches such as context folding Sun et al. (2025) isolate token-intensive subtasks into temporary trajectories and collapse them after completion. These advances improve planning, memory management, and collaborative mechanisms. Existing methods organize experience at the task, agent, or subtask level, whereas `ScrubAgent` learns step-level memory resolution within a trajectory and keeps task-critical evidence compact and retrievable.

2.2 Integrated and Decoupled Context Management

Recent context-management methods can be grouped by the object that they edit and the location of the policy. Decoupled curation assigns context editing to a separate curator, which provides modular control but separates credit assignment from the acting policy. External memory methods maintain a bank outside the active working context, which is useful for distilled facts in dialogue and question answering. Unconditional consolidation compresses each interaction into a fixed-size state and may discard superseded details. `ScrubAgent` uses an integrated policy that jointly emits task actions and memory writes under the deployed scrubbed context. It manages the live trajectory under a hard 32K budget through adaptive resolution and addressable keyframes. A matched internal adaptation of an external-memory manager reaches 46.2 on BrowseComp-Plus, while a matched consolidation adaptation reaches 48.9 and 49.5 on BrowseComp-Plus and SWE-Bench Verified, respectively.

2.3 Reinforcement Learning for LLM Agents

Reinforcement learning has become crucial for training agents to optimize task performance through environmental feedback. InstructGPT Ouyang et al. (2022) pioneered RLHF by applying PPO to align language models with human preferences. PPO Schulman et al. (2017) maintains policy stability through clipped objective functions but requires substantial computational overhead. DPO Rafailov et al. (2023) simplified this pipeline by directly optimizing preference data without explicit reward models. GRPO Shao et al. (2024) further advanced efficiency by normalizing rewards within trajectory groups and eliminating value networks. This approach proved effective for reasoning tasks in recent systems. AGILE Feng et al. (2024) introduced comprehensive frameworks combining memory, tools, and expert interactions. AgentGym-RL Xi et al. (2024) extended online learning to diverse interactive environments through multi-turn trajectory optimization. Recent methods also explore process-level supervision, such as tree-based search strategies that construct intermediate training signals from outcome rewards Luo et al. (2024). ScrubGRPO builds on GRPO by introducing

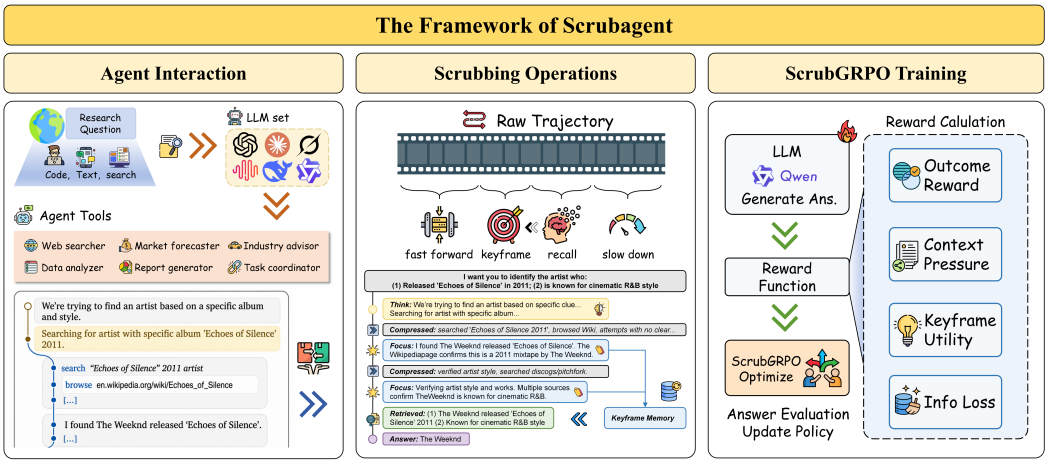


Figure 2: **Overview of `ScrubAgent`. Left**: The agent interacts with the environment and generates raw trajectories. **Middle**: Scrubbing operations organize the trajectory by preserving important segments, compressing routine steps, and storing discoveries as keyframes. **Right**: ScrubGRPO trains the agent with outcome reward and process rewards for context pressure, keyframe utility, and information preservation.

process rewards for memory-writing decisions in long-horizon agents. These rewards guide the agent to learn when to compress routine content, when to preserve high-resolution context, and what discoveries should be indexed as retrievable evidence for future reasoning.

3 Preliminary

3.1 LLM Agents and Long-Horizon Interaction

We consider a large language model agent that interacts with an external environment to solve a task query q. The agent generates a multi-turn trajectory

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

where a_t denotes the agent action at step t and o_t is the corresponding observation returned by the environment. Each action typically contains reasoning content and a tool invocation, while each observation contains the tool result. In a standard ReAct-style formulation, the agent generates

$$a_t \sim \pi_\theta(a_t \mid q, \tau_{<t}), \tag{2}$$

where $\tau_{<t} = (a_1, o_1, \ldots, a_{t-1}, o_{t-1})$ is the complete interaction history before step t. This formulation appends all previous actions and observations into the context window.

In long-horizon tasks such as deep research and software engineering, τ can grow to hundreds of steps. The resulting context creates both computational and reasoning challenges: attention cost grows quadratically with sequence length, and relevant information becomes increasingly difficult to locate in a long, noisy trajectory.

3.2 Reinforcement Learning from Verifiable Rewards

We optimize agents with reinforcement learning from verifiable rewards. Each trajectory receives a binary outcome reward $R \in \{0, 1\}$ based on task success. Group Relative Policy Optimization samples G trajectories for each task $q \sim \mathcal{D}$ and computes advantages through group normalization. The objective is

$$\mathcal{J}_{\text{GRPO}} = \mathbb{E}_{q\sim\mathcal{D}} \left[\frac{1}{G} \sum_{i=1}^{G} \frac{1}{|\tau_i|} \sum_{t=1}^{|\tau_i|} \mathcal{L}_{i,t}^{\text{GRPO}} \right], \tag{3}$$

$$\mathcal{L}_{i,t}^{\text{GRPO}} = \min\{r_{i,t}(\theta)\hat{A}_{i,t}, \text{clip}(r_{i,t}(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{i,t}\}, \tag{4}$$

where

$$r_{i,t}(\theta) = \frac{\pi_\theta(\tau_{i,t} \mid q, \tau_{i,<t})}{\pi_{\theta_{\text{old}}}(\tau_{i,t} \mid q, \tau_{i,<t})}, \tag{5}$$

and $\hat{A}_{i,t} = (R_i - \mu)/\sigma$, with μ and σ computed from the rewards of the sampled trajectory group.

4 METHODOLOGY

4.1 SCRUBBING OPERATIONS

`ScrubAgent` treats context construction as a learnable memory-writing policy. At each step, the agent jointly outputs a task action and a scrubbing decision,

$$\pi_\theta(a_t, z_t \mid q, c_t), \tag{6}$$

where z_t determines how the current experience is recorded and c_t is the scrubbed context. We design five operations, grouped into resolution control and keyframe memory.

Resolution Control. This group manages the recording resolution of trajectory segments.

(i) `focus_begin`(*reason*): starts a high-resolution window for upcoming interactions. The argument explains why the following steps may require detailed preservation.

(ii) `focus_end`(*summary*): ends the current high-resolution window. The summary synthesizes what was learned and serves as a compact record of the focused segment.

(iii) `compress_recent`(*n, note*): retroactively compresses the most recent n steps. This allows the agent to clean up routine interactions once their low information value becomes clear.

Keyframe Memory. This group manages selective long-term retention.

(iv) `keyframe`(*insight*): stores the current moment as a retrievable knowledge unit. The insight field requires the agent to distill why the moment matters, rather than storing raw observations alone.

(v) `recall`(*query*): retrieves relevant keyframes by semantic similarity. Retrieved keyframes are injected into the current context as structured observation blocks.

4.2 CONTEXT MANAGER

The context manager $\mathcal{S}$ constructs the working context c_t from the complete history and all scrubbing decisions:

$$c_t = \mathcal{S}(q, \tau_{<t}, \mathcal{W}_t, \mathcal{K}_t, \mathcal{R}_t), \tag{7}$$

where $\mathcal{W}_t$ is the set of focus windows, $\mathcal{K}_t$ is the set of keyframes, and $\mathcal{R}_t$ contains active recall results. The manager produces a compact context under the budget $L_{\max}$.

The construction follows three rules. Steps inside focus windows are preserved as full action-observation pairs. Steps outside focus windows are replaced by summaries, produced either by `compress_recent` or by an automatic summarizer. Steps with `keyframe` invocations are represented by lightweight markers, while recalled keyframes are appended as structured memory blocks.

For example, suppose the agent invokes `focus_begin` at step 3, `focus_end` at step 6, `keyframe` at step 8, and `compress_recent(4)` at step 12. The context manager transforms

$$\mathcal{S}(\tau_{1:12}) = (s_{1:2}, \tau_{3:6}, s_7, [\text{KF}]_8, s_{9:12}), \tag{8}$$

where $\tau_{3:6}$ retains the full action-observation pairs, $s_{1:2}$, s_7, and $s_{9:12}$ are compressed summaries, and $[\text{KF}]_8$ is a lightweight keyframe marker.

4.3 KEYFRAME MEMORY ARCHITECTURE

The keyframe memory $\mathcal{M}$ is a selective episodic store for past insights. Each keyframe is represented as

$$\mathcal{M}_k = (k, \text{insight}_k, \text{snippet}_k, \mathbf{e}_k), \tag{9}$$

Table 1: **Main results on BrowseComp-Plus (N=150) and SWE-Bench Verified (N=500).** We compare `ScrubAgent` with frontier models, ReAct baselines, and recent context-management methods. **Context Config** denotes the available token budget, **Comp%** denotes the compression ratio, and **Tool Calls** counts executed steps, since one step issues one tool call. Parentheses report gains or drops relative to Long Context ReAct† with Qwen2.5-7B and 327K context. `ScrubAgent` attains the highest average pass@1 among comparable Qwen2.5 methods while maintaining 65% to 67% compression.

Model	Context Config	Comp%	BrowseComp-Plus Pass@1	BrowseComp-Plus Tool Calls	SWE-Bench Verified Pass@1	SWE-Bench Verified Tool Calls
Frontier Models (ReAct)						
Claude 4 Sonnet	327K	0%	78.6	13.2	72.4	44.1
Gemini 2.5 Pro	1000K	0%	84.7	12.8	73.5	48.2
GPT-5	327K	0%	79.3	14.2	71.8	42.6
DeepSeek-V3.1	327K	0%	61.3	10.6	61.0	53.2
GLM-4.5-Air	327K	0%	56.6	11.1	57.6	51.2
ReAct Agent (Qwen2.5-7B-Instruct)						
Short Context	32K	0%	24.7	3.5	38.0	23.2
+ RL (GRPO)	32K	0%	39.3	5.1	42.4	25.6
Long Context†	327K	0%	42.0 (±0.0)	9.8	48.8 (±0.0)	45.2
+ RL (GRPO)	327K	0%	47.3 (+5.3)	9.3	50.8 (+2.0)	50.8
Context Management Methods (Qwen2.5-7B-Instruct)						
LongAgent	32K×8	∼33%	36.8 (-5.2)	13.1	43.5 (-5.3)	59.7
InfLLM	32K×8	∼36%	37.4 (-4.6)	12.9	44.2 (-4.6)	60.8
CLEX	32K×10	∼40%	38.9 (-3.1)	14.5	45.7 (-3.1)	65.3
Summary Agent	32K×10	∼40%	33.3 (-8.7)	15.8	43.2 (-5.6)	70.2
SECOM	32K×8	∼42%	40.6 (-1.4)	13.7	46.3 (-2.5)	58.9
Context Folding	32K×10	∼90%	36.7 (-5.3)	11.7	43.6 (-5.2)	66.4
+ RL (GRPO)	32K×10	∼90%	49.7 (+7.7)	14.6	49.9 (+1.1)	72.6
+ RL (FoldGRPO)	32K×10	∼90%	54.3 (+12.3)	17.5	51.4 (+2.6)	88.1
ScrubAgent (Qwen2.5-7B-Instruct, Ours)						
ScrubAgent (no RL)	32K×10	∼50%	38.9 (-3.1)	13.4	44.5 (-4.3)	68.9
+ RL (GRPO)	32K×10	∼58%	50.2 (+8.2)	15.3	50.7 (+1.9)	74.8
+ RL (ScrubGRPO)	32K×10	**65%**	**54.8 (+12.8)**	17.4	**53.6 (+4.8)**	85.2
ScrubAgent (Qwen2.5-14B-Instruct, Ours)						
+ RL (ScrubGRPO)	32K×10	**67%**	**56.4** (+14.4)	18.1	**53.8** (+5.0)	88.6

where k is the keyframe identifier, insight_k is the distilled discovery, snippet_k is a bounded context extract, and $\mathbf{e}_k$ is the embedding used for retrieval. The insight field performs cognitive compression at storage time, turning raw experience into an actionable memory unit.

When `recall(query)` is invoked, the system computes the similarity between the query embedding $\mathbf{e}_q$ and all keyframe embeddings $\{\mathbf{e}_k\}$. The top-matching keyframes are returned as structured observations containing the stored insight and a relevance explanation. The memory capacity is bounded by K, encouraging selective retention and effective keyframe usage.

4.4 SCRUBGRPO

Training effective context scrubbing requires two properties. First, the policy must be optimized under the same scrubbed context used at deployment. Second, sparse outcome rewards alone are insufficient for fine-grained memory-writing decisions.

We introduce ScrubGRPO, which extends GRPO with scrubbed-context optimization and process rewards. For each task q, the algorithm samples G trajectories from the scrubbing-augmented policy. Each trajectory receives an outcome reward $R_i \in \{0, 1\}$. The objective is

$$\mathcal{J}_{\text{ScrubGRPO}} = \mathbb{E}_{q\sim\mathcal{D}}\left[\frac{1}{\sum_i |\tau_i|}\sum_{i=1}^{G}\sum_{t=1}^{|\tau_i|}\mathbf{1}^{\text{LLM}}_{\tau_{i,t}}\mathcal{L}^{\text{Scrub}}_{i,t}\right], \tag{10}$$

$$\mathcal{L}^{\text{Scrub}}_{i,t} = \min\{r_{i,t}(\theta)\hat{A}_{i,t}, \text{clip}(r_{i,t}(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{i,t}\}, \tag{11}$$

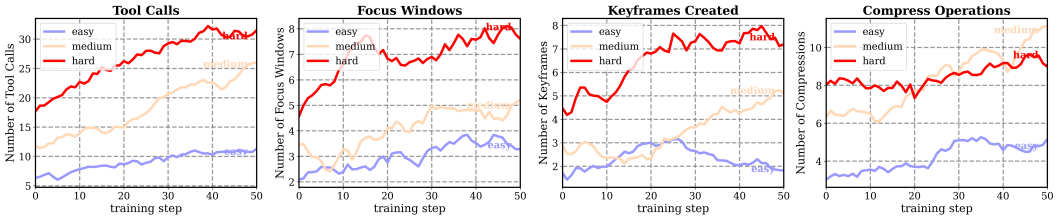


Figure 3: **Training dynamics of `ScrubAgent`.** We track four behavioral metrics on BrowseComp-Plus across training steps, stratified by task difficulty: easy ($\geq$87.5%), medium (0–87.5%), and hard (0%) based on baseline acc@8. **`ScrubAgent`** increasingly allocates memory operations to harder tasks, indicating learned task-adaptive context organization.

where the importance ratio conditions on the scrubbed context:

$$r_{i,t}(\theta) = \frac{\pi_\theta(\tau_{i,t} \mid q, c_{i,t})}{\pi_{\theta_{\text{old}}}(\tau_{i,t} \mid q, c_{i,t})}. \tag{12}$$

Here $c_{i,t} = \mathcal{S}(q, \tau_{i,<t}, \mathcal{W}, \mathcal{K}, \mathcal{R})$, and $\mathbf{1}^{\text{LLM}}_{\tau_{i,t}}$ masks out tool-observation tokens.

The advantage incorporates process rewards:

$$\hat{A}_{i,t} = \frac{\widetilde{R}_{i,t} - \mu}{\sigma}, \qquad \widetilde{R}_{i,t} = R_i + Q_{i,t}, \tag{13}$$

where μ and σ are group statistics of shaped rewards. The process reward $Q_{i,t}$ contains three components.

(i) Context Pressure Penalty. When the scrubbed context exceeds a threshold fraction α of the maximum budget, we penalize full-resolution tokens outside focus windows:

$$Q^{\text{pressure}}_{i,t} = -\lambda_p. \tag{14}$$

This encourages timely compression of routine interactions.

(ii) Keyframe Utility Penalty. At episode completion, keyframes that were never recalled receive

$$Q^{\text{keyframe}}_{i,t} = -\lambda_k, \tag{15}$$

applied to the corresponding `keyframe` operation. This discourages wasteful memory creation.

(iii) Information Loss Penalty. For failed trajectories, a lightweight judge identifies whether compressed segments lost critical information. Problematic compression operations receive

$$Q^{\text{info}}_{i,t} = -\lambda_l. \tag{16}$$

This prevents over-aggressive compression.

The total process reward is

$$Q_{i,t} = Q^{\text{pressure}}_{i,t} + Q^{\text{keyframe}}_{i,t} + Q^{\text{info}}_{i,t}. \tag{17}$$

Together, these rewards balance compression efficiency, selective memory use, and information preservation.

5 Experiments

- (**RQ1**) Does ScrubAgent outperform baselines on long-horizon tasks with limited context budget?
- (**RQ2**) How does the agent use scrubbing operations and what patterns emerge in its behavior?
- (**RQ3**) How sensitive is ScrubAgent to key hyper-parameters such as context budget and keyframe capacity?
- (**RQ4**) What is the contribution of each scrubbing operation and process reward component?

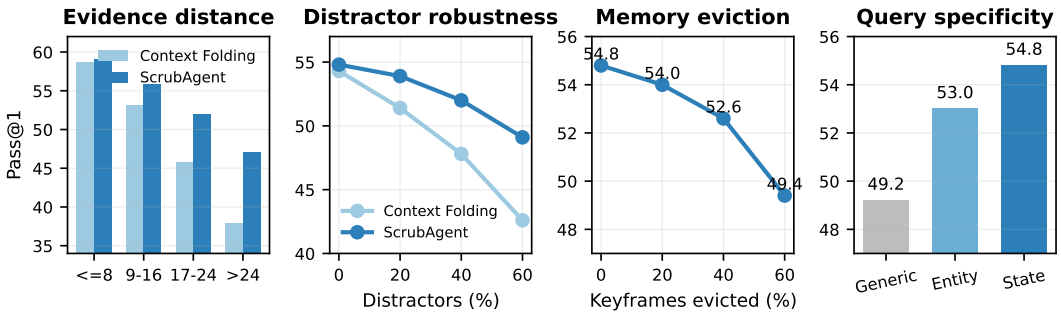


Figure 4: **Memory robustness analysis.** We vary critical-evidence distance, irrelevant-observation share, forced keyframe eviction, and recall-query specificity.

Table 2: BrowseComp-Plus pass@1 by executed trajectory length for five agent configurations. Parentheses give the task count in each method-specific bin. *Drop* is the decrease from the shortest to the longest bin, *Prefill* is the mean episode prefill in thousands of tokens, and *Recall* is the fraction of answer-bearing evidence retained or recovered by a matched probe.

Method	≤ 15 steps	15–30 steps	> 30 steps	Drop↓	Prefill (K)↓	Recall↑
Long Context ReAct + GRPO	52.0 (100)	42.0 (40)	21.5 (10)	30.5	321.8	65.1%
Summary Agent	41.0 (65)	29.0 (65)	22.0 (20)	19.0	298.4	61.8%
Context Folding + FoldGRPO	63.0 (51)	52.4 (79)	39.6 (20)	23.4	**78.6**	70.4%
`ScrubAgent` + GRPO	59.0 (42)	50.2 (79)	37.4 (29)	21.6	221.9	77.6%
`ScrubAgent` + ScrubGRPO	**63.4** (35)	**54.6** (79)	**46.8** (36)	**16.6**	210.5	**83.5%**

5.1 Experimental Settings

Datasets and Metrics. We evaluate `ScrubAgent` on 150 BrowseComp-Plus Chen et al. (2025) instances and 500 SWE-Bench Verified Jimenez et al. (2024) instances, with training data from BrowseComp-Plus, SWE-Gym Pan et al. (2025a), and SWE-Rebench Badertdinov et al. (2025). We report pass@1, tool and token efficiency, compression ratio, and operation-level memory statistics.

Baselines and Models. The matched Qwen2.5-7B-Instruct Qwen et al. (2025) comparison includes short- and long-context ReAct Yao et al. (2023), LongAgent Zhao et al. (2024), InfLLM Xiao et al. (2024), CLEX Chen et al. (2024), Summary Agent, SECOM Pan et al. (2025b), and Context Folding Sun et al. (2025). We additionally report reference-agent results from Claude 4 Sonnet, Gemini 2.5 Pro, GPT-5, DeepSeek-V3.1, and GLM-4.5-Air.

Implementation. ScrubGRPO uses VeRL with Qwen2.5-7B and 14B policies, 32 tasks and 8 rollouts per update, 50 iterations, learning rate 10^{-6}, a 32K working context, $K = 10$, and 256-token snippets. Qwen3-Embedding-8B indexes keyframes, the information-loss judge is training-only, and inference uses greedy decoding. Appendices A.2, A.1, and G give the memory protocol, training configuration, and costs.

5.2 Main Result (**RQ1**)

Table 1 reports 54.8% and 53.6% pass@1 with 65% compression at 7B, and 56.4% and 53.8% at 14B. `ScrubAgent` is comparable to FoldGRPO, the strongest matched baseline, on BrowseComp-Plus (54.8 vs. 54.3) and raises SWE-Bench pass@1 from 51.4 to 53.6 under the same budget. Appendices C and D add matched Memory-R1-style, MEM1-style, and tuned hand-crafted comparisons. The results tie the gain to a compact working context that keeps discoveries directly retrievable.

5.3 Model Analysis (**RQ2**)

Figure 3 shows that harder instances trigger more compression, focus, keyframe creation, and tool calls during training. This pattern indicates task-adaptive memory allocation. The trajectory and keyframe-utilization breakdowns are reported in Appendix F.

5.4 Memory Robustness Analysis

Figure 4 tests addressable memory. The margin grows with evidence distance and distractor share, performance degrades gradually under forced eviction, and state-aware queries raise pass@1 from 49.2 with generic queries to 54.8. The panels connect long-horizon robustness to selective recall.

5.5 Trajectory Length and Retrieval Analysis

Table 2 follows five configurations across executed-length strata. Beyond 30 steps, the full method reaches 46.8%, compared with 37.4% under GRPO and 39.6% under FoldGRPO. Each method bins the same 150 questions by its own executed steps, and Table 1 reports overall performance. Appendix F details keyframe activity, and Appendix E reports 83.5% relevant recalls and recalled insights in 76% of successful hard trajectories.

5.6 Hyperparameter Analysis (**RQ3**)

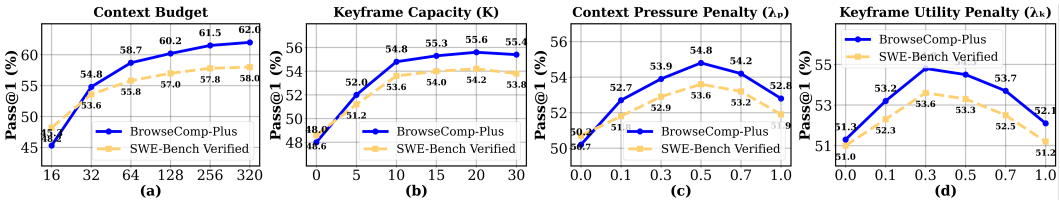


Figure 5: **Hyperparameter sensitivity.** From left to right, the panels vary context budget, keyframe capacity, context-pressure penalty, and keyframe-utility penalty.

Figure 5(a) places the 32K operating point within a wider budget sweep. Accuracy rises with context and the increment shrinks beyond 256K, as Appendix B details. The 32K setting remains the matched-budget configuration of Table 1, while the wider sweep measures the effect of additional working context. The other panels show saturation after $K = 10$ and peaks at moderate reward weights. The capacity and penalty sweeps thus locate a setting that preserves useful keyframes while keeping the active context compact. From 16K to 32K, pass@1 rises by 9.6 and 5.4 points on BrowseComp-Plus and SWE-Bench Verified, whereas from 256K to 320K it rises by 0.5 and 0.2 points.

5.7 Ablation Study (**RQ4**)

Table 3 shows that removing keyframe retrieval causes the largest drop, to 50.7 and 50.0, while compression remains at 63%. Removing compression lowers that rate to 42%, removing focus to 58%, and outcome-only rewards reach 47.3 and 48.8. Averages follow the same ordering, from 54.2 for the full method to 51.6 without focus, 51.1 without compression, and 48.1 with outcome-only rewards. These results place the gain in addressable retrieval, resolution control, and intermediate-state rewards.

Table 3: **Component ablation.** Accuracy and compression for the ScrubAgent components.

Method	BC+↑	SWE↑	Comp.↑	Avg.↑
`ScrubAgent` **(Full)**	**54.8**	**53.6**	**65%**	**54.2**
w/o Focus	52.0	51.2	58%	51.6
w/o Compress	51.3	50.8	42%	51.1
w/o Keyframe+Recall	50.7	50.0	63%	50.4
w/o Process Rewards	50.2	50.7	58%	50.5
Outcome Reward Only	47.3	48.8	0%	48.1

6 Conclusion

We present `ScrubAgent`, which learns adaptive resolution control and addressable keyframe memory to improve long-horizon performance while reducing context usage by 65% under the 7B setting.

AI Usage Statement. AI assisted only with language polishing.

REFERENCES

Anthropic. Introducing Claude 4. https://www.anthropic.com/news/claude-4, 2025. Accessed: January 2026.

Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. Swe-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents, 2025. URL https://arxiv.org/abs/2505.20411.

Guanzheng Chen, Xin Li, Zaiqiao Meng, Shangsong Liang, and Lidong Bing. CLEX: Continuous Length Extrapolation for Large Language Models. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2310.16450.

Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, Sahel Sharifymoghaddam, Yanxi Li, Haoran Hong, Xinyu Shi, Xuye Liu, Nandan Thakur, Crystina Zhang, Luyu Gao, Wenhu Chen, and Jimmy Lin. BrowseComp-Plus: A more fair and transparent evaluation benchmark of deep-research agent, 2025. URL https://arxiv.org/abs/2508.06600.

Peiyuan Feng, Yichen He, Guanhua Huang, Yuan Lin, Hanchong Zhang, Yuchen Zhang, and Hang Li. Agile: A novel reinforcement learning framework of llm agents, 2024. URL https://arxiv.org/abs/2405.14751.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, 2024.

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?, 2024. URL https://arxiv.org/abs/2310.06770.

Kuan Li, Zhongwang Zhang, Huifeng Yin, Liwen Zhang, Litu Ou, Jialong Wu, Wenbiao Yin, Baixuan Li, Zhengwei Tao, Xinyu Wang, Weizhou Shen, Junkai Zhang, Dingchu Zhang, Xixi Wu, Yong Jiang, Ming Yan, Pengjun Xie, Fei Huang, and Jingren Zhou. Websailor: Navigating super-human reasoning for web agent, 2025. URL https://arxiv.org/abs/2507.02592.

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*, 11:157–173, 2023.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, et al. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*, 2024.

OpenAI. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025. URL https://arxiv.org/abs/2601.03267.

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*, 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.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training Software Engineering Agents and Verifiers with SWE-Gym. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025a. arXiv:2412.21139.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Xufang Luo, Hao Cheng, Dongsheng Li, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Jianfeng Gao. SeCom: On Memory Construction and Retrieval for Personalized Conversational Agents. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025b. URL https://openreview.net/forum?id=xKDZAW0He3.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

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, Y. K. Li, Y. Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H. Chi, Nathanael Schärli, and Denny Zhou. Large Language Models Can Be Easily Distracted by Irrelevant Context. *arXiv preprint arXiv:2302.00093*, 2023.

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*, volume 36, 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.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All You Need. In *Advances in Neural Information Processing Systems 30 (NeurIPS)*, 2017.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, 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*, volume 35, pp. 24824–24837, 2022.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isabella Fulford, Hyung Won Chung, Alexandre Passos, William Fedus, and Amelia Glaese. BrowseComp: A Simple Yet Challenging Benchmark for Browsing Agents. *arXiv preprint arXiv:2504.12516*, 2025.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Dingwen Yang, Chenyang Liao, Xin Guo, Wei He, Songyang Gao, Lu Chen, Rui Zheng, Yicheng Zou, Tao Gui, Qi Zhang, Xipeng Qiu, Xuanjing Huang, Zuxuan Wu, and Yu-Gang Jiang. Agentgym: Evolving large language model-based agents across diverse environments. *arXiv preprint arXiv:2406.04151*, 2024.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Xin Guo, Dingwen Yang, Chenyang Liao, Wei He, Songyang Gao, Lu Chen, Rui Zheng, Yicheng Zou, Tao Gui, Qi Zhang, Xipeng Qiu, Xuanjing Huang, Zuxuan Wu, and Yu-Gang Jiang. Agentgym: Evaluating and training large language model-based agents across diverse environments. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pp. 27914–27961, 2025.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. Infllm: Training-free long-context extrapolation for llms with an efficient context memory, 2024. URL https://arxiv.org/abs/2402.04617.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, et al. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. *arXiv preprint arXiv:2508.19828*, 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 *The Eleventh International Conference on Learning Representations (ICLR)*, 2023. URL https://arxiv.org/abs/2210.03629.

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

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, et al. Memagent: Reshaping long-context llm with multi-conv RL-based memory agent. In *International Conference on Learning Representations (ICLR)*, 2026. Oral presentation; arXiv:2507.02259.

Jun Zhao, Can Zu, Hao Xu, Yi Lu, Wei He, Yiwen Ding, Tao Gui, Qi Zhang, and Xuanjing Huang. LongAgent: Scaling Language Models to 128k Context through Multi-Agent Collaboration. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 16678–16697, 2024.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory, 2023. URL https://arxiv.org/abs/2305.10250.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, et al. MEM1: Learning to synergize memory and reasoning for efficient long-horizon agents. *arXiv preprint arXiv:2506.15841*, 2025.

Appendix

A Implementation Details and the Scrubbing Procedure **14**
A.1 Model, Environment, and Training Configuration 14
A.2 Control Flow of the Five Operations 14
A.3 The Information-Loss Judge 14
A.4 Operation Serialization and Parsing 16
A.5 The GRPO Objective 16
B Extended Context-Budget Sweep **16**
C Adaptations of Learned Memory Policies **17**
D Tuned Hand-Crafted Compaction Policy **18**
E Retrieval Quality of Keyframe Memory **18**
E.1 Case Study on Failed Retrieval 19
F Results by Trajectory Length **20**
G Compute Cost and Overhead Accounting **20**
H Variance, Seed Protocol, and Significance Testing **21**
I Prompts, Data Construction, and Evaluation Protocol **22**
I.1 Benchmark Statistics 22
I.2 Role Prompts 23
I.3 Data Construction 24
I.4 Evaluation Protocol 24
J Portability of the Operation Set **25**
K Limitations, Significance, and Resources **25**

A Implementation Details and the Scrubbing Procedure

This appendix provides the configuration behind every number in Sec. 5 and the control flow of the context manager, to support the claims of Sec. 4. All quantities are measured on the same runs that produce Table 1. Qwen2.5-7B-Instruct is the policy unless a row states otherwise, and the 14B run reuses the configuration with the backbone swapped.

A.1 Model, Environment, and Training Configuration

Table 4 gives the operating configuration for the main comparison. The policy is trained under the scrubbed context $c_{i,t}$ used at deployment. The retriever is frozen, uses cosine similarity with top-m recall, and returns 4,096-dimensional embeddings. The policy itself produces compression summaries. The information-loss judge uses temperature zero and a 180-token response cap only when computing the reward of failed trajectories, and inference uses greedy decoding.

Table 4: Configuration of `ScrubAgent` and ScrubGRPO.

Models and memory	
Main / scaled policy	Qwen2.5-7B / 14B-Instruct
Retriever	Qwen3-Embedding-8B
Working budget	32,768 tokens, 10 windows
Trajectory cap	327,680 tokens
Recall depth / capacity	$m = 3$ / $K = 10$
Snippet cap	256 tokens
Info-loss judge	GPT-4o-mini
ScrubGRPO training	
Framework / engine	VeRL (FSDP) / vLLM
Hardware	8×A100 80GB
Tasks / rollouts	32 / 8 per update
Iterations	50 (≈2 epochs)
Learning rate	1×10^{-6}
Clip low / high	0.20 / 0.28
Reward weights	$\lambda_p = 0.5, \lambda_k = 0.3, \lambda_l = 0.5$
Pressure threshold	$\alpha = 0.5$
Random seeds	0, 1, 2
Environment	
Research steps (train/eval)	30 / 60
SWE steps (train/eval)	100 / 200
Action tokens / timeout	1,024 / 120 s
Research tools	`search`, `open_page`
SWE tools	`execute_bash`, `think`
SWE editor	`str_replace_editor`
Retry / overflow	1 / `compress_recent`

The rollout engine allocates 0.60 of GPU memory to its KV cache. Each update contains 256 trajectories from 32 tasks and eight rollouts per task. AdamW uses $\beta = (0.9, 0.999)$, weight decay zero, a 0.03 warmup ratio, and KL coefficient zero. Rollouts use temperature 1.0 and top-p 1.0. Advantages are normalized within groups with $\epsilon = 10^{-6}$, and the observation-token mask is $\mathbf{1}^{\mathrm{LLM}}$.

We run every method on ten windows of 32,768 tokens over one episode, the 32K×10 entry of Table 1. The cumulative trajectory budget is then 327,680 tokens, which matches the single 327K window of the long-context ReAct reference to within 0.2%. Every context-management baseline with a 32K×10 entry in Table 1 follows this schedule. We define the compression ratio as

$$\text{Comp\%} \;=\; 1 - \frac{\sum_t |c_t|}{\sum_t |\tau_{<t}|}, \tag{18}$$

averaged over decision steps and trajectories, where the denominator is the transcript the same rollout would have occupied at full resolution. Under Eq. (18) the reported 65% for `ScrubAgent` means an average active context of 12.1K tokens at the step count at which the long-context reference reaches 34.6K tokens. The per-query accounting is detailed in Appendix G.

A.2 Control Flow of the Five Operations

One decision step of `ScrubAgent` is presented in full in Algorithm 1, including the eviction rule that keeps the store inside capacity and the bookkeeping that feeds the keyframe utility penalty. The context manager rebuild is presented in Algorithm 2. Both run at every step, and the summarizer Σ in them is the policy itself under greedy decoding, so no auxiliary model enters the loop. Furthermore, Σ writes a maximum of 512 tokens for every summary.

A.3 The Information-Loss Judge

Q^{info} is the only reward term that calls an external model, and it runs on failed trajectories alone. The judge reads the compressed record and the failure signal. It never reads the removed spans, so a

Algorithm 1 One `ScrubAgent` decision step: joint action and scrub dispatch

Require: query q, policy π_θ, budget $L_{\max}$, capacity K, snippet cap B_{sn}, retriever f_{emb}, summarizer Σ
Ensure: action a_t, scrubbed context c_{t+1}

sample $(a_t, z_t) \sim \pi_\theta(\cdot \mid q, c_t)$ ▷ $z_t \in \{\text{five operations, none}\}$
execute a_t, receive o_t, append $e_t = (a_t, o_t)$ to τ
if $z_t = \texttt{focus_begin}(reason)$ **then**
    open window $w \leftarrow [t, \infty)$ with note $reason$; $\mathcal{W} \leftarrow \mathcal{W} \cup \{w\}$
else if $z_t = \texttt{focus_end}(summary)$ **then**
    close the open window at t and record $summary$ as the segment record s_w
else if $z_t = \texttt{compress_recent}(n, note)$ **then**
    $\mathcal{C} \leftarrow$ the last n recorded steps that lie outside every open window
    $s_{\mathcal{C}} \leftarrow \Sigma(\mathcal{C}, note)$; drop $\mathcal{C}$ from full resolution
else if $z_t = \texttt{keyframe}(insight)$ **then**
    $snap \leftarrow$ last B_{sn} tokens of e_t; $\mathbf{e} \leftarrow f_{\text{emb}}(insight \,\|\, snap)$
    $\mathcal{K} \leftarrow \mathcal{K} \cup \{(insight, snap, \mathbf{e})\}$, $\text{used}(\cdot) \leftarrow 0$
    if $|\mathcal{K}| > K$ **then**
        evict $\arg\min_k \big(\text{used}(k),\ \text{age}(k)\big)$ ▷ least recalled, then oldest
    end if
else if $z_t = \texttt{recall}(query)$ **then**
    $\mathcal{R} \leftarrow \text{top-}m\ \{\cos(f_{\text{emb}}(query), \mathbf{e}_k) : k \in \mathcal{K}\}$
    $\text{used}(k) \mathrel{+}= 1$ for every returned k ▷ feeds Q^{keyframe}
end if
if $|c_t| > \alpha L_{\max}$ **then**
    $Q^{\text{pressure}} \leftarrow -\lambda_p$ on full-resolution steps outside windows
end if
$c_{t+1} \leftarrow \mathcal{S}(q, \tau, \mathcal{W}, \mathcal{K}, \mathcal{R})$ ▷ Algorithm 2

Algorithm 2 Context manager rebuild $\mathcal{S}$

Require: q, transcript τ, windows $\mathcal{W}$, store $\mathcal{K}$, recall set $\mathcal{R}$
$\mathcal{B} \leftarrow [q]$ ▷ query block is never compressed
for each recorded step e_i in chronological order **do**
    if e_i lies inside an open window or in the most recent $\Delta{=}4$ steps **then**
        append e_i at full resolution
    else if e_i carries a `keyframe` call **then**
        append the marker $[\text{KF}]_i$ with the insight headline only
    else if e_i carries a summary s_j **then**
        append s_j
    else
        queue e_i for automatic summarization by Σ
    end if
end for
append each $k \in \mathcal{R}$ as a `<recalled_keyframe>` block: insight, snippet, step id
while $|\mathcal{B}| > L_{\max}$ **do**
    compress the longest routine run that lies outside every window and away from $\mathcal{R}$
end while
return $\mathcal{B}$

positive verdict states that the record as written failed to carry the evidence, which is the event the penalty targets. The exact prompt used by the judge is:

> **System.** You audit one run of an agent that solves a task with a tool environment and compresses its own interaction history. You receive the task query, the final answer, the compressed records the agent kept, and the list of compression operations it issued. Decide whether a compression operation removed information that the agent still needed and could not recover from any kept record. Judge the record as written. Answer with a single JSON object and nothing else.

User. Query: `{query}`
Final answer: `{final_answer}`
Verifier result: `{fail_reason}`
Kept records, in order: `{kept_summaries_and_keyframes}`
Compression operations: `{op_list}` (each entry gives an id, the step range it replaced, and the note the agent wrote)
Rules. (1) `info_loss` is `true` only when the missing information is verifiable from the query and the answer, with the number of such facts capped at 3. (2) Attribute each missing fact to the single operation id whose step range contained it; use `-1` when no operation is responsible. (3) Ignore a missing fact whenever a kept record or a keyframe insight already states it. (4) Do not penalize compression of routine navigation, repeated searches, or tool output the agent read and then acted on.
Output JSON, one object per verdict:

```
{"info_loss": true|false,
"facts": [{"op_id": int, "missing": str}]}
```

Each `facts` entry contributes one $-\lambda_l$ to the operation named by `op_id`, and an episode contributes at most three such entries, so $Q^{\text{info}} \in [-1.5, 0]$ with $\lambda_l = 0.5$. To assess alignment between the judge and human labeling, we perform a manual inspection of 200 failed trajectories sampled from evaluation runs, and two annotators review each trajectory independently. We then compute exact-match agreement against the human majority label. The judge and the human majority agree on 88.0% of cases, with Cohen's κ of 0.79, and the two annotators agree with each other on 90.5% ($\kappa = 0.81$). A swapped judge from another family shifts the BrowseComp-Plus pass@1 of the final policy by 0.4 points, and the reward signal keeps its sign on 96.4% of audited episodes.

A.4 Operation Serialization and Parsing

In our experiments, we emit the five operations as a single JSON object that follows the thought and the tool call in the same assistant turn, which is the standard tool-calling format used by ReAct-style agents. The policy therefore produces one structured decision per step and the parser never needs a second model call. Malformed payloads trigger one retry with the parse error appended, and a second failure is treated as `none`. The fraction of steps that end in a forced `none` is 0.8% on BrowseComp-Plus and 1.3% on SWE-Bench Verified at the end of training, with a value around 9% in the first iteration.

A.5 The GRPO Objective

Sec. 3 states the two objects ScrubGRPO builds on. The exact form is:

$$\mathcal{J}_{\text{GRPO}} = \mathbb{E}_{q\sim\mathcal{D}}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|\tau_i|}\sum_{t=1}^{|\tau_i|}\mathcal{L}_{i,t}^{\text{GRPO}}\right], \tag{19}$$

$$\mathcal{L}_{i,t}^{\text{GRPO}} = \min\Big(r_{i,t}(\theta)\hat{A}_{i,t}, \quad \text{clip}(r_{i,t}(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{i,t}\Big), \tag{20}$$

where the importance ratio conditions on the full history,

$$r_{i,t}(\theta) = \frac{\pi_\theta(\tau_{i,t} \mid q, \tau_{i,<t})}{\pi_{\theta_{\text{old}}}(\tau_{i,t} \mid q, \tau_{i,<t})}, \tag{21}$$

and $\hat{A}_{i,t} = (R_i - \mu)/\sigma$, with μ and σ computed from the rewards of the sampled trajectory group. Eq. (19) differs from the ScrubGRPO objective of Eq. (10) in the conditioning variable of $r_{i,t}$ and in the advantage $\hat{A}_{i,t}$, which carries the process reward $Q_{i,t}$.

B Extended Context-Budget Sweep

The working-context budget is 32K in the matched comparisons. The extended sweep records pass@1 over six active-budget settings from 16K to 320K. The difference columns in Table 5 use the 32K

Table 6: Learned and hand-designed memory policies adapted to our setting. **Object** names what the policy writes, **Trigger** when it writes, and **LRN** whether the writing decision is trained. Avg. is the mean pass@1 across the two benchmarks, the same convention as Table 3. Every row uses Qwen2.5-7B-Instruct and the 32K×10 budget.

System	Object	Trigger	LRN	BC-Plus	SWE-Bench	Avg.
MemGPT Packer et al. (2023)	external store	paging call	✗	39.8	44.0	41.9
Memory-R1 Yan et al. (2025)	fact bank	per episode	✔	46.2	45.1	45.7
MemAgent Yu et al. (2026)	one state slot	per segment	✔	44.6	47.8	46.2
MEM1 Zhou et al. (2025)	one state slot	per turn	✔	48.9	49.5	49.2
Context Folding + FoldGRPO	subtask branch	planned branch	✔	54.3	51.4	52.9
`ScrubAgent` (**ScrubGRPO**)	live transcript	per step	✔	**54.8**	**53.6**	**54.2**

point as their reference, so the effect of additional context can be read on the same scale as the operating configuration.

Table 5: Pass@1 across working-context budgets. Differences are percentage-point changes from 32K.

Budget	BC-Plus	SWE-Bench	Δ BC-Plus	Δ SWE-Bench
16K	45.2	48.2	−9.6	−5.4
32K	**54.8**	**53.6**	0.0	0.0
64K	58.7	55.8	+3.9	+2.2
128K	60.2	57.0	+5.4	+3.4
256K	61.5	57.8	+6.7	+4.2
320K	62.0	58.0	+7.2	+4.4

Increasing the budget from 32K to 256K raises pass@1 by 6.7 points on BrowseComp-Plus and 4.2 points on SWE-Bench Verified. The final increase from 256K to 320K adds 0.5 and 0.2 points, respectively. The sweep thus locates the largest budget gains below 256K while preserving 32K as the operating point for the matched comparisons. The remaining parameter grids are $K \in \{0, 5, 10, 15, 20, 30\}$, $\lambda_p \in \{0, 0.1, 0.3, 0.5, 0.7, 1.0\}$, and $\lambda_k \in \{0, 0.1, 0.3, 0.5, 0.7, 1.0\}$. The reported configuration uses $K = 10$, $\lambda_p = 0.5$, and $\lambda_k = 0.3$.

C Adaptations of Learned Memory Policies under a Matched Budget

Sec. 2 separates `ScrubAgent` from learned memory policies by the object each one controls. We apply MemGPT, Memory-R1, MEM1 and MemAgent in the context-scrubbing comparison and re-implement all four on the training stack of Sec. 5, including the backbone, the tool API, the trajectory cap, the cumulative 32K×10 budget, the rollout count and the 50-iteration schedule. Each adapted system keeps its own memory structure and its own training objective intact, and each control signal is tuned over the grid in Table 7. Following their procedures, we evaluate the four systems on the same test tasks.

We highlight two readings of Table 6. First, the ordering tracks the granularity of the record a policy is allowed to write: an overwrite slot refreshed per turn reaches 49.2 average, a fact bank curated per episode reaches 45.7, and a step-level resolution decision over the live transcript reaches 54.2. The single-slot systems degrade in the regime where the transcript is longest, since evidence from step 4 can only survive inside the most recent consolidation. Second, Memory-R1 asks 1.06× our GPU-hours because its curator and its answer policy are trained in sequence, and it lands 8.5 average points below `ScrubAgent`. The decoupled curator is the weakest trained system in the group, and we attribute this to its reader and writer being optimized against different objectives.

Table 7: Tuning grid and the operating point chosen for each adapted system, together with the GPU-hours of its training run. The `ScrubAgent` run consumes 680 A100-hours (Appendix G).

System	Tuned quantity	Grid	Chosen	A100-h
Memory-R1	bank size, curator lr	$\{20, 50, 100\}$, $\{1, 5\}\times10^{-6}$	50, 5×10^{-6}	720
MEM1	state budget	$\{256, 512, 1024\}$ tokens	512	640
MemAgent	segment count, slot size	$\{5, 10, 20\}$, $\{128, 256\}$	10, 256	700
MemGPT	store size, page-in calls	$\{64, 128, 256\}$K, $\{1, 3\}$	128K, 3	0
`ScrubAgent`	K, $L_{\max}$, λ_p	App. B	10, 32K, 0.5	680

Table 8: The hand-crafted policy built up rule by rule, with pass@1 on BrowseComp-Plus and SWE-Bench Verified, and the same policy split by task difficulty against `ScrubAgent` on BrowseComp-Plus. Difficulty strata follow Fig. 3 and contain 39, 61, and 50 evaluation tasks.

Configuration	BC-Plus	SWE-Bench	Avg.
Full transcript at 32K (truncation)	24.7	38.0	31.4
+ R1 (Summary Agent)	33.3	43.2	38.3
+ R2 protected classes	37.6	45.1	41.4
+ R3 fixed notepad	39.8	46.0	42.9
+ R4 warmup guard (tuned)	41.5	46.8	44.2
`ScrubAgent` **(ScrubGRPO)**	**54.8**	**53.6**	**54.2**
By difficulty, BrowseComp-Plus pass@1			
Subset	Easy	Medium	Hard
Tuned hand-crafted	59.4	42.5	26.3
`ScrubAgent`	**78.4**	**56.1**	**34.7**

D A Tuned Hand-Crafted Compaction Policy

Summary Agent in Table 1 is the hand-crafted compaction policy that production agents ship, the summarize-when-full rule described in Sec. 5. We strengthen it along the four axes those agents use and tune every constant on the training split. The backbone, tool API, and 32K$\times$10 budget are identical to `ScrubAgent`.

- **R1 summarize-when-full.** When the active window passes $L_{\max}$, replace the oldest k steps by a summary. Tuned $k \in \{2, 4, 6, 8\}$, chosen $k = 4$. This rule alone reproduces Summary Agent.
- **R2 protected classes.** Never compress the most recent tool result of a search, the last error traceback, or the answer candidates seen so far.
- **R3 fixed notepad.** Maintain a 512-token scratch region that the model rewrites at every rollover, holding the working plan and the entities collected so far.
- **R4 warmup guard.** Keep the first 4 steps at full resolution so the initial decomposition survives.

Table 8 reports the rule-by-rule build up. Each rule raises pass@1, and the fully tuned policy still trails `ScrubAgent` by 13.3 points on BrowseComp-Plus and 6.8 on SWE-Bench Verified. We conducted a further analysis focusing on the difficulty strata where that gap sits. The hand-crafted policy compresses on a window counter, so it writes the same record shape on a 6-step task and on a 40-step task, and its hard-subset pass@1 of 26.3 sits 8.4 points below `ScrubAgent`. Table 14 shows the learned policy conditioning the same decisions on difficulty, with 1.8 keyframes and 46.8% utilization on easy tasks against 4.8 and 70.2% on hard ones. The tuned policy also keeps the compression ratio competitive, at 59% against our 65%, so the remaining gap comes from the placement of the retained content inside the budget.

E Retrieval Quality of Keyframe Memory

Keyframe retrieval is the component with the largest average effect in Table 3. To better understand when and why retrieval helps, we conduct two analyses: (1) a labeling study of 1,035 logged `recall`

Table 9: Retrieval quality on BrowseComp-Plus over 1,035 logged `recall` events. An event is **relevant** when the returned insight narrows at least one decision the agent faces at that step. Rates that are averaged over trajectories use the 39/61/50 difficulty split.

Quantity	Value
Returned keyframe relevant to the current observation	83.5%
easy / medium / hard trajectories	72.3 / 86.7 / 88.3
Relevant to the query as a whole	89.4%
Precision at rank 1	87.3%
Mean rank of the first relevant keyframe	1.7
Near-duplicate of an already recalled keyframe	6.9%
Needed evidence spans covered by the returned top-3	61.3%
same measure over the full transcript	79.5%
same measure over 3 random keyframes	12.8%
Successful hard trajectories quoting a recalled insight	76.0%
Relevance on trajectories longer than 30 steps	85.1%

Table 10: The retriever and the store swapped independently. **Rel.** is the relevance rate of Table 9, and pass@1 is on BrowseComp-Plus. The no-memory row is the *w/o Keyframe+Recall* variant of Table 3, and the `random` row returns three stored keyframes sampled without replacement.

Retriever / store	Rel.↑	Pass@1↑	Δ
Qwen3-Embedding-8B (default)	83.5%	**54.8**	–
bge-large-en-v1.5	81.9%	54.1	(-0.7)
`text-embedding-3-small`	78.4%	53.6	(-1.2)
BM25 over insight text	71.2%	51.7	(-3.1)
Random keyframes from the same store	21.5%	47.9	(-6.9)
No keyframe memory	n/a	50.7	(-4.1)

events examining when a returned keyframe narrows a decision the agent faces, and (2) a swap analysis investigating how the embedding model and the store shift pass@1. We log every `recall` call in the three evaluation seeds on BrowseComp-Plus, and each event holds the query text the agent wrote, the active context at that step, and the returned keyframe with its stored insight and snippet. GPT-4o-mini labels all 1,035 events under the protocol of Appendix A.3, and two annotators label a 240-event subset. The judge and the human majority agree on 86.3% of the subset, with Cohen's κ of 0.77, and the two annotators agree on 89.2% ($\kappa = 0.78$).

We highlight three readings of Table 9. The relevance rate rises monotonically with task difficulty, from 72.3% on easy trajectories to 88.3% on hard ones and 85.1% beyond the training horizon, so the retriever is most accurate exactly where Table 2 measures the largest gain. Swapping the frozen embedding model costs little: bge-large-en-v1.5 gives up 0.7 pass@1 and 1.6 relevance points, the 1,536-dimensional OpenAI small embedding gives up 1.2 points, and lexical BM25 matching over the insight text costs 3.1 points, which locates the remaining sensitivity in the query understanding of the encoder behind the retrieval step. Sampling three keyframes at random from the same store gives 47.9, below the 50.7 of running with no memory at all, so a wrong keyframe occupies budget and misleads the decision, and addressability is what carries the gain.

E.1 Case Study on Failed Retrieval

We first conducted a case study assigning every irrelevant event to one cause. Over-broad queries dominate: 58.6% of irrelevant events come from a `recall` query that drops a constraint the agent already held, 21.8% from an insight the trajectory later superseded, 12.4% from an embedding that confuses an entity with the page that mentions it, and 7.2% from near-duplicates of a keyframe already in context. The distribution supports the design choice behind Q^{keyframe}, since a superseded or duplicated keyframe is never recalled and receives the utility penalty. Over-broad queries recover quickly: in 41.7% of those events the agent issues a narrower `recall` within five steps, in 33.9% it falls back to `search`, and in 24.4% the episode closes with the evidence unrecovered. Table 11

Table 11: One failure case from the log, abridged. The recall query at step 24 drops the concert constraint, and the store cannot answer a question the agent never wrote down.

Query	Which conductor led the 1968 festival premiere of the score written for the film in which the second-named artist appears?
Step 9	`keyframe` with insight "the composer of the film is Bernard Herrmann". Snippet keeps the search result listing the cast.
Step 24	`recall(query="Herrmann conducting work")`. Returned: the step 9 keyframe plus two festival pages unrelated to 1968. Relevance label: irrelevant.
Effect	The agent had already compressed the step 11 to 16 page visits that carried the year, so the returned block confirms the composer and answers nothing about the festival.
Recovery	Step 31 opens `focus_begin`, re-issues `search` with the year in the query, and closes the episode correctly at step 38. Cost: 7 steps and 21.4K tokens of extra prefill.

Table 12: Keyframe writing and recall activity across the **`ScrubAgent`** trajectory-length bins of Table 2. Values are means per trajectory.

Steps	**Tasks**	**Keyframes written**	**Recalls issued**
≤ 15	35	1.7	0.9
15–30	79	3.1	2.1
> 30	36	5.4	4.1

provides an illustrative example of how one dropped constraint leads to an irrelevant recall and to the recovery steps that follow.

F Results by Trajectory Length

The main text reports BrowseComp-Plus pass@1 in three trajectory-length bins for five configurations in Table 2. All configurations use the same 150 questions, and each completed trajectory is assigned to a bin using its own executed step count. The five rows contribute $100/40/10$, $65/65/20$, $51/79/20$, $42/79/29$, and $35/79/36$ trajectories across the bins, respectively. Weighting the displayed conditional rates by each row's bin counts gives 47.3, 33.3, 54.3, 50.2, and 54.8 after rounding, matching Table 1. We use the full **`ScrubAgent`** bins to examine memory activity in Table 12. Training rollouts are capped at 30 steps on BrowseComp-Plus and 100 steps on SWE-Bench Verified, while evaluation allows 60 and 200. The SWE-Bench Verified analysis in Table 13 provides a separate supplementary breakdown. Its bins are defined for that benchmark and do not replace the BrowseComp-Plus bins in the main text.

The conditional BrowseComp-Plus pass@1 separation is largest beyond 30 executed steps, as reported in Table 2. Table 12 adds a memory-activity view of the **`ScrubAgent`** bins: keyframes written per trajectory rise from 1.7 to 5.4 and recalls from 0.9 to 4.1. The supplementary SWE-Bench Verified breakdown in Table 13 shows a larger separation in its longest bin. These observations connect the main-text memory-robustness panels in Fig. 4 to the increasing use of addressable memory over longer trajectories.

G Compute Cost and Overhead Accounting

To assess efficiency and training cost, we report the number of processed tokens, the wall-clock time, the embedding and retrieval overhead, and the total GPU hours of **`ScrubAgent`** and of the strongest budget-matched baselines on BrowseComp-Plus and SWE-Bench Verified. All measurements come from the 7B runs of Table 1, conducted on nodes of $8\times$A100 80GB GPUs.

Training. ScrubGRPO adds 70 A100-hours over the ReAct + GRPO control of Table 15, a $1.11\times$ factor, and the extra time sits in the reward computation, with rollout generation unchanged. Against

Table 13: Pass@1 by executed trajectory length on SWE-Bench Verified (500 tasks). Buckets use fixed step boundaries at 60 and 110 on the pooled step count.

			Pass@1			
Steps	**Tasks**	**Pooled mean**	327K ReAct + RL	Summary Agent	FoldGRPO	`ScrubAgent`
$\leq$60	168	38.5	66.1	56.4	68.5	68.5
61-110	193	72.0	48.2	40.6	48.6	50.3
$>$110	139	118.2	35.9	30.9	34.6	40.2
All	500	73.6	50.8	43.2	51.4	**53.6**

Table 14: Keyframe utilization by difficulty stratum on BrowseComp-Plus.

Difficulty	Keyframes / trajectory	Recalls / trajectory	Utilization
Easy	1.8	1.1	46.8%
Medium	3.1	2.0	58.9%
Hard	4.8	3.6	70.2%

Context Folding with FoldGRPO, the strongest budget-matched competitor, the gap is $1.07\times$. The 7B ScrubGRPO run requires 680 A100-hours on a node of $8\times$A100 80GB GPUs.

Inference. The active window shrinks by 65% per step, and the cumulative saving is smaller than that factor because the trained policy takes more steps. Relative to the 327K reference at the same backbone, `ScrubAgent` uses $1.53\times$ less prefill (210.5K compared to 321.8K tokens) and $1.63\times$ less model time (38.6 s compared to 63.1 s) on BrowseComp-Plus, and $1.70\times$ less prefill and $2.01\times$ less model time on SWE-Bench Verified, as Table 16 lists. The scrubbed policy is cheaper to serve than the full-context reference even though it takes $1.87\times$ the steps to answer the same query, because it holds $2.86\times$ fewer tokens at each step. Relative FLOPs count the linear projection term at 14 GFLOP per token for the 7B backbone and the attention term at $0.4008 \times L_{\text{eff}}$ GFLOP per token, where $L_{\text{eff}} = \sqrt{\mathbb{E}[L^2]}$ is taken over the steps of an episode. On this accounting, `ScrubAgent` does $2.4\times$ less work per BrowseComp-Plus query and $3.0\times$ less per SWE-Bench query than the full-context reference it outperforms by 7.5 and 2.8 pass@1. End-to-end latency narrows to $1.13\times$ because each additional step also carries a 1.9 s tool round trip. The 252,000 GPU-seconds of the 70-hour process-reward overhead therefore return after 10,300 BrowseComp-Plus queries served with the 24.5 s of saved model time each, and a full 680-hour run buys 100,000 such queries.

Memory path. One trajectory of a medium-difficulty task writes 3.1 keyframes and issues 2.0 recalls (Table 14), and averaged over the 150 evaluation tasks those two counts are 3.33 and 2.30. Each of the resulting 5.63 encoder calls costs 47 ms on the 8B retriever, so the whole addressable-memory path adds 265 ms to a 71.7 s query, and the similarity scan over the ten stored vectors costs 0.03 ms against an average model step of 2.2 s. **Judge.** Q^{info} is called on failed trajectories alone, which is 7,424 of the 12,800 training episodes at the observed 58.0% mean failure rate over the 50 iterations. Each call carries 10.0K input tokens and returns 180, giving 74.2M input and 1.34M output tokens at \$0.15 and \$0.60 per million, so the whole run costs \$11.9 and the judge never runs at inference. Context Folding with FoldGRPO reaches 3.5K mean active tokens through branch collapsing, the most aggressive ratio in Table 1. The addressable store keeps 8.6K more tokens visible per step and converts them into the longest stratum of Table 12.

H Variance, Seed Protocol, and Significance Testing

Seed protocol. Every training run fixes three seeds, 0, 1, and 2, and the reported table entries are the means over those three runs. The sampler that draws the 32 tasks per iteration is seeded per run, the search index is a frozen snapshot, and the SWE-Bench sandbox returns a recorded image digest, so the environment contributes no run-to-run randomness beyond wall-clock scheduling. We use greedy decoding for all evaluation tasks, which leaves policy initialization, batch order, and rollout sampling as the sources of variance, and the spread we report is the standard deviation over the three independent runs. The final checkpoint is chosen by the training-set reward alone, and evaluation

Table 15: Training cost on one node of 8 A100 GPUs, 50 iterations, 32 tasks and 8 rollouts per task, so 12,800 trajectories per run. Wall-clock is A100-hours divided by 8.

Run	A100-h	Wall-clock	vs. GRPO control
ReAct + GRPO (32K)	610	76.3 h	1.00×
`ScrubAgent` + GRPO	640	80.0 h	1.05×
Context Folding + FoldGRPO	634	79.3 h	1.04×
`ScrubAgent` + ScrubGRPO (7B)	**680**	85.0 h	1.11×
`ScrubAgent` + ScrubGRPO (14B)	1,190	148.8 h	1.95×
Judge API spend, 7B run	$11.9	$0.24 per iteration	n/a

Table 16: Per-query inference accounting on BrowseComp-Plus. **Active** is the mean scrubbed context at a decision step, **Comp** is Eq. (18), **Prefill** is the cumulative prefill over the episode, **Model** excludes the 1.9 s per-step environment wait. Relative FLOPs are defined in the text.

Configuration	Steps	Active	Comp	Prefill	Model	End-to-end	Rel. FLOPs
Long Context ReAct† (327K)	9.8	34.6K	0%	339.1K	66.5 s	85.1 s	1.00
+ RL (GRPO)	9.3	34.6K	0%	321.8K	63.1 s	80.8 s	0.95
`ScrubAgent` (no RL)	13.4	17.3K	50%	231.8K	44.1 s	69.6 s	0.50
+ RL (GRPO)	15.3	14.5K	58%	221.9K	41.2 s	70.3 s	0.45
+ RL (ScrubGRPO)	17.4	12.1K	65%	210.5K	38.6 s	71.7 s	**0.40**
`ScrubAgent` + ScrubGRPO, SWE-Bench	85.2	21.5K	65%	1,831.8K	469.7 s	631.6 s	**0.33**
Long Context ReAct† + RL, SWE-Bench	50.8	61.4K	0%	3,119.1K	946.3 s	1,042.8 s	1.00

scores play no part in that choice. Differences are tested with a paired bootstrap over evaluation tasks with 10,000 resamples, where the pairing keeps the task set identical across the two systems being compared. The standard deviation over seeds measures training variance and the interval measures task-sampling variance, so both are needed: Table 18 gives them side by side.

We highlight four readings of Table 18. The BrowseComp-Plus gap to Context Folding with FoldGRPO is 0.5 points with an interval of $[-0.3, +1.3]$ and $p = 0.24$, so Sec. 5 describes those two systems as comparable on this benchmark, and the same comparison on the 500-task SWE-Bench set gives $+2.2$ with $[+0.7, +3.7]$ and $p = 0.004$. Against the two budget-matched variants of our own system the keyframe store costs 4.1 points and the process rewards 4.6, both with intervals well away from 0, which is the ablation evidence the claim in Sec. 5.7 rests on. The 14B backbone adds 1.6 points on BrowseComp-Plus with $p = 0.011$, a smaller increment than either component ablation. Power sets the reading of the near-tied cell: with three seeds, a pooled task count of 150, and the observed standard deviations, BrowseComp-Plus resolves differences of 1.2 points or larger at 80% power, while SWE-Bench Verified resolves 0.65 points, so a 0.5 point gap on the smaller set sits inside the resolution floor and the 2.2 point gap on the larger set does not.

I Prompts, Data Construction, and Evaluation Protocol

I.1 Benchmark Statistics

Here, we provide the benchmark statistics and the number of samples utilized in our experiments, and Table 19 collects the per-benchmark facts Sec. 5 assumes. **BrowseComp-Plus** Chen et al. (2025) comprises 680 training and 150 evaluation instances, in the form of a multi-hop question whose answer is a named entity supported by a passage in the corpus. We use the entire evaluation split comprised of 150 questions. **SWE-Bench Verified** Jimenez et al. (2024) comprises 500 verified instances, each pairing a repository issue with the test patch that decides it, and we train on 740 instances drawn from SWE-Gym Pan et al. (2025a) and SWE-Rebench Badertdinov et al. (2025). Both training sets are disjoint from both evaluation sets at the query level, and the research evaluation runs against a frozen corpus snapshot so that a retrieved passage cannot change between seeds.

Table 17: Memory and judge overhead. Embedding and scan costs are per trajectory on BrowseComp-Plus, judge costs are per 680-hour training run.

Component	Measured quantity	Share
Keyframe embedding	3.33 encoder calls per trajectory, 47 ms each	157 ms
Recall query encoding	2.30 calls per trajectory, 47 ms each	108 ms
Cosine scan over $\mathcal{K}$	$10 \times 4{,}096$ dot products, 0.03 ms	below 0.01% of a step
Encoder footprint	15.4 GB, frozen	1 of 8 GPUs
All encoder time	265 ms per trajectory	0.37% of a query
Automatic summarization	4.9 greedy calls per trajectory, 512 tokens	2.5K tokens
Info-loss judge	7,424 calls, 74.2M in and 1.34M out tokens	$11.9
Judge inference time	6.1 s per trajectory, off the critical path	0% of rollout

Table 18: Variance of the results in Table 1. Each cell reports the pass@1 mean and standard deviation over three independent training runs, and each row reports the paired bootstrap delta against `ScrubAgent` (7B) with its 95% interval. Deltas with an interval that excludes 0 are the comparisons the text states as improvements, and the one BrowseComp-Plus interval that crosses 0, against Context Folding with FoldGRPO, is stated as comparable. Every SWE-Bench difference in this table is significant at $p < 0.01$, with the one exception of the 14B row at $p = 0.34$.

	Pass@1 mean±std		vs. `ScrubAgent` (7B)		95% CI	
System	BC-Plus	SWE	Δ BC	Δ SWE	on Δ BC	p (BC)
`ScrubAgent` + ScrubGRPO (7B)	**54.8±0.6**	**53.6±0.5**				
`ScrubAgent` + ScrubGRPO (14B)	56.4±0.5	53.8±0.6	−1.6	−0.2	$[-2.8, -0.4]$	0.011
Context Folding + FoldGRPO	54.3±0.8	51.4±0.7	+0.5	+2.2	$[-0.3, +1.3]$	0.24
Long Context ReAct + RL (327K)	47.3±1.2	50.8±0.9	+7.5	+2.8	$[+6.1, +8.9]$	<0.001
Tuned hand-crafted policy	41.5±1.1	46.8±1.0	+13.3	+6.8	$[+11.7, +14.9]$	<0.001
Summary Agent	33.3±1.4	43.2±1.0	+21.5	+10.4	$[+19.6, +23.4]$	<0.001
Memory-R1 adaptation	46.2±1.0	45.1±0.8	+8.6	+8.5	$[+7.1, +10.1]$	<0.001
MEM1 adaptation	48.9±0.9	49.5±0.8	+5.9	+4.1	$[+4.4, +7.4]$	<0.001
`ScrubAgent` w/o Keyframe+Recall	50.7±0.9	50.0±0.7	+4.1	+3.6	$[+3.0, +5.2]$	<0.001
`ScrubAgent` w/o Process Rewards	50.2±1.3	50.7±1.1	+4.6	+2.9	$[+3.4, +5.8]$	<0.001

I.2 Role Prompts

Here, we provide all the prompt templates used for our experiments. The system prompt carries the role, the tool schema, and one shared memory block, and the memory block is identical across the two domains, which is the sense in which the operations are domain agnostic. The following is the prompt template for the research role. For brevity, assume one agent is in the loop:

> You are a research agent working over a fixed corpus of documents. Answer the question by searching and by opening pages. Each turn, write your reasoning, then exactly one tool call, then at most one memory operation.
> **Tools.** `search(query)` returns ranked passages with document identifiers. `open_page(doc_id)` returns the page text.
> **Stopping.** Write `answer(final_answer)` when the evidence is complete. A final answer must name the entity the question asks for and nothing else.

This is the prompt for the software engineering role:

> You are a software engineering agent working inside one repository. Each turn, write your reasoning, then exactly one tool call, then at most one memory operation.
> **Tools.** `execute_bash(command)` runs a shell command in the container. `str_replace_editor(path, old, new)` edits one file. `think(note)` records a note without touching the repository.
> **Discipline.** Reproduce the reported failure with one minimal command before editing. Re-run the failing test after each edit. A final answer is the diff you are leaving in the repository.

This is the shared memory block, appended to both roles verbatim:

Table 19: Benchmark statistics and the signal each one supplies. **acc@8** is the success rate over eight unscrubbed rollouts of the base model, used only to stratify difficulty.

Item	BrowseComp-Plus	SWE-Bench Verified
Training tasks	680	740 (SWE-Gym + SWE-Rebench)
Evaluation tasks	150	500
Tools	`search`, `open_page`	`execute_bash`, `str_replace_editor`, `think`
Step cap, train / eval	30 / 60	100 / 200
Mean steps (= tool calls), **`ScrubAgent`**	17.4	85.2
Outcome reward	answer equivalence	fail-to-pass test patch
Process reward terms	all three	all three
Difficulty strata	39 / 61 / 50	128 / 197 / 175
Observation cap per step	4,096 tokens	2,048 tokens
Verifier calls per episode	1	1

Memory. Your transcript is being rewritten as you go. Emit one JSON object in the `memory` field of your turn:

```
{"op": "focus_begin", "reason": str}
{"op": "focus_end", "summary": str}
{"op": "compress_recent", "n": int, "note": str}
{"op": "keyframe", "insight": str}
{"op": "recall", "query": str}
{"op": "none"}
```

Use `focus_begin` when the next few steps will decide the answer, so their observations stay readable. Use `focus_end` with a summary that states what the window established. Use `compress_recent` on the steps that stopped carrying information, and write the `note` as the fact you want to keep from them. Use `keyframe` the moment you learn something you will need later; the `insight` must be a sentence a different reader could act on, with the entity, the number, and where it came from. Use `recall` with a query that names the missing fact and its constraints.

The active window holds 32,768 tokens and the store holds 10 keyframes. When the window fills, the harness compresses the oldest routine steps without your note, which loses more than writing one.

The exact format in which a recalled keyframe re-enters the transcript is:

```
<recalled_keyframe step=9 score=0.71>
insight: the composer of the film is Bernard Herrmann
  snippet: [256-token extract stored at write time]
</recalled_keyframe>
```

I.3 Data Construction

The research training pool is the 680-instance training split of BrowseComp-Plus, held fixed across the three seeds, and 641 of the 680 reach a verifier-decidable answer inside the 30-step cap when the window stays unscrubbed. The engineering pool combines SWE-Gym and SWE-Rebench into 740 instances, deduplicated by repository and issue identifier against the 500 verified evaluation instances, and filtered to instances whose test patch runs in under 9 minutes, which is the container budget we hold fixed for every method. Difficulty strata in both domains come from acc@8 of the untrained backbone: at or above 87.5% an instance is easy, at 0% it is hard, and the rest are medium, so the boundaries of Fig. 3 and of Table 14 are the same boundaries used for the 39/61/50 split of Table 8. We found no overlap between the evaluation questions and the pages visited during training, checked by 13-gram containment in both directions.

I.4 Evaluation Protocol

Pass@1. For evaluation, we directly use the gold answer provided in BrowseComp-Plus and the official test patch provided in SWE-Bench Verified. One greedy episode per task is scored by the benchmark verifier and averaged over the three seeds. On the research side an answer is correct when it matches the gold entity under a normalizer that lowercases, strips punctuation and articles, and

Table 20: The five operations by the arguments they consume, with the per-trajectory firing rate measured in each domain. The **reads** column is the complete interface an operation has to the environment, and a `none` step leaves the transcript at default resolution.

Operation	Arguments	Reads	BC-Plus	SWE
`focus_begin`	reason (text)	step index	1.3	2.9
`focus_end`	summary (text)	step index	1.2	2.7
`compress_recent`	n (int), note (text)	step span, token count	3.6	8.4
`keyframe`	insight (text)	token count, step index	3.33	4.2
`recall`	query (text)	free text only	2.30	3.1
`none`	$\varnothing$	$\varnothing$	5.67	63.9
Total tool calls per episode			17.4	85.2

compares numbers after unit conversion. On the engineering side, the official test patch must turn every fail-to-pass test green with the pass-to-pass set intact. **Efficiency.** Tool calls count verifier-accepted invocations, including failed retries after one malformed call. Token counts are read from the harness log, and Comp% follows Eq. (18). **Memory statistics.** A keyframe counts as used when a `recall` returned it at least once in the episode, which is the quantity Q^{keyframe} shapes and Table 14 reports as Utilization, and unused is its complement. **Latency.** Timed on one node with the retriever resident on a separate GPU, with the environment wait reported apart in Table 16.

J Portability of the Operation Set

To test whether five manually specified operations travel to other tools and domains, we report what each operation reads and how often each one fires in the two domains. An operation reads position and count and nothing else: a step index, a token length, and a free-text field the policy writes itself. No operation signature mentions a page, a repository, a test, or a tool name.

We highlight three readings of Table 20. The two domains share one operation set, one memory block in the prompt, and one reward weighting, and both improve over their own unscrubbed control in Table 1, with $+15.5$ points on the research side and $+11.2$ on the engineering side over the 32K ReAct control trained with the plain outcome reward. These two gains are the portability evidence the manuscript carries. The firing rates in Table 20 show the same interface adapting to different content: the engineering agent issues 8.4 `compress_recent` calls per episode against 3.6 for the research agent, which the episode length accounts for, and the rate per 100 steps inverts to 9.9 against 20.7, since one page visit carries more tokens than one test exit code. The totals in the last row are the tool-call counts of Table 1, and one step yields at most one recording decision in either domain. The two domains here are both ReAct-style hosts, and the operations reach the environment through the same tool-calling schema as any other function, so a host that already emits structured tool calls can expose them without a change to its control loop. Hosts that plan in a separate module before acting remain untested.

K Limitations, Significance, and Resources

Scope of the claims. The 65% figure of the abstract and of Table 1 is a per-decision-step quantity under Eq. (18), and the cumulative prefill factor it produces is $1.53\times$ on BrowseComp-Plus and $1.70\times$ on SWE-Bench Verified, since the trained policy takes more steps than the full-context control (Appendix G). The advantage over the strongest budget-matched baseline is 0.5 points on the 150-task research set with an interval that crosses 0, and 2.2 points on the 500-task engineering set with $p = 0.004$, so Sec. 5 uses *comparable* for the first pair and *outperforms* for the second (Appendix H). The trajectory-length breakdown uses the three method-specific BrowseComp-Plus bins of Table 2, and the separate SWE-Bench Verified analysis appears in Table 13. The largest conditional BrowseComp-Plus separation occurs beyond the 30-step training cap. The adapted baselines of Appendix C keep their published memory structures and objectives while sharing our backbone, our budget, and our environments, so their scores here are not directly comparable to the

numbers in their own benchmark tables. The retrieval analysis of Appendix E measures a frozen encoder over stores of at most 10 keyframes and reports 83.5% relevance for the default retriever. The two domains tested are deep research and software engineering on Qwen2.5 backbones, and the memory block is written against a tool-calling interface. Other model families and non-tool-calling interfaces remain untested.

Statistical protocol. Each reported number is the mean of three training seeds, and the spread over those seeds is the standard deviation in Table 18. Differences are tested with a paired bootstrap over evaluation tasks with $10{,}000$ resamples, which keeps the task set fixed across the compared systems. We report the resolution floor next to every near-tied comparison, since 150 tasks resolve 1.2 points at 80% power and 500 tasks resolve 0.65 points. For all comparisons, the systems share the base model, the training pool, the verifier, the rollout count, the step cap, and the cumulative token budget, which Table 4 records. The adapted systems of Appendix C use the same test tasks.

Compute resources. We train the 7B policy using $8\times$A100 80GB GPUs, while the 14B policy is trained on the same node count under sequence parallelism. The estimated runtime required to train each model size is detailed in Table 15. Consequently, a typical run of the 7B ScrubGRPO policy takes approximately 85 hours on one node, whereas the longest run, which trains the 14B policy, exceeds six days. All experiments were conducted using NVIDIA A100 GPUs with 80GB of memory on internal servers, eight per run. Training costs 680 A100-hours for the 7B ScrubGRPO run and 1,190 for the 14B run, and the judge adds \$11.9 of API spend per 7B run. Rollout generation dominates both totals, and the memory path adds 265 ms per episode on an 8B encoder held on a separate GPU, so the addressable store costs one accelerator slot and no extra token generation at inference.

Broader impact. Teaching an agent to decide what to keep makes long-horizon autonomy cheaper to run, which lowers the barrier to the useful cases and to the harmful ones alike. Two properties of the design bound the risk we can see: every keyframe is a text record the agent wrote about its own observation, so a deployment audit can read the memory the agent is acting on, and the operation stream is logged as structured JSON at every step, which leaves an inspectable trace of what the agent chose to forget. We recommend shipping such a trace alongside the answer, and we keep the retrieval index inside the process so that a session's keyframes cannot outlive it.