Context as a Tool: Context Management for Long-Horizon SWE-Agents

Anonymous ACL submission

Abstract

Agents based on large language models have recently shown strong potential on real-world software engineering (SWE) tasks that require long-horizon interaction with repository-scale codebases. However, most existing agents rely on append-only context maintenance or passively triggered compression heuristics, which often lead to context explosion, semantic drift, and degraded reasoning in long-running interactions. We propose CAT, a new context management paradigm that elevates context maintenance to a callable tool integrated into the decision-making process of agents. CAT formalizes a structured context workspace consisting of stable task semantics, condensed long-term memory, and high-fidelity short-term interactions, and enables agents to proactively compress historical trajectories into actionable summaries at appropriate milestones. To support context management for SWE-agents, we propose a trajectory-level supervision framework, CAT-GENERATOR, based on an offline data construction pipeline that injects context-management actions into complete interaction trajectories. Using this framework, we train a context-aware model, SWE-Compressor. Experiments on SWE-Bench-Verified demonstrate that SWE-Compressor reaches a 57.6% solved rate and significantly outperforms ReAct-based agents and static compression baselines, while maintaining stable and scalable long-horizon reasoning under a bounded context budget.

1 Introduction

Large language models (LLMs) (Yang et al., 2025a; Touvron et al., 2023; Anthropic, 2025; Achiam et al., 2023) have achieved remarkable progress on tasks such as code generation and bug fixing (Chen, 2021; Austin et al., 2021; Liu et al., 2024b; Chai et al., 2024). As research attention increasingly shifts toward real-world software engineering (SWE) scenarios, maintaining stable and effective reasoning in complex and long-horizon interactive tasks has emerged as a primary challenge in agent research. Most SWE tasks (e.g., repository-level issue resolution (Jimenez et al., 2023)) often require LLMs to continuously interpret environment feedback, execute actions, and revise strategies over hundreds of interaction rounds, which poses greater demands on context management.

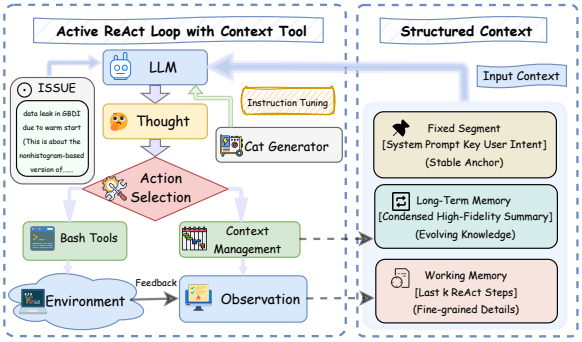


Figure 1: Overview of CAT with a structured context workspace for long-horizon reasoning.

Most existing code agents relying on paradigms such as ReAct (Yao et al., 2022), which adopts an append-only context maintenance strategy, where all past interactions are continuously concatenated into the messages. While this method may suffice for short-horizon tasks, it often leads to rapid context expansion in long-horizon scenarios, resulting in information redundancy, semantic drift, and even reasoning collapse. To mitigate these challenges, prior work (Jiang et al., 2023; Shinn et al., 2023; Wang et al., 2025b; Packer et al., 2023) has explored context compression, summarization, or multi-level memory mechanisms to constrain context size. Nevertheless, previous approaches treat context management as a passively triggered heuristic mechanism, so they lack the flexibility to adapt compression timing and content across different task phases, limiting their adaptability and scalability in complex environments. In this paper, we propose a new context management paradigm, CAT. We argue that in long-horizon interactive

tasks, context management should be internalized as a model capability rather than enforced through external constraints. Motivated by this perspective, CAT treats context management as a callable and plannable tool, on par with environment-interaction tools such as code editing and command execution, which integrates context maintenance into the action process as an active and learnable component.

In Figure 1, CAT organizes the context into a structured workspace consisting of stable task-semantic anchors, an evolvable long-term memory, and a short-term working memory. It further enables the agent to proactively trigger context folding at stage boundaries, compressing redundant histories into high-fidelity and actionable long-term memory representations. Building on this design, we introduce a trajectory-level supervision framework, CAT-GENERATOR, which injects context-management behaviors into complete interaction trajectories via offline reconstruction. Using CAT-GENERATOR, we train a context-aware model, SWE-Compressor, allowing it to learn when to compress context, how to generate effective summaries, and how to reuse compressed representations during subsequent reasoning.

Experimental results on SWE-Bench show that CAT substantially outperforms ReAct without context management and baselines with static compression strategies under the same model scale and interaction budget, demonstrating the strong context scalability in long-horizon interaction settings. The contribution of his paper is summarized as:

- We propose CAT, a new context management paradigm that internalizes context maintenance as a learnable, tool-based capability for long-horizon interactive reasoning.
- We design a structured context workspace with proactive context folding, enabling effective memory construction and sustained reasoning under bounded context budgets.
- We introduce CAT-GENERATOR, a trajectory-level supervision framework for learning context-management behaviors, and train a context-aware model, SWE-Compressor, that effectively compresses and reuses context during extended interactions.

2 SWE-Compressor

Unlike prior agents treating context management as a passive heuristic or post-processing step, CAT elevates it to a callable and plannable capability on par with environment-interaction tools, making context maintenance an active and learnable component of the agent's decision-making process. Specifically, we formalize a structured context workspace, introduce tool-based context management with segmented compression for ReAct, and present a trajectory-level supervision framework (Figure 3) with an offline data pipeline that enables effective context compression and reuse in subsequent reasoning and decision-making.

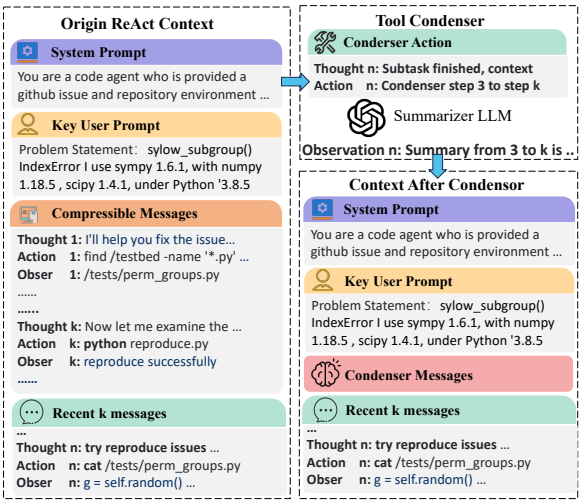


Figure 2: Example of structured context condensation in CAT during long-horizon tasks.

2.1 Structured Context Workspace

When performing long-horizon ReAct reasoning in complex environments, agent performance largely depends on how its working context is organized. Motivated by this observation, CAT models context as a controllable and dynamically updated cognitive workspace composed of three functional segments, as illustrated in Figure 2. The first is a fixed segment that preserves the system prompt and key user intent. The second is a long-term memory segment that stores a condensed, high-fidelity summary of historical trajectories. The third is a high-fidelity working memory segment that retains the most recent k ReAct interaction steps. This design provides a stable semantic anchor for the task while preserving fine-grained information from recent environment feedback, thereby supporting precise contextualized actions. Formally, the working context at step t is represented as

$$C(t) = \left(Q,\ M(t),\ I^{(k)}(t)\right) \quad (1)$$

where Q denotes the non-compressible component consisting of the system prompt and key user objec-

tives, $M(t)$ represents the high-fidelity summary of historical trajectories, and $I^{(k)}(t)$ denotes the complete records of the most recent k ReAct interactions. At initialization, $C(1) = (Q, \emptyset, \emptyset)$. As reasoning progresses, the agent continuously updates recent interactions and long-term summaries, and adjusts the content and structure of $M(t)$ through the context-management tool when needed. In this way, stable goals, condensed knowledge, and fine-grained working memory are coordinated within a unified framework, mitigating both semantic drift and uncontrolled context expansion.

2.2 Context Management as a First-Class Tool

The key innovation of CAT is to explicitly model context management as a callable tool operation and to place it at the same decision level as environment-interaction tools such as file editing or command execution. Consequently, when generating a response at step t, the agent evaluates whether to invoke the context-management tool in the same manner as it selects other external actions. In practice, the agent tends to proactively trigger context management when a subtask has been completed and requires a stage-wise summary, when the trajectory has grown sufficiently large that historical compression is necessary to maintain operational efficiency, or when subsequent reasoning benefits more from a concise, structured summary than from verbose raw logs. Through this design, context management is transformed from a post hoc procedure into a self-regulating reasoning step, becoming an integral part of the agent's strategy rather than an external constraint.

2.3 Structured Memory Generation

For the compressible historical segment, CAT applies structured summarization to condense information accumulated during long-horizon reasoning into a compact and actionable long-term memory. The goal is to reduce context length while preserving information that remains causally relevant for subsequent decisions. The resulting long-term memory summarizes key aspects of task progress, including intermediate goals, adopted strategies and their outcomes, salient environment feedback, and persistent constraints that continue to shape future reasoning. After summarization, the agent reconstructs its working context by combining the fixed task semantics, the condensed long-term memory, and the most recent high-fidelity interaction steps, enabling stable and consistent reasoning under a bounded context budget.

2.4 Supervised Trajectory Construction

Data Generation Pipeline To internalize toolized context management capability in LLM, we propose CAT-GENERATOR, a two-stage retrospective trajectory construction pipeline, as illustrated in Figure 3. CAT-GENERATOR first generates complete base ReAct reasoning trajectories without introducing any context compression operations, to maximize the naturalness and completeness of task-solving behaviors. Then, these trajectories are minimally and structurally reconstructed: context management tool invocations are injected at appropriate steps without altering the original sequence of environment interactions, yielding SFT data that are consistent with the reasoning paradigm of CAT.

Phase I: Base ReAct Trajectory Generation. Given a task instance, we first deploy a standard ReAct agent to execute a complete interaction process in a controlled environment, while explicitly disabling the context management tool and allowing only environment-related actions (e.g., code editing, command execution, or information retrieval). The resulting raw trajectory is denoted as

$$\mathcal{T}_{\text{base}} = \{(C_{\text{base}}(1), R_{\text{base}}(1)), \ldots, (C_{\text{base}}(T), R_{\text{base}}(T))\} \quad (2)$$

where $R_{\text{base}}(t)$ follows the standard ReAct response structure, consisting of Thought and Action. The objective of this stage is to preserve as much intermediate reasoning, failure patterns, and environment feedback as possible, thereby providing sufficient causal evidence for learning context compression decisions in later stages.

Phase II: Trajectory Refactoring by injecting Compression Operation. We perform an offline reconstruction of the base trajectory $\mathcal{T}_{\text{base}}$ to obtain an augmented trajectory $\mathcal{T}_{\text{retro}}$ that includes context management tool invocations. This stage is to transform context compression from an implicit side effect during generation into an explicitly modeled, controllably injected tool operation.

(1) Condensor Position Generation. We begin by conducting a structured analysis of the base trajectory to identify a set of candidate time steps suitable for inserting context management tool calls, denoted as $\mathcal{A} = \{a_1, \ldots, a_m\}$. The selection of insertion position jointly considers multiple signals, including: (i) context expansion signals, such as

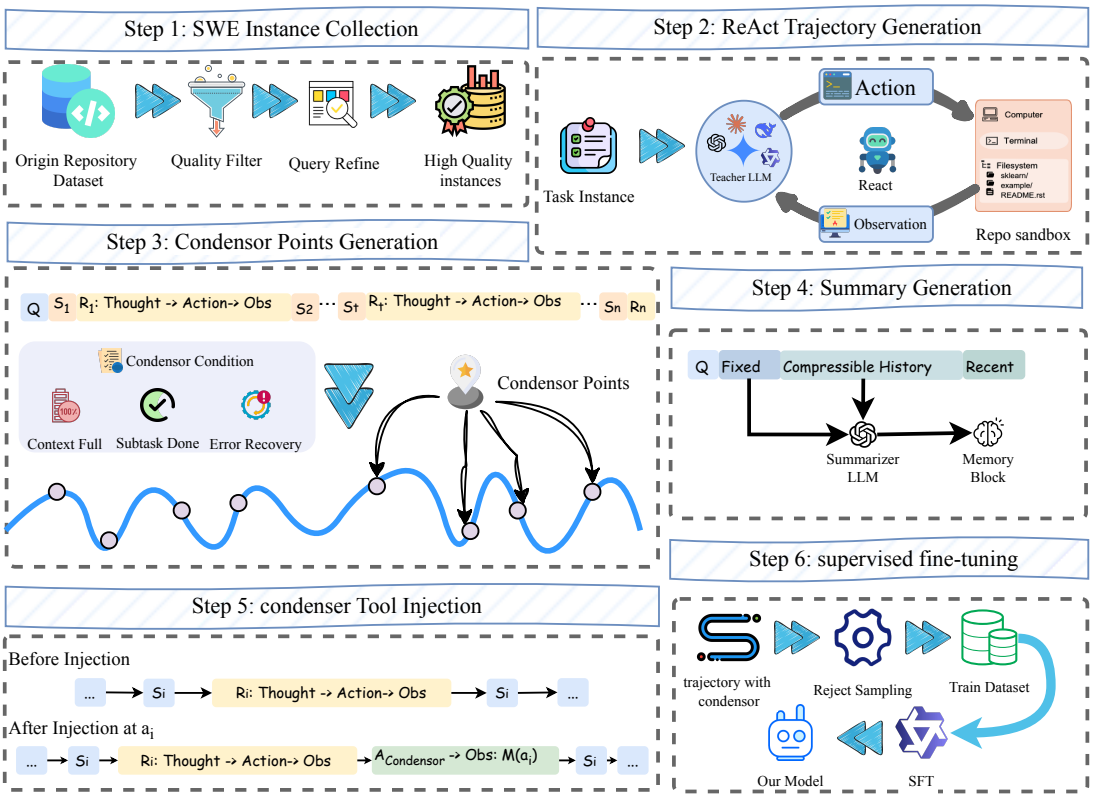


Figure 3: Overview of the data construction and training pipeline for CAT. The process includes SWE instance collection, base ReAct trajectory generation, condenser point identification, structured summary generation, tool-based context injection, and supervised fine-tuning with rejection sampling.

Statistic	CAT-Instruct
Trajectory length (steps)	
Average	87.4
Median	77.5
Max	500
Context size (tokens)	
Avg tokens per step	13,044
Median tokens per step	10,875
Max tokens per step	65,536
Context-mgmt actions / traj	
Average	4.22
Median	4.00
Max	26
Compression	
Avg tokens before	15,585
Avg tokens after	4,676
Avg ratio (%)	30

Table 1: Statistics of SFT data for CAT-Instruct.

sustained growth in context length or decreasing token utilization efficiency; (ii) structural boundary signals, such as subtask completion, strategy switching, or intermediate milestones; and (iii) error-correction signals, where new feasible directions emerge after repeated failures. We treat these signals as heuristic triggers rather than fixed thresholds, aligning insertion points with natural moments for stage-wise summarization in long-horizon reasoning and improving the learnability of compression behaviors.

(2) Segmented Context Construction and Compression Input Preparation. For each candidate insertion position a_i, we construct a segmented representation of the context at that step by partitioning the visible context into a fixed segment Q, a recent high-fidelity working memory segment $I^{(k)}(a_i)$, and a compressible historical trajectory segment. The fixed segment and the recent trajectory segment are preserved verbatim, while the historical segment is provided as input to model for long-term memory summarization. This procedure enforces the structured context workspace constraints of CAT during the offline stage.

(3) Long-Term Memory Block Generation. Based on the segmented context, we invoke a high-capacity language model to generate a structured long-term memory block $M(a_i)$. The summarizer uses the same backbone as the reasoning model (SWE-Compressor), keeping summarization aligned with the agent's internal reasoning style. This memory block aims to faithfully summarize critical information from the compressible history,

including completed subtasks, attempted strategies and their outcomes, important environment state changes, facts that continue to constrain subsequent decisions, and key information that remains useful for future steps. The generated $M(a_i)$ serves as the Observation of a context management tool invocation and is written into the long-term memory segment for subsequent reasoning.

(4) Trajectory Stitching and Minimal-Intrusion Compression Injection. Finally, we adopt a minimal-intrusion trajectory stitching strategy to inject context management behaviors into the original ReAct trajectory. Specifically, for each insertion point a_i, we explicitly insert a context management tool invocation at that step as an independent Action, whose corresponding Observation is the generated long-term memory block $M(a_i)$. $\mathcal{T} = \{(C(1), R(1)), (C(2), R(2)), \ldots, (C(T), R(T))\}$ where T denotes the total number of steps required to complete the task. Each trajectory fully captures the process from the initial task specification, through multiple rounds of environment interaction and stage-wise context compression, to final task completion. The advantage of trajectory-level supervision lies in preserving the temporal continuity of context evolution, allowing the model not only to observe the immediate summary produced by a context-management tool invocation but also to learn how that summary influences reasoning and decision-making across subsequent steps. Moreover, it enables the model to learn cross-step strategic judgments, such as how different invocation timings affect downstream efficiency and stability, thereby better reflecting the decision structure of real long-horizon tasks.

(5) Rejection Sampling Fine-Tuning. To construct high-quality trajectory-level SFT data, we apply a rejection sampling strategy during data curation to filter interaction trajectories using trajectory-level and step-level criteria. At the trajectory level, samples that fail to complete the task or enter unrecoverable error states are discarded. At the step level, we further remove trajectories exhibiting unreasonable context-management behaviors, such as excessively frequent tool invocations with minimal information gain, severe semantic drift, or internal state inconsistencies. The resulting curated set of trajectories constitutes our SFT dataset (CAT-Instruct). Table 1 summarizes key statistics of CAT-Instruct, including trajectory length, context size, and the frequency and effectiveness of context-management actions. Using CAT-Instruct for supervised fine-tuning, we obtain SWE-Compressor, which internalizes context management as a learned model capability.

3 Experiments

Datasets. We evaluate the proposed method on the SWE-Bench-Verified (Hou et al., 2024) subset. SWE-Bench is a benchmark designed to assess the ability of LLMs to solve real-world software engineering tasks collected from 12 real-world GitHub repositories. The Verified split is a high-quality subset of SWE-Bench consisting of 500 instances, which are manually curated to provide clearer problem descriptions and more reliable evaluation criteria. We report solved rate as the primary evaluation metric, defined as the proportion of instances that are successfully resolved.

Training Data Construction. For training, we collect a large number of instances from two open-source datasets, SWE-smith (Yang et al., 2025b) and SWE-ReBench (Badertdinov et al., 2025). We first employ CAT-GENERATOR to automatically generate agent interaction trajectories and apply a rejection sampling strategy to filter high-quality samples. This process yields a curated set of 20k supervised fine-tuning instances, referred to as CAT-Instruct, which effectively enhance the model's context-management capability. In addition, following the data construction protocol of SWE-smith, we collect an additional 20k high-quality supervised fine-tuning instances, denoted as BASE-INSTRUCT, which do not involve context-management skills. These data are used to train baseline models, ensuring fair and comparable evaluation against the proposed method.

Agent Post-training. We adopt Qwen2.5-Coder-32B (Hui et al., 2024) as the base model and perform post-training on the CAT-Instruct dataset to obtain the final model, SWE-Compressor. The model is trained for up to three epochs using the AdamW (Loshchilov and Hutter, 2017) optimizer with a weight decay of 0.01. We employ a cosine learning rate schedule with a warm-up ratio of 0.1 and a peak learning rate of 5×10^{-5}. During inference, we use the OpenHands (Wang et al., 2024) framework, where the agent can invoke tools including `execute_bash`, `str_replace_editor`, `submit`, and `context`. For all experiments, the temperature is fixed to 0.0. The model is trained

Method		SWE-Bench Verified	
Model	**Model Size**	**Scaffold**	**Pass@1**
ReAct Agent with 100B+ LLM			
GPT-5.1 (OpenAI, 2025)	🔒	OpenHands	76.3
GPT-4o (OpenAI, 2023)	🔒	Agentless	38.8
Claude-3.5-Sonnet (Anthropic, 2024)	🔒	OpenHands	53.0
Claude-4.5-Sonnet (Anthropic, 2025)	🔒	OpenHands	77.2
Gemini-2.5-Pro (Google Cloud, 2025)	🔒	OpenHands	59.6
Gemini-3-Pro (Google DeepMind, 2025)	🔒	OpenHands	76.2
ReAct Agent			
R2E-Gym-32B (Jain et al., 2025)	32B	OpenHands	34.4
SWE-Gym-32B (Pan et al., 2024a)	32B	OpenHands	20.6
SWE-agent-LM-32B (Yang et al., 2025b)	32B	SWE-agent	40.2
DeepSWE-32B-Preview (Luo et al., 2025)	32B	OpenHands	42.2
SWE-Mirror-LM-32B (Wang et al., 2025a)	32B	OpenHands	52.2
FrogBoss-32B (Sonwane et al., 2025)	32B	OpenHands	54.6
Seed-OSS-36B (Team, 2025)	36B	OpenHands	55.2
Llama3-SWE-RL-70B (Wei et al., 2025)	70B	OpenHands	41.0
Lingma-SWE-GPT-72B (Ma et al., 2024)	72B	SWE-SynInfer	28.8
SWE-Fixer-72B (Xie et al., 2025)	72B	SWE-Fixer	32.8
GLM-4.5-Air	12/106B	OpenHands	57.6
Qwen3-235B-A22B (Yang et al., 2025a)	22/235B	OpenHands	34.4
Qwen3-Coder-480B-A35B (Yang et al., 2025a)	35/480B	OpenHands	69.6
DeepSeek-V3.1 (Liu et al., 2024a)	37/671B	OpenHands	61.0
DeepSeek-R1-0528 (Guo et al., 2025a)	37/671B	OpenHands	45.6
Summary Agent			
ReAct Agent	32B	OpenHands	49.8
Threshold-Compression Agent	32B	OpenHands	53.8
CAT			
SWE-Compressor	32B	OpenHands	**57.6**

Table 2: Performance comparison on SWE-Bench Verified (N=500). We report Pass@1 results for different agent systems under a unified evaluation setting, grouped by model scale and agent framework.

with a context length of 65,536 tokens; for the evaluation reported in Table 2, we allow the agent to perform up to 500 interaction rounds.

Baselines. We compare the proposed method with the following baselines: (1) ReAct (Yao et al., 2022): This baseline follows the ReAct framework and does not employ any explicit context management. Once the context window is exhausted, the dialogue terminates early. (2) Threshold-Compression (OpenHands, 2025): This agent applies context compression only when the context length exceeds a predefined threshold. Upon triggering, it follows the same compression scheme as CAT: the system prompt and key user intent are preserved verbatim, together with the most recent k interaction messages, while all remaining earlier messages are summarized into a compact representation. For all baselines, we use the same base model as SWE-Compressor and use the same summarizer backbone for any compression operation. SFT is performed on the BASE-INSTRUCT dataset, which consists of 20k instances without context-management capabilities. Besides, we also compare our method with existing closed-source and open-source systems, such as GPT-5 and DeepSeek-R1.

Main Results. Table 2 presents the main experimental results, demonstrating the effectiveness of CAT. On the challenging SWE-Bench-Verified benchmark, SWE-Compressor achieves a 57.6% solved rate, reaching state-of-the-art performance under the setting of agent post-training on a 32B model. Under the same fine-tuning data budget, SWE-Compressor significantly outperforms both the ReAct Agent and the Threshold-Compression Agent baselines. Moreover, its performance is comparable to that of substantially larger models, and in some settings even surpasses them, under the same agent framework. These results indicate that CAT and CAT-GENERATOR are particularly effective for long-horizon interactive software engineering tasks such as those in SWE-Bench.

4 Further Analysis

Token Usage Analysis To evaluate the context management capability of CAT, we analyze 500 interaction trajectories from SWE-Bench. Specifically, we report the number of surviving trajectories

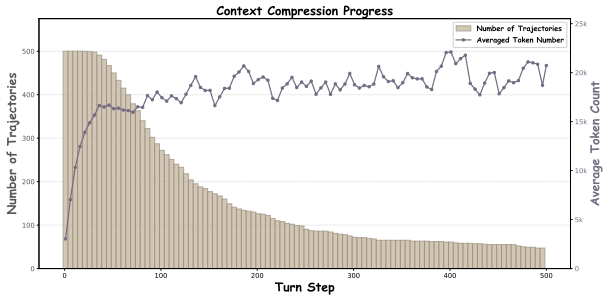


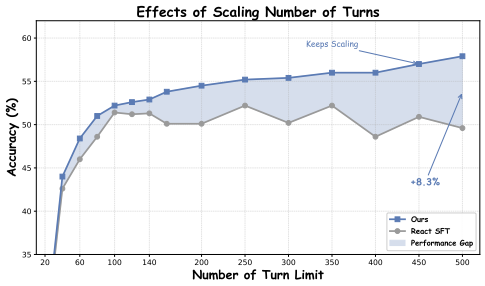
Figure 4: Context token usage and trajectory survival of CAT over interaction rounds on SWE-Bench-Verified.

at each interaction round ($|\mathcal{T}(t)|$) and the average number of context tokens over the same set of trajectories at that round ($A(t)$). The average context token count $A(t)$ is formally defined as follows:

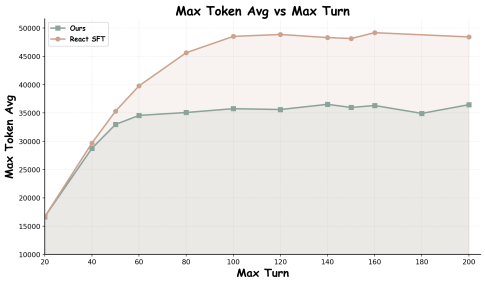
$$A(t) = \frac{1}{|\mathcal{T}(t)|} \sum_{j \in \mathcal{T}(t)} \text{TokenCount}\left(C_t^{(j)}\right) \quad (3)$$

where $\mathcal{T}(t)$ denotes the set of surviving trajectories whose lengths exceed t interaction rounds, and $C_t^{(j)}$ represents the maintained context of trajectory j at round t. In Figure 4, CAT maintains a highly compact context. As the interaction progresses, the average token count quickly stabilizes after approximately 100 rounds and remains below 32k tokens, without exhibiting continuous growth over time, indicating the effectiveness of CAT in preventing context explosion. Moreover, the trajectory survival curve indicates that, in our experiments, more than 40% of tasks remain interactive after 100 rounds. This observation further suggests that CAT equips the model with stable and extensible long-horizon interaction capabilities, highlighting its strong potential for addressing highly complex and long-running software engineering tasks.

Context Comparison between CAT and ReAct. Figure 5 evaluates CAT and ReAct on SWE-Bench under identical maximum interaction round budgets. Two methods exhibit different behaviors across varying interaction budgets. Across all comparable interaction budgets, the model equipped with CAT consistently outperforms the SFT-based ReAct baseline, indicating more efficient utilization of historical information under the same reasoning budget. As the interaction budget increases, ReAct performance saturates after around 60 rounds and subsequently degrades, primarily due to its append-only context strategy: once the context window is filled, additional interactions fail to introduce effective information and instead hinder further reasoning. In contrast, CAT continues to improve steadily with increasing interaction budgets, maintaining a clear upward trend even at 500 rounds, suggesting that CAT can continuously integrate salient information within a bounded context budget while compressing redundant history. In Figure 5, the context token usage of CAT remains stable at approximately 35k tokens, whereas the ReAct baseline rapidly exhausts the available context window.



(a) Performance of varying interaction budgets.



(b) Context token usage comparison.

Figure 5: Comparison of scalability and efficiency between CAT and ReAct on SWE-Bench. (a) CAT exhibits an upward trend in performance as the interaction budget increases to 500 rounds, whereas ReAct saturates and degrades after 60 rounds. (b) CAT maintains stable context usage (35k tokens) via condensation, while ReAct rapidly exhausts the context window.

Performance by Task Difficulty. We partition SWE-Bench into difficulty levels based on the original dataset's reported human solution time. Specifically, instances are categorized as easy ($\leq$15 minutes, 194 instances), medium (15 minutes–1 hour, 261 instances), and hard ($\geq$1 hour, 45 instances). Figure 6 presents agent performance stratified by task difficulty, comparing the scores obtained under different context management strategies, showing that CAT delivers stable and consistent performance improvements across easy, medium, and hard instances. The performance gains are substantially larger on the medium and hard subsets than on the easy subset. This observation suggests that when tasks require more complex reasoning processes

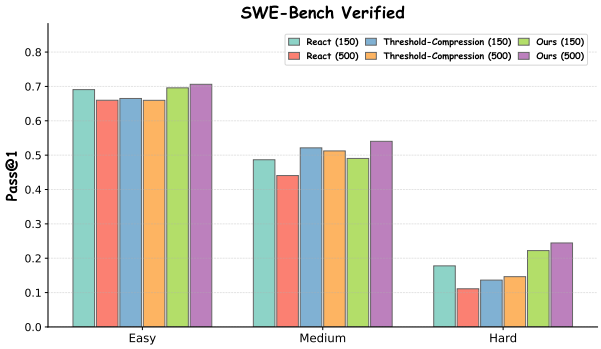


Figure 6: Performance classified by task difficulty on SWE-Bench Verified. CAT outperforms baselines, with notably larger gains on medium and hard tasks that demand complex, long-horizon reasoning.

Method	Max Steps	Tokens	Pass (%)
ReAct	150	1.96M	53.2
ReAct	500	2.54M	48.8
Threshold-Compression	150	2.49M	54.2
Threshold-Compression	500	5.18M	53.8
CAT (Base SFT)	150	1.95M	53.0
CAT (Base SFT)	500	5.07M	55.0
CAT	150	**1.89M**	54.8
CAT	500	2.75M	**57.8**

Table 3: Performance and token usage under different maximum interaction steps on SWE-Bench-Verified.

and longer-horizon context maintenance, the training signal introduced by CAT-Instruct becomes more effective. These findings further highlight the advantages of CAT-GENERATOR and CAT in addressing difficult tasks.

Effect of Interaction Budget and Token Efficiency. Table 3 summarizes the performance and token usage of different methods under varying maximum interaction budgets. When a larger number of interactions is allowed (500 steps), CAT achieves the highest pass rate among all methods, indicating stronger long-horizon reasoning capability. Under a smaller interaction budget (150 steps), CAT both maintains competitive performance and uses the fewest tokens across all methods. This favorable performance-efficiency trade-off is primarily attributed to the proposed context management mechanism, which prioritizes salient information while compressing redundant history, enabling efficient reasoning within a constrained context budget. The comparison between CAT and its Base SFT variant isolates the effect of CAT-GENERATOR. Results show that SWE-Compressor consistently outperforms Base SFT model, especially under larger interaction budgets, confirming the importance of CAT-GENERATOR.

5 Related Work

Code Agents. As LLMs plateau on standalone code generation, recent work shifts toward agentic systems for real-world software engineering, typically evaluated on SWE-bench and Multi-SWE-bench (Jimenez et al., 2023; Zan et al., 2025). Existing approaches improve performance either by enhancing agent designs with interactive tools and test-time scaling (Wang et al., 2024; Yang et al., 2024; Jain et al., 2025; Lin et al., 2025; Gao et al., 2025), or by strengthening model-level agentic capability through large executable environments, synthetic supervision, and reinforcement learning (Pan et al., 2024a; Badertdinov et al., 2025; Guo et al., 2025b; Yang et al., 2025b; Wang et al., 2025a; Sonwane et al., 2025; Wei et al., 2025; He et al., 2025; Luo et al., 2025).

Context Management. To support long-horizon decision making, prior work explores context compression and memory mechanisms, including saliency-based filtering, hierarchical summarization, and multi-level memory architectures (Li, 2023; Jiang et al., 2023; Pan et al., 2024b; Ye et al., 2025; Sun et al., 2025; Packer et al., 2023; Wang et al., 2023; Hu et al., 2025; Xiao et al., 2024). However, most rely on static compression or fixed memory policies, whereas our Tool Condensor enables dynamic, execution-driven context management that actively preserves decision-critical information over extended horizons.

6 Conclusion

In this work, we propose CAT, a context management paradigm that treats context maintenance as a first-class, toolized capability in long-horizon agents. By integrating context management into the decision process of agent, CAT enables proactive and structured condensation of interaction history, overcoming the limitations of append-only contexts and passive compression. We further introduce a trajectory-level supervision framework with an offline retrofitting pipeline to inject context-management actions into full interaction trajectories. Experiments on SWE-Bench demonstrate that CAT consistently outperforms ReAct agents and static compression baselines, while maintaining stable context usage and scalability under extended interaction budgets, underscoring the importance of modeling context evolution as an active and learnable component of agent behavior.

Limitations

Heuristic supervision for tool invocation. CAT-GENERATOR injects context-management actions by identifying condenser positions from reconstructed trajectories using heuristic signals (e.g., context growth and stage boundaries). As a result, the learned invocation behavior may not be globally optimal under different interaction styles, repositories, or context budgets, and we do not provide a principled objective for minimizing tool calls or maximizing information gain.

Summary fidelity and error accumulation. CAT relies on LLM-generated long-term memory blocks. We do not include an explicit verification mechanism to detect hallucinations, omissions, or inconsistencies in compressed memories, nor do we quantify how summary errors accumulate over hundreds of rounds. This can potentially propagate incorrect beliefs to later steps when the agent reuses compressed context.

Ethical Statement

Potential Risks. Our work strengthens long-horizon SWE agents by improving context management, which may increase the effectiveness of automated code changes. Potential risks include (i) over-reliance on agent outputs in safety- or security-critical settings, where erroneous patches or hallucinated long-term memories could cause failures; (ii) misuse to generate or scale unwanted code changes; and (iii) privacy risks when agents process repository artifacts that may contain personal identifiers. We recommend sandboxed execution, human review, and conservative deployment practices.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Anthropic. 2024. Introducing claude 3.5 sonnet. https://www.anthropic.com/news/claude-3-5-sonnet. Accessed: 2025-12-22.

Anthropic. 2025. Introducing claude sonnet 4.5. https://www.anthropic.com/news/claude-sonnet-4-5. Accessed: 2025-12-22.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

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

Linzheng Chai, Shukai Liu, Jian Yang, Yuwei Yin, Ke Jin, Jiaheng Liu, Tao Sun, Ge Zhang, Changyu Ren, Hongcheng Guo, et al. 2024. Mceval: Massively multilingual code evaluation. *arXiv preprint arXiv:2406.07436*.

Mark Chen. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Pengfei Gao, Zhao Tian, Xiangxin Meng, Xinchen Wang, Ruida Hu, Yuanan Xiao, Yizhou Liu, Zhao Zhang, Junjie Chen, Cuiyun Gao, et al. 2025. Trae agent: An llm-based agent for software engineering with test-time scaling. *arXiv preprint arXiv:2507.23370*.

Google Cloud. 2025. Gemini 2.5 pro | generative ai on vertex ai. https://docs.cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-5-pro. Accessed: 2025-12-22.

Google DeepMind. 2025. Gemini 3 pro — our most intelligent ai model. https://deepmind.google/models/gemini/pro/. Accessed: 2025-12-22.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025a. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Lianghong Guo, Yanlin Wang, Caihua Li, Pengyu Yang, Jiachi Chen, Wei Tao, Yingtian Zou, Duyu Tang, and Zibin Zheng. 2025b. Swe-factory: Your automated factory for issue resolution training data and evaluation benchmarks. *arXiv preprint arXiv:2506.10954*.

Zhenyu He, Qingping Yang, Wei Sheng, Xiaojian Zhong, Kechi Zhang, Chenxin An, Wenlei Shi, Tianle Cai, Di He, Jiaze Chen, and Jingjing Xu. 2025. Swe-swiss: A multi-task fine-tuning and rl recipe for high-performance issue resolution. https://www.notion.so/SWE-Swiss-A-Multi-Task-Fine-Tuning-and-RL-Recipe-for-High-Performance-Issue-Resolution-21e174dedd4880ea829ed4c861c44f88. Notion Blog.

Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology*, 33(8):1–79.

Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. 2025. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32779–32798.

Chengying Huan, Ziheng Meng, Yongchao Liu, Zhengyi Yang, Yun Zhu, Yue Yun, Shipeng Li, Rong Gu, Xiabao Wu, Haitao Zhang, et al. 2025. Scaling graph chain-of-thought reasoning: A multi-agent framework with efficient llm serving. *arXiv preprint arXiv:2511.01633*.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

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

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

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

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

Yucheng Li. 2023. Unlocking context constraints of llms: Enhancing context efficiency of llms with self-information-based content filtering. *arXiv preprint arXiv:2304.12102*.

Jiaye Lin, Yifu Guo, Yuzhen Han, Sen Hu, Ziyi Ni, Licheng Wang, Mingguang Chen, Hongzhang Liu, Ronghao Chen, Yangfan He, et al. 2025. Se-agent: Self-evolution trajectory optimization in multi-step reasoning with llm-based agents. *arXiv preprint arXiv:2508.02085*.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024a. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Jun Liu, Zhenglun Kong, Changdi Yang, Fan Yang, Tianqi Li, Peiyan Dong, Joannah Nanjekye, Hao Tang, Geng Yuan, Wei Niu, et al. 2025. Rcr-router: Efficient role-aware context routing for multi-agent llm systems with structured memory. *arXiv preprint arXiv:2508.04903*.

Shukai Liu, Linzheng Chai, Jian Yang, Jiajun Shi, He Zhu, Liran Wang, Ke Jin, Wei Zhang, Hualei Zhu, Shuyue Guo, et al. 2024b. Mdeval: Massively multilingual code debugging. *arXiv preprint arXiv:2411.02310*.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Michael Luo, Naman Jain, Jaskirat Singh, Sijun Tan, Ameen Patel, Qingyang Wu, Alpay Ariyak, Colin Cai, Shang Zhu Tarun Venkat, Ben Athiwaratkun, Manan Roongta, Ce Zhang, Li Erran Li, Raluca Ada Popa, Koushik Sen, and Ion Stoica. 2025. Deepswe: Training a state-of-the-art coding agent from scratch by scaling rl. https://pretty-radio-b75.notion.site/DeepSWE-Training-a-Fully-Open-sourced-State-of-the-Art-Coding-Agent-by-Scaling-RL-22281902c1468193aabbe9a8c59bbe33. Notion Blog.

Yingwei Ma, Rongyu Cao, Yongchang Cao, Yue Zhang, Jue Chen, Yibo Liu, Yuchen Liu, Binhua Li, Fei Huang, and Yongbin Li. 2024. Lingma swe-gpt: An open development-process-centric language model for automated software improvement. *arXiv preprint arXiv:2411.00622*.

OpenAI. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

OpenAI. 2025. Gpt-5.1: A smarter, more conversational chatgpt. https://openai.com/index/gpt-5-1/. Accessed: 2025-12-22.

OpenHands. 2025. Openhands context condensensation for more efficient ai agents.

Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. 2023. Memgpt: Towards llms as operating systems.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. 2024a. Training software engineering agents and verifiers with swe-gym. *arXiv preprint arXiv:2412.21139*.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, et al. 2024b. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. *arXiv preprint arXiv:2403.12968*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652.

Atharv Sonwane, Isadora White, Hyunji Lee, Matheus Pereira, Lucas Caccia, Minseon Kim, Zhengyan Shi, Chinmay Singh, Alessandro Sordoni, Marc-Alexandre Côté, et al. 2025. Bugpilot: Complex

bug generation for efficient learning of swe skills. *arXiv preprint arXiv:2510.19898*.

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

Valentin Tablan, Scott Taylor, Gabriel Hurtado, Kristoffer Bernhem, Anders Uhrenholt, Gabriele Farei, and Karo Moilanen. 2025. Smarter together: Creating agentic communities of practice through shared experiential learning. *arXiv preprint arXiv:2511.08301*.

ByteDance Seed Team. 2025. Seed-oss-36b-instruct: A 36b instruction-tuned open-source large language model. `https://huggingface.co/ByteDance-Seed/Seed-OSS-36B-Instruct`. Accessed: 2025-12-22.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Bing Wang, Xinnian Liang, Jian Yang, Hui Huang, Shuangzhi Wu, Peihao Wu, Lu Lu, Zejun Ma, and Zhoujun Li. 2023. Scm: Enhancing large language model with self-controlled memory framework. *arXiv e-prints*, pages arXiv–2304.

Junhao Wang, Daoguang Zan, Shulin Xin, Siyao Liu, Yurong Wu, and Kai Shen. 2025a. Swe-mirror: Scaling issue-resolving datasets by mirroring issues across repositories. *arXiv preprint arXiv:2509.08724*.

Qingyue Wang, Yanhe Fu, Yanan Cao, Shuai Wang, Zhiliang Tian, and Liang Ding. 2025b. Recursively summarizing enables long-term dialogue memory in large language models. *Neurocomputing*, 639:130193.

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

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

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. 2024. Infllm: Training-free long-context extrapolation for llms with an efficient context memory. *Advances in Neural Information Processing Systems*, 37:119638–119661.

Yuan-An Xiao, Pengfei Gao, Chao Peng, and Yingfei Xiong. 2025. Improving the efficiency of llm agent systems through trajectory reduction. *arXiv preprint arXiv:2509.23586*.

Chengxing Xie, Bowen Li, Chang Gao, He Du, Wai Lam, Difan Zou, and Kai Chen. 2025. Swe-fixer: Training open-source llms for effective and efficient github issue resolution. *arXiv preprint arXiv:2501.05040*.

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

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652.

John Yang, Kilian Lieret, Carlos E Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. 2025b. Swe-smith: Scaling data for software engineering agents. *arXiv preprint arXiv:2504.21798*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, et al. 2025. Agentfold: Long-horizon web agents with proactive context management. *arXiv preprint arXiv:2510.24699*.

Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Linhao Zhang, Shulin Xin, Lu Chen, Qi Liu, Xiaojian Zhong, Aoyan Li, et al. 2025. Multi-swe-bench: A multilingual benchmark for issue resolving. *arXiv preprint arXiv:2504.02605*.

A Related Work

Code Agents. As LLMs approach saturation on traditional code generation tasks, research has shifted toward their agentic capabilities in real-world codebases, typically evaluated on SWE-bench and Multi-SWE-bench (Jimenez et al., 2023; Zan et al., 2025). Existing efforts fall into two main directions. The first focuses on agent design. OpenHands and SWE-Agent (Wang et al., 2024; Yang et al., 2024) equip LLMs with interfaces such as editors and shells, enabling iterative file edits and command execution. Building on this setup, R2E-Gym, SE-Agent, and TraeAgent (Jain et al., 2025; Lin et al., 2025; Gao et al., 2025) pursue test-time scaling by increasing sampling and decision steps to approximate upper-bound performance. The second direction seeks to improve model-level agentic capability. SWE-Gym, SWE-rebench, and SWE-Factory (Pan et al., 2024a; Badertdinov et al., 2025; Guo et al., 2025b) build large executable environments for agent training, while SWE-Smith, SWE-Mirror, and BugPilot (Yang et al., 2025b; Wang et al., 2025a; Sonwane et al., 2025) generate synthetic tasks and trajectories for supervised fine-tuning. Reinforcement learning has also been explored: SWE-RL (Wei et al., 2025) uses patch similarity as rewards, and SWE-Swiss and DeepSWE (He et al., 2025; Luo et al., 2025) investigate execution-based rewards as a promising alternative.

Context Management. As agent technologies become crucial for long-horizon tasks, recent work aims to help LLMs sustain stable decision-making in complex environments. One line develops efficient context compression and memory management: methods like Selective Context, AgentFold, Context-folding, and LLMLingua (Li, 2023; Jiang et al., 2023; Pan et al., 2024b; Ye et al., 2025; Sun et al., 2025) shorten inputs via salient-information filtering or hierarchical summaries, while AgentDiet and ACON (Xiao et al., 2025; Kang et al., 2025) enhance long-trajectory fidelity through reflection, contrastive learning, or latent-space compression. Another direction builds multi-level memory systems—MemGPT, SCM, HIAGENT, and InfLLM (Packer et al., 2023; Wang et al., 2023; Hu et al., 2025; Xiao et al., 2024)—coordinating short- and long-term memory with retrieval modules for consistent extended reasoning. Multi-agent studies explore role-aware routing, shared memory, and divide-and-conquer frameworks to mitigate context explosion (Liu et al., 2025; Tablan et al., 2025; Huan et al., 2025). Yet, these approaches rely on static compression, heuristic retrieval, or fixed memory, limiting adaptability and long-term coherence. In contrast, Tool Condensor enables dynamic, execution-driven context management, flexibly removing redundancy while preserving critical information—essential for complex long-horizon tasks.

B Discussion

This subsection discusses CAT from two complementary perspectives: its generality beyond software engineering tasks and the underlying mechanisms revealed by the further analysis.

B.1 Generality beyond SWE tasks

Although this work focuses on software engineering scenarios, the core idea of CAT—explicitly modeling context management as a callable tool within the agent's action space—is not inherently specific to SWE. Instead, it targets a broader class of agentic tasks characterized by long-horizon interaction, persistent dependence on historical decisions, and continuously growing contexts. Beyond repository-level code repair, this paradigm naturally extends to tasks such as deep research, complex document analysis, long-term planning, and multi-stage decision making. In these settings, agents must maintain stable representations of key facts, intermediate conclusions, and long-term objectives across many interaction rounds. Similar to SWE, append-only context strategies or passively triggered compression heuristics are prone to semantic drift and information overload. By internalizing context management as part of the agent's decision process, CAT offers a unified and general approach to addressing these challenges.

B.2 Interpreting the further analysis results

The experiments presented in the Further Analysis section are designed to provide complementary insights into why CAT is effective, rather than serving as isolated empirical observations. First, the token usage analysis shows that CAT rapidly stabilizes context length within a bounded range and maintains this stability over hundreds of interaction rounds, demonstrating its ability to prevent uncontrolled context growth in practice. Second, the comparison with ReAct under identical interaction budgets reveals that proactive and structured context folding not only reduces redundancy but

also improves the effective utilization of historical information, leading to sustained performance gains as interaction horizons increase. Third, the difficulty-based analysis indicates that the benefits of CAT become more pronounced on medium and hard tasks, where deeper reasoning chains and longer temporal dependencies are required. Finally, the joint analysis of interaction budgets and token efficiency highlights the role of CAT-GENERATOR in teaching the model to balance when to compress context and how to preserve decision-critical information.

Taken together, these analyses support a central conclusion: the strength of CAT does not stem from compression alone, but from treating context evolution as an explicit, learnable decision process. This design enables agents to maintain accurate task understanding under bounded context budgets, providing a principled foundation for stable long-horizon reasoning.

C Checklist

Personally identifying or offensive content. Our data sources (SWE-Bench-Verified and training instances from SWE-smith and SWE-ReBench) are derived from public GitHub repositories/issues, which may contain usernames, commit metadata, or other identifiers, and may include offensive content in natural-language discussions. In this work, we do not claim a complete, formal audit for PII/offensive content beyond relying on the upstream dataset construction and filtering protocols described in the corresponding dataset papers. When releasing any additional artifacts, we recommend (and plan) to apply automated scrubbing for common PII patterns (e.g., emails, access tokens) and to avoid redistributing raw issue threads when not necessary.

D: Human subjects / annotators. We did not recruit or compensate human participants/annotators for this work. We use publicly released benchmarks/datasets; any human curation (e.g., SWE-Bench-Verified) is described in the corresponding dataset papers.

Data consent. We do not newly collect data directly from individuals. Our benchmarks and training instances are derived from publicly available software engineering artifacts (e.g., GitHub repositories and issue threads) released as part of SWE-Bench-Verified, SWE-smith, and SWE-ReBench. Accordingly, we do not obtain individual consent beyond what is implied by the original public posting and the terms of use of the underlying platforms and datasets.

Ethics review board approval. Because this work does not involve human-subject experiments or new data collection from human participants, we did not seek ethics review board (IRB) approval; the study is typically considered exempt under standard IRB criteria for research using publicly available information.

AI assistants. AI-assisted tools were used solely to polish language and improve clarity during manuscript preparation. All technical content, analyses, and conclusions were conceived, verified, and approved by the authors.