COMPACT0: LEARNING GENERALIZABLE COMPACTION INTELLIGENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon LLM agents require context management that preserves task-critical information as interaction histories grow. Existing compaction strategies face two generalization challenges: (i) harness generalization, requiring a unified, sufficiently expressive management policy across heterogeneous harnesses; and (ii) domain generalization, requiring effective context management across diverse task types. We introduce `Compact0`, a dedicated compaction intelligence model for efficient context management across various agent harnesses and task domains. Technically, `Compact0` uses a Context Abstraction Layer (CAL) that unifies native context stores through a shared management protocol, enabling harness generalization. For domain generalization, `Compact0` learns agentic context organization through CAL to retain, compress, and recover evidence and execution state via supervised fine-tuning followed by reinforcement learning guided by downstream task rewards and compaction quality rubrics. Experiments spanning NL2Repo coding, deep research, streaming QA, and Claw-style benchmarks demonstrate generalization to unseen harnesses and domains, with performance and token-efficiency gains over default harness mechanisms and specialized context-management baselines. These results support compaction intelligence as a transferable and important capability for long-horizon agents.

1 INTRODUCTION

Long-horizon LLM agents increasingly support autonomous research, software engineering, and iterative optimization (Yao et al., 2023; AlphaEvolve Team, 2025). Despite million-token context windows (OpenAI, 2026b; Anthropic, 2026), *larger windows alone do not solve long-horizon tasks*. (I) **Task contexts can exceed the window itself**: large workspace or code files already reach millions of tokens (Zhang et al., 2025), while week- or month-long workflows continually accumulate longer interaction history and often exceed 1M token limit (Akkil et al., 2026). (II) **Staying within the window does not guarantee effective reasoning**: irrelevant and redundant information causes *context rot*, degrading performance before the context limit is reached (Hong et al., 2025; Xia et al., 2026). Long-horizon agents therefore need context management that keeps the working context compact while preserving the information required for continued execution (Li et al., 2026c).

Context management broadly follows *two research lines: active and passive*, distinguished by who initiates the management operation (Li et al., 2026c; Wu et al., 2025). ❶ **Active context management** lets the execution agent decide when and how to reorganize its context, for example through proactive folding in AgentFold or explicit management and recall tools in ACM (Ye et al., 2025; Li et al., 2026c). ❷ **Passive context management** places the trigger outside the execution agent: a runtime policy invokes compression according to a context budget or a predefined schedule, as in periodic summarization and externally managed history compression (Wu et al., 2025; Kang et al., 2025). Here, "passive" describes the execution agent's role; the compression module itself can still be learned (Wu et al., 2025; Hu et al., 2026). In this work, we focus on *passive context management* because it can be integrated into existing runtimes without requiring the execution agent to learn additional management actions, making it a practical choice for serving heterogeneous agent systems (Wu et al., 2025; Nous Research, 2026a).

In mainstream harnesses, passive management commonly takes the form of a `/compact` interface with automatic triggering: OpenCode combines a generated checkpoint with recent context, while

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.

Hermes combines tool-output pruning with structured summarization (OpenCode, 2026; Nous Research, 2026a). Two generalization challenges remain. (I) **Harness generalization**: native message formats, history access, and compaction interfaces differ across harnesses, tying management logic to specific runtimes (Anomaly, 2026; Pi Contributors, 2026; OpenAI, 2026a). (II) **Domain generalization**: the information needed to resume execution varies across tasks; a strategy effective for information seeking can degrade coding performance (Li et al., 2026c; Lindenbauer et al., 2025). A reusable compaction model must therefore handle both heterogeneous interfaces and task-dependent information needs.

In the face of these two challenges, we advocate the paradigm of **compaction intelligence**: a dedicated model that provides reliable context management for different execution LLMs, task domains, and harnesses (naturally through their `/compact` interfaces). We introduce **`Compact0`** as a step toward this goal. For **harness generalization**, we design a *Context Abstraction Layer* (CAL) that unifies heterogeneous context stores under a shared management protocol; lightweight adapters handle native interfaces, allowing the same model to operate across harnesses. For **domain generalization**, we train `Compact0` to perform *agentic context organization* through interactions with CAL across diverse tasks, learning which evidence, constraints, and execution state to preserve or recover. These context-management decisions are learned end to end by the model, without relying on hand-designed decision rules. General language model alone does not inherently confer this capability (Wu et al., 2025; Hu et al., 2026), which is why we explicitly train a dedicated compaction model. How, then, should such a model be trained?

Starting from supervised fine-tuning (SFT), we train `Compact0` with *reinforcement learning with verifiable rewards* (RLVR) across diverse tasks, execution models, and harnesses. Each training episode uses CAL to compact a context, then resumes execution with the manipulated view. The optimization signal consists of two components: *(1) downstream task rewards*, which provide the primary feedback from task outcomes, and *(2) temporal rubrics*, which assess immediate information fidelity and subsequent execution utility. During the training process, only the compaction model is updated, while executors remain frozen and harnesses fixed. This shared training process aligns compaction with successful continuation across heterogeneous execution settings. In brief, our contributions are threefold:

- **Compaction Intelligence.** We advocate the paradigm of compaction intelligence, in which a dedicated model learns to provide harness- and domain-generalizable context management.
- **Compact0 Model.** We train `Compact0`, which uses a Context Abstraction Layer (CAL) to manage context across agent harnesses and learns from diverse tasks through SFT and RLVR to generalize across task domains.
- **Empirical effectiveness.** Experiments on long-horizon NL2Repo coding, deep research, streaming document QA, and OpenClaw-style tasks demonstrate performance and token-efficiency gains over default harness mechanisms and specialized context-management baselines.

2 Related Work

Long-Horizon Agent. Long-horizon agents increasingly target hundreds of interactions and hours of sustained execution across five domains (Ding et al., 2025; Li et al., 2026a; Xu et al., 2026b; Zhu et al., 2026). (I) **Repository-scale SWE** couples specification-driven construction with cross-file maintenance (Ding et al., 2025; Deng et al., 2025). (II) **Autonomous research** links hypotheses, experiments, and scientific evidence through iterative refinement (Lupidi et al., 2026; Xu et al., 2026a). (III) **Deep research** combines iterative discovery with grounded, multi-source synthesis (Xiong et al., 2026; Li et al., 2026b; Liu et al., 2026). (IV) **Workspace tasks** coordinate reasoning and updates across interdependent artifacts (Tang et al., 2026; Opsahl-Ong et al., 2026; Li et al., 2026a). (V) **Continuous optimization** pursues cumulative, correctness-preserving improvements through empirical feedback (Xu et al., 2026b; Cui et al., 2026; Zhu et al., 2026). Across these settings, context management must retain instructions, user preferences, and task-critical state while filtering obsolete or redundant information (Liu et al., 2025; Su et al., 2026; Hu et al., 2026; Li et al., 2026c). This makes a training recipe for *compaction intelligence*, aligned with the value of retained information for subsequent execution, essential to reliable long-horizon autonomy (Wu et al., 2025; Kang et al., 2025; Hu et al., 2026; Min et al., 2026).

Context Management. Context management has evolved from truncating histories toward selective retention, external storage, and adaptive access to information needed for continued exe-

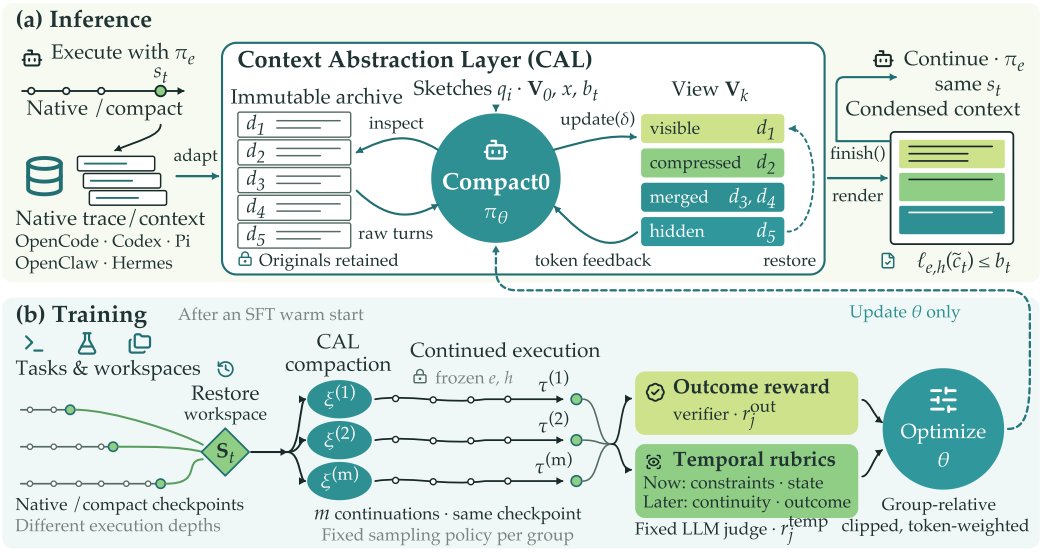


Figure 1: **Overview of `Compact0`.** (a) At a native compaction boundary, CAL maps heterogeneous harness records to a shared interface. `Compact0` inspects the preserved archive and updates turn states to produce a condensed context within budget. (b) Training restores checkpoints at diverse task depths and samples compaction episodes followed by frozen executor–harness continuations. Outcome rewards and temporal rubrics jointly optimize the compaction policy.

cution (Wu et al., 2025; Hu et al., 2026; Li et al., 2026c). **Active management** integrates context editing and consolidation into the agent's action space, allowing information selection to evolve with task reasoning at meaningful execution milestones (Liu et al., 2025; Ye et al., 2025; Li et al., 2026c). State-adaptive memory retains compact cues to archived trajectories and reconstructs relevant evidence on demand, combining active recall with externally triggered consolidation (Hu et al., 2026). Hierarchical designs separate global planning from local exploration through branches or recursive calls, exposing condensed results to the parent context while containing intermediate detail (Sun et al., 2025; Zhang et al., 2025). **Passive management** externalizes these decisions through periodic or budget-triggered summarization and observation compression, as exemplified by ReSum and ACON (Wu et al., 2025; Kang et al., 2025). Here, failure-derived guidelines, compressor distillation, and feedback from resumed execution improve what is retained without requiring the execution policy to choose when compression occurs (Kang et al., 2025; Min et al., 2026; Hu et al., 2026). Across mature harnesses, this paradigm becomes a runtime responsibility: automatic compaction combines pruning and checkpoint summaries with protected recent context under configurable budgets (OpenCode, 2026; Nous Research, 2026a). We target passive management as a reusable runtime capability, allowing compaction to improve without retraining execution agents or expanding their action spaces (Wu et al., 2025; Kang et al., 2025). However, harness-specific interfaces and domain-dependent retention policies still constrain transfer (Liu et al., 2025; Hu et al., 2026; Su et al., 2026; Li et al., 2026c), motivating CAL's common management protocol and a shared `Compact0` trained with downstream execution rewards across diverse tasks and frozen execution models. A detailed comparison of `Compact0` with existing context-management paradigms appears in Appendix F.

3 Method

Figure 1 summarizes the inference and training framework of `Compact0`. In the following subsections, we first formulate context compaction across tasks, executors, and harnesses (Section 3.1), then introduce CAL and its agentic compaction process (Section 3.2). Finally, we describe the training algorithm that produces such a transferable compaction policy, `Compact0` (Section 3.3).

3.1 Problem Formulation

For a task instance x, an execution model e with policy π_e interacts with its environment through an agent harness h, producing a trajectory as follows:

$$\tau_t = (s_0, a_0, o_1, s_1, \ldots, a_{t-1}, o_t, s_t),\ a_t \sim \pi_e(\cdot \mid x, c_t),\ (s_{t+1}, o_{t+1}) \sim p_x(\cdot \mid s_t, a_t), \quad (1)$$

where t counts interactions, s_t is the environment state, a_t an action, o_{t+1} the returned observation, c_t the model-visible working context, and p_x the task's transition–observation kernel. The harness constructs c_t from a database $\mathcal{D}_t$ of the complete observable execution record, including messages, actions, and observations; long-horizon execution makes retaining this growing record in context increasingly costly (Ding et al., 2025; Li et al., 2026a; Wu et al., 2025).

Let $\mathcal{C}$ denote a compaction mechanism and $\mathcal{C}_h$ its implementation in harness h. At a compaction boundary, it produces a condensed context view $\tilde{c}_t = \mathcal{C}_h(x, c_t, \mathcal{D}_t; b_t)$ with token budget b_t; execution then resumes from the same state s_t using this view. Implementations of $\mathcal{C}_h$ include passing the full context to an LLM for summarization (Wu et al., 2025; Nous Research, 2026a), applying heuristic deletion or learned compression to tool observations (Lindenbauer et al., 2025; Kang et al., 2025), and maintaining compact records with on-demand history retrieval (Hu et al., 2026; Li et al., 2026c). The required arguments, history-access interfaces, and output contracts vary across harnesses (Liu et al., 2025; OpenCode, 2026; Nous Research, 2026a). The objective is to preserve information useful for downstream task completion within the available context budget (Kang et al., 2025; Hu et al., 2026; Min et al., 2026):

$$\begin{aligned} \mathcal{C}^\star \in \arg\max_{\mathcal{C}} \; & \mathbb{E}_{(x,e,h)\sim\mathcal{P}} \, \mathbb{E}_{\tau\sim\mathrm{Rollout}(x,e,h;\mathcal{C}_h)} \big[\mathcal{U}(x,\tau)\big] \\ \text{subject to} \quad & \ell_{e,h}(\tilde{c}_t) \le b_t \quad \text{at every compaction,} \end{aligned} \tag{2}$$

where $\mathcal{P}$ is a distribution over tasks, execution models, and harnesses; $\mathrm{Rollout}$ generates a complete trajectory τ under the fixed execution policy and harness with $\mathcal{C}_h$ invoked at compaction boundaries; $\mathcal{U}$ measures task-completion utility; and $\ell_{e,h}$ counts tokens after harness serialization with the execution model's tokenizer.

3.2 Context Abstraction and Agentic Compaction

Realizing compaction intelligence first requires a single model to manage context across heterogeneous harnesses. This is challenging because their context structures and management interfaces differ substantially: OpenCode stores relational message/part records, whereas Pi maintains parent-linked JSONL histories (Anomaly, 2026; Pi Contributors, 2026). Compaction may append a checkpoint (`appendCompaction` in Pi) or replace the working history (`replace_compacted_history` in Codex) (Pi Contributors, 2026; OpenAI, 2026a). Fine-tuning a separate model for each native interface scales poorly as the set of harnesses grows. We introduce the Context Abstraction Layer (CAL), a virtual layer that connects a single context-management model to heterogeneous agent harnesses through a shared interface.

Context Abstraction Layer (CAL). CAL exposes execution records through a common management protocol. At compaction boundary t, the harness adapter constructs

$$(\mathcal{T}, \mathbf{V}_0) = \mathcal{A}_h(\mathcal{D}_t, c_t), \qquad \mathcal{T} = (d_i)_{i\in\mathcal{I}}, \quad \mathcal{I} = \{1, \dots, n\}, \tag{3}$$

where $\mathcal{A}_h$ imports n indexed turn records d_i, each grouping an execution interaction with its associated messages, and $\mathbf{V}_0$ records their current presentation. The archive $\mathcal{T}$ preserves original content and provenance; CAL edits only its presentation. Management and native installation then factorize as

$$\mathbf{V}^\star = \mathcal{M}_\theta(x, \mathcal{T}, \mathbf{V}_0; b_t, \mathcal{F}_{e,h}), \qquad \tilde{c}_t = \mathcal{R}_h(x, \mathcal{T}, \mathbf{V}^\star), \tag{4}$$

where $\mathcal{M}_\theta$ is the controller parameterized by θ, $\mathcal{F}_{e,h}$ its CAL tool environment, and $\mathcal{R}_h$ renders and installs the revised view. Only the adapters $\mathcal{A}_h$ and $\mathcal{R}_h$ require harness-specific implementation; the controller remains shared. To our knowledge, this contract accommodates most mainstream storage patterns (Anomaly, 2026; OpenAI, 2026a; OpenClaw Contributors, 2026b;a; Nous Research, 2026b), and extends to future runtimes with accessible records and replaceable contexts.

Agentic Context Organization (ACO). With CAL providing a shared interface, ACO formulates context reorganization as a multi-step decision process. `Compact()` realizes $\mathcal{M}_\theta$ through interaction with CAL, starting from short sketches q_i of every turn d_i and the current view $\mathbf{V}_0$, conditioned on x and b_t. View metadata persists across compactions, with new turns initialized as `visible`. At management step k, the view takes the form

$$\mathbf{V}_k = \{(g, z_g, y_g) : g \in \mathcal{G}_k\}, \qquad \biguplus_{g\in\mathcal{G}_k} g = \mathcal{I}, \tag{5}$$

where $\mathcal{G}_k$ partitions $\mathcal{I}$ into disjoint groups g, $z_g \in \{\texttt{visible}, \texttt{compressed}, \texttt{merged}, \texttt{hidden}\}$ is their members' shared state, and y_g holds summary text for compressed or merged groups, empty otherwise. A `merged` group combines multiple turns; all other groups are singletons. The model reads original turns through $\mathrm{inspect}(\mathcal{J})$ for indices $\mathcal{J} \subseteq \mathcal{I}$, without changing the view. To revise retention, $\mathrm{update}(\delta)$ atomically applies a batch edit δ that repartitions selected groups' members and assigns their states and summaries, allowing individual turns to leave existing merges; $\mathrm{finish}()$ requests completion. Decisions follow the policy

$$u_k \sim \pi_\theta(\cdot \mid \eta_k), \qquad (\mathbf{V}_{k+1}, r_k) = \mathcal{F}_{e,h}(\mathcal{T}, \mathbf{V}_k, u_k), \tag{6}$$

where π_θ is the policy of `Compact0`, u_k a tool action, η_k the initial input plus prior tool exchanges, and r_k the CAL reply; updates report rendered token usage, with x and b_t fixed throughout the call. A diagnostic hidden during exploration can thus be inspected and restored when a later subtask requires its exact contents. Because $\mathcal{T}$ is never overwritten, retention decisions remain reversible: storage is lossless even though the working view omits information. An accepted `finish` at step $k^\star$ yields $\mathbf{V}^\star = \mathbf{V}_{k^\star}$ for rendering and installation by $\mathcal{R}_h$ in Equation (4). Each `visible` group contributes its original record, each `compressed` or `merged` group contributes y_g once, and `hidden` groups contribute nothing. Groups are ordered by their last turn before native formatting and fixed system instructions are restored. CAL accepts `finish` only if the rendered context satisfies $\ell_{e,h}(\tilde{c}_t) \le b_t$ and the native message protocol, otherwise returning feedback for revision; execution then resumes from s_t under the objective in Equation (2).

3.3 Learning Transferable Compaction Intelligence

With CAL enabling context management across harnesses, we next train a dedicated model to use this interface effectively. Our training recipe for `Compact0` combines diverse context states with feedback from continued task execution.

Training across Context States. We first initialize `Compact0` with SFT on CAL compaction trajectories generated by DeepSeek-V4-Pro and selected from successful task rollouts (Appendix A details data selection). We then apply reinforcement learning across heterogeneous tasks, workspaces, and execution settings. To expose the model to different stages of task progress, we collect restorable checkpoints at native `/compact` triggers throughout execution and sample

$$(x, e, h) \sim \mathcal{P}_{\mathrm{train}}, \qquad (t, \mathbf{S}_t) \sim \mu_{x,e,h}, \qquad \mathbf{S}_t = (s_t, \mathcal{D}_t, c_t, b_t), \tag{7}$$

where $\mathcal{P}_{\mathrm{train}}$ mixes training tasks and frozen executor–harness pairs, and $\mu_{x,e,h}$ samples checkpoints across execution depths; $\mathbf{S}_t$ restores the workspace and environment state s_t together with its corresponding context records and budget. At each sampled boundary, we collect $m \ge 2$ independent rollouts. Each first invokes `Compact0` through CAL as in Section 3.2, then resumes the harness from the same workspace state with the condensed view:

$$(\tau^{(j)}, \mathcal{H}^{(j)}) \sim \mathrm{Rollout}(x, e, h; \mathcal{C}_{\theta_{\mathrm{old}},h} \mid \mathbf{S}_t), \qquad j = 1, \dots, m, \tag{8}$$

where θ_{old} is the sampling policy's parameter snapshot, $\mathcal{C}_{\theta_{\mathrm{old}},h}$ implements its CAL-based compaction, and $\mathcal{H}^{(j)}$ records all management exchanges and installed views alongside the complete execution trajectory $\tau^{(j)}$. We denote the first compaction episode in this log by $\xi^{(j)}$ and its output by $\tilde{c}_t^{(j)}$. Subsequent native triggers invoke the same policy and supply further checkpoints, exposing training to context states shaped by earlier compactions.

Learning from Continued Execution. We optimize `Compact0` with two reward sources. The first, *outcome reward*, uses the task verifier to compute $r_j^{\mathrm{out}} = \mathcal{U}(x, \tau^{(j)}) \in [0, 1]$. It anchors optimization to task completion but gives little feedback on local errors, such as the cumulative constraint loss underlying the *compaction cliff* (Zerhoudi et al., 2026). The second, *temporal rubric reward*, uses DeepSeek-V4-Pro as a fixed LLM judge to score four criteria: (i) *constraint fidelity*: preservation of instruction scope and qualifiers; (ii) *state fidelity*: accuracy of workspace facts and unresolved obligations; (iii) *dependency continuity*: timely retention or recovery of evidence needed later; and (iv) *outcome relevance*: evidence linking retention decisions to subsequent progress or failure. Given boundary evidence $\mathbf{Z}_j = (x, \mathcal{D}_t, c_t, \tilde{c}_t^{(j)}, \xi^{(j)})$, the reward is $r_j^{\mathrm{temp}} = \alpha\, \mathcal{E}_{\mathrm{now}}(\mathbf{Z}_j) + (1-\alpha)\, \mathcal{E}_{\mathrm{later}}(\mathbf{Z}_j, \tau^{(j)}, \mathcal{H}^{(j)})$, where $\mathcal{E}_{\mathrm{now}}$ and $\mathcal{E}_{\mathrm{later}}$ average scores in $[0, 1]$ for (i)–(ii) and (iii)–(iv), respectively, and $\alpha \in [0, 1]$ balances the two scopes. Retrospective scores require trace evidence

linking this compaction to later behavior. To encourage harness and domain generalization, we construct mixed RL batches:

$$\mathcal{B} = \{(x_i, e_i, h_i, \mathbf{S}_{t_i})\}_{i=1}^{B}, \qquad \left|\{\mathrm{dom}(x_i)\}_{i=1}^{B}\right| \geq 2, \qquad \left|\{h_i\}_{i=1}^{B}\right| \geq 2, \tag{9}$$

where B is the number of checkpoints and $\mathrm{dom}(x_i)$ denotes the task domain. Each entry follows Equation (7) and supplies m continuations as in Equation (8); all entries update the same compaction policy. To balance reward scales, each channel $d \in \{\mathrm{out}, \mathrm{temp}\}$ uses the group-relative advantage $\hat{v}_{ij}^{d} = (r_{ij}^{d} - \bar{r}_{i}^{d})/(\sigma_{i}^{d} + \varepsilon)$ (Shao et al., 2024), where i indexes checkpoints, j indexes their continuations, and $\bar{r}_{i}^{d}$ and σ_{i}^{d} are computed over eligible continuations of checkpoint i (reward masking in Appendix E); $\varepsilon > 0$ stabilizes normalization. Let $\mathcal{K}_{ij}$ index tokens generated by `Compact0` in $\xi_{i}^{(j)}$, with probability ratio $\omega_{ij,\ell} = \frac{\pi_\theta(y_{ij,\ell}|\eta_{ij,\ell})}{\pi_{\theta_{\mathrm{old}}}(y_{ij,\ell}|\eta_{ij,\ell})}$ for token $y_{ij,\ell}$ and its recorded prefix $\eta_{ij,\ell}$, including CAL input, tool exchanges, and preceding tokens. For each optimizer mini-batch with checkpoint indices $\mathcal{I}_{\mathrm{mb}} \subseteq \{1, \ldots, B\}$, let $N_{\mathrm{mb}} = \sum_{i \in \mathcal{I}_{\mathrm{mb}}} \sum_{j=1}^{m} |\mathcal{K}_{ij}|$. Token-level normalization gives each management token equal base weight in the objective (Yu et al., 2025):

$$\mathcal{L}_{\mathrm{mb}}(\theta) = -\frac{1}{N_{\mathrm{mb}}} \sum_{i \in \mathcal{I}_{\mathrm{mb}}} \sum_{j=1}^{m} \sum_{\ell \in \mathcal{K}_{ij}} \sum_{d} \lambda_d \min\left(\omega_{ij,\ell}\hat{v}_{ij}^{d}, \mathrm{clip}(\omega_{ij,\ell})\hat{v}_{ij}^{d}\right), \tag{10}$$

where $\lambda_{\mathrm{out}} = 1$, $\lambda_{\mathrm{temp}} = \lambda \geq 0$, and clip truncates to $[1-\epsilon_{\mathrm{low}}, 1+\epsilon_{\mathrm{high}}]$ with $0 < \epsilon_{\mathrm{low}} < \epsilon_{\mathrm{high}} < 1$. Each sampled batch is processed once in two disjoint mini-batches, with one optimizer update per mini-batch and the sampling snapshot θ_{old} fixed across both updates. Only θ is updated; executors, harnesses, and the judge remain fixed, with CAL enforcing the budget in Equation (2).

4 Experiments

4.1 Experimental Setup

Tasks and Data. Training spans three domains (coding, deep research, and Claw tasks) using TaskCraft (Shi et al., 2025), DeNovoSWE (Zhao et al., 2026), DeepSearchQA (Gupta et al., 2026), and ClawGym (Bai et al., 2026); Appendix A details the data distribution and trajectory statistics. We evaluate on Doc2Repo from BeyondSWE (Chen et al., 2026) and NL2Repo (Ding et al., 2025) (Table 1), and on BrowseComp+ (Chen et al., 2025), HotpotQA (Yang et al., 2018), TriviaQA (Joshi et al., 2017), ClawGym (Bai et al., 2026), ClawArena (Ji et al., 2026), and QwenClawBench (Qwen Team & Alibaba Data, 2026) (Table 2). Doc2Repo and NL2Repo are inherently long-horizon tasks; context management operates within each task instance, as it does for BrowseComp+. For long-context QA (HotpotQA + TriviaQA) and the three Claw benchmarks, tasks arrive sequentially within a shared session. This streaming setting models successive requests in an ongoing session. Appendix B details both protocols and baseline configurations.

Table 1: Doc2Repo and NL2Repo. Perf.: mean test pass rate (%); #Tokens: mean executor and compactor tokens (K/task); Time: mean execution time including compaction (s/task).[1]

Method	Doc2Repo			NL2Repo		
	Perf.	#Tokens	Time	Perf.	#Tokens	Time
OpenCode + DeepSeek-V4-Flash-0731						
Vanilla	55.99	4178	2481	48.29	6284	4603
Sliding Window	38.91	2411	1953	23.15	5150	3935
ReSum	45.12	3914	1871	39.83	5741	3819
CoMem	51.28	3648	3076	41.36	5417	6603
Paritok	52.08	2232	2526	46.90	3315	3089
`Compact0`	**57.59**	2699	1970	**50.21**	3928	3961
Pi + Qwen3.6-27B						
Vanilla	48.82	6101	2708	26.10	9015	3058
Sliding Window	29.20	3850	3125	12.84	5780	3692
ReSum	31.91	5481	2853	18.37	8298	3281
CoMem	33.55	3718	4199	18.30	5571	5789
Paritok	45.34	3351	2847	26.89	5034	3211
`Compact0`	**50.32**	4354	3017	**33.10**	3526	3523

Models and Training. We train Qwen3.6-35B-A3B with full-parameter SFT for two epochs at a learning rate of 10^{-5}, followed by RLVR at 10^{-6} (Section 3.3). Each RL rollout batch contains $B = 24$ checkpoints with $m = 8$ continuations each (192 total); one RL epoch performs two optimizer updates, each on $12 \times 8 = 96$ continuations. Training hyperparameters and RL rubric prompts are detailed in Appendices E and G.

[1]We follow the original benchmarks' single-run scoring protocol. Each task typically includes 20–200 verifier checks, providing fine-grained assessment within a single execution. The official evaluation likewise enables compaction but performs only a single run; thus, our evaluation is aligned with the official setup.

Table 2: Performance and efficiency across agentic benchmarks. Perf. (%) denotes accuracy for BrowseComp+/ClawArena, F1 for QA, and mean normalized rubric scores otherwise. #Tokens is mean executor input/output in thousands (K), excluding compaction; Time is mean execution time including compaction, in seconds. Both are averaged per task, or per round for ClawArena. Light/pale green indicate the best/second-best Perf. (↑) within each benchmark for a fixed executor and harness; best scores are bold, and ties share the same formatting.

Harness	Method	BrowseComp+			Long-context QA			ClawGym			ClawArena			QwenClawBench		
		Perf.	#Tokens	Time	Perf.	#Tokens	Time	Perf.	#Tokens	Time	Perf.	#Tokens	Time	Perf.	#Tokens	Time
Executor: Qwen3.6-35B-A3B																
ReAct	Vanilla	16.0	1207	642	47.45	170	19	71.86	212	101	64.69	976	94	53.55	468	197
	Recent-K	20.0	307	509	46.33	119	11	70.49	78	87	33.53	266	173	33.73	280	247
	ReSum	18.0	1177	549	50.62	43	14	71.90	155	100	56.38	381	131	55.73	527	222
	CoMem	17.0	778	714	47.91	28	18	65.26	108	392	55.49	530	286	48.40	230	636
	`Compact0`	**25.0**	988	507	**53.78**	86	25	**73.13**	94	115	**68.25**	493	176	**56.85**	333	263
NanoBot	Vanilla	**25.0**	1066	658	47.27	179	16	73.39	207	96	61.42	428	69	51.02	808	199
	Recent-K	18.0	420	630	46.55	123	12	73.31	142	132	54.60	222	251	41.04	375	227
	ReSum	18.0	975	629	49.09	47	14	74.76	202	95	57.27	342	101	53.24	543	219
	CoMem	21.0	760	802	47.35	37	18	74.08	142	251	50.15	129	273	48.55	330	524
	`Compact0`	23.0	832	706	**49.38**	99	23	**75.00**	132	104	**63.50**	335	129	**53.91**	215	258
Hermes	Vanilla	35.0	1721	616	46.12	175	17	72.90	188	77	69.44	277	47	61.83	378	124
	Recent-K	39.0	689	559	46.95	124	12	71.26	133	114	55.19	153	69	54.69	285	185
	ReSum	41.0	1097	666	**50.19**	47	14	71.78	189	89	59.64	258	70	61.98	299	140
	CoMem	36.0	867	853	46.38	37	18	76.76	118	204	54.90	110	271	52.79	241	376
	`Compact0`	**45.0**	960	646	49.79	56	20	**78.32**	171	90	**73.29**	190	87	**64.68**	287	175
Executor: DeepSeek-V4-Flash-0731																
ReAct	Vanilla	59.0	489	993	52.70	152	101	76.65	83	258	**68.25**	211	110	61.22	369	468
	Recent-K	45.0	236	1083	51.06	113	68	**77.08**	58	110	56.97	112	226	48.80	298	317
	ReSum	62.0	450	974	52.52	42	39	76.90	75	95	61.72	212	108	57.35	408	352
	CoMem	52.0	309	1435	45.24	29	57	69.33	86	712	48.37	89	275	46.00	192	1358
	`Compact0`	**65.0**	307	1159	**57.88**	76	90	76.76	53	129	65.58	147	149	**63.46**	218	357
NanoBot	Vanilla	56.0	927	985	54.24	168	104	**77.88**	172	254	69.73	276	118	62.42	476	485
	Recent-K	**63.0**	396	809	**55.84**	117	72	76.55	121	134	55.19	125	284	50.74	453	371
	ReSum	58.0	672	902	51.81	45	35	75.75	149	120	62.61	228	109	60.28	452	315
	CoMem	49.0	638	1390	45.18	43	54	63.29	153	737	54.90	156	285	55.05	407	1778
	`Compact0`	61.0	553	879	55.43	84	99	76.98	69	138	**75.37**	103	179	**63.44**	318	361
Hermes	Vanilla	51.0	608	1025	52.88	164	112	77.54	139	224	65.28	176	93	62.89	429	434
	Recent-K	54.0	386	980	51.89	117	67	76.45	124	130	**67.95**	74	67	50.17	425	394
	ReSum	**57.0**	585	1010	54.35	45	38	78.82	123	104	64.09	183	92	62.16	435	312
	CoMem	50.0	478	1527	49.14	37	57	69.53	118	594	48.37	74	194	42.33	290	1130
	`Compact0`	56.0	435	1066	**56.38**	89	106	**79.02**	95	132	64.39	141	105	**65.98**	220	474

Baselines. For coding tasks, we compare with Vanilla, Sliding Window, ReSum (Wu et al., 2025), CoMem (Zhang et al., 2026), and Paritok (Shi & Chen, 2026). Vanilla uses built-in harness compaction; ReAct uses the truncation rule in Appendix B. Sliding Window retains the latest K interaction turns and is labeled Recent-K in the agentic comparison. Baseline details are in Appendix B.

4.2 Main Results

Existing Context Management Generalizes Poorly. Table 2 shows that baseline performance varies across datasets. ReSum trails vanilla on ClawArena in all six executor–harness combinations by 1.19–9.80 points. With Qwen3.6-35B-A3B and Hermes, CoMem improves ClawGym (76.76 vs. 72.90) but degrades ClawArena (54.90 vs. 69.44). Recent-K trails vanilla on QwenClawBench in all six settings by 7.14–19.82 points. These patterns motivate assessing task utility and resource use.

Learning Transferable Compaction. `Compact0` achieves the best performance in 21 of the 30 settings in Table 2. On ClawArena, it improves over vanilla under all three Qwen harnesses by 2.08–3.85 points. With DeepSeek-V4-Flash-0731 and NanoBot, it reaches 75.37 versus vanilla's 69.73, while trailing vanilla by 2.67 and 0.89 points under ReAct and Hermes. All six ClawArena settings use fewer executor tokens but more execution time than vanilla. The same distinction appears on QwenClawBench with DeepSeek-V4-Flash-0731 and Hermes: performance rises from 62.89 to 65.98 (+3.09 points) and tokens fall from 429K to 220K, while time increases from 434 to 474 seconds. Token savings therefore require separate evaluation from latency.

Harness and Domain Generalization OpenCode and Pi are excluded from both SFT and RL, providing two entirely unseen evaluation harnesses (Appendix A). Table 1 identifies Paritok as a strong efficiency baseline: it cuts total token use by 44.2–47.2% relative to vanilla and uses the fewest tokens in three of the four settings. `Compact0` achieves the highest performance in all

Table 3: Knowledge preservation (↑) and compression rate (↓), in %. All averages five knowledge types; Comp. Rate is the post-/pre-compaction token ratio. Dark/light green mark best/second-best.

Metric	ReSum	CoMem	Paritok	`Compact0`
Constraint	52.1	65.5	69.8	**82.1**
Procedural	80.6	84.3	**88.3**	82.7
Belief	65.0	74.0	71.2	**78.0**
Preference	66.2	77.1	72.5	**80.8**
Episodic	82.0	**88.6**	84.6	87.1
All	69.2	77.9	77.3	**82.1**
Comp. Rate	24	35	39	31

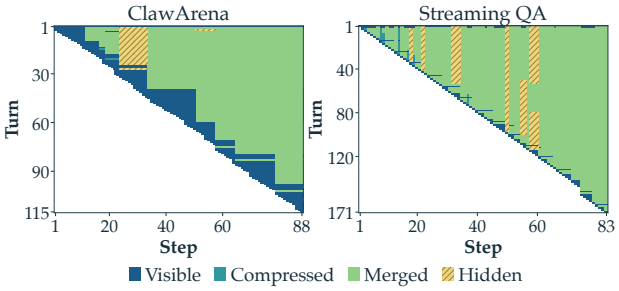


Figure 3: CAL state evolution. White precedes record creation; hatched regions denote hidden records.

four settings, exceeding the strongest baseline by 1.50–6.21 points and Paritok by 3.31–6.21 points, while reducing tokens by 28.6–60.9% relative to vanilla. On Pi/NL2Repo, it improves over Paritok from 26.89 to 33.10 while reducing tokens from 5034K to 3526K. Relative to vanilla, execution time falls by 13.9–20.6% under OpenCode and rises by 11.4–15.2% under Pi. Streaming long-context QA (HotpotQA and TriviaQA) is also unseen during training. `Compact0` improves over vanilla in all six QA executor–harness settings by 1.19–6.33 F1 points while reducing executor token use by 44.7–68.0%. These results support `Compact0`'s generalization across harnesses and domains.

4.3 Ablation Study

Figure 2 compares four variants. ❶ *Base* uses the pretrained backbone through CAL without compaction training; ❷ *SFT* learns context-management actions from demonstrations. ❸ *Outcome RL* continues from SFT using only task-outcome rewards. ❹ *Outcome RL + rubrics* adds temporal rubric rewards to obtain the full `Compact0` model. Across the three benchmarks, SFT gains 2.39–3.70 points over Base. Outcome RL adds 0.50 points on QwenClawBench and 2.70 on Doc2Repo, but reduces ClawGym performance by 0.30 points. The token profiles vary across tasks: Base uses fewer tokens than SFT and Outcome RL on ClawGym, while SFT stays close to the full model on QwenClawBench and Doc2Repo. Adding rubrics yields another 0.80–3.35 points while reducing tokens by 20.1–25.5% relative to Outcome RL. On Doc2Repo, this corresponds to 54.24 → 57.59 performance and 3415K → 2699K tokens. Rubric rewards consistently improve performance while reducing token use relative to outcome-only training.

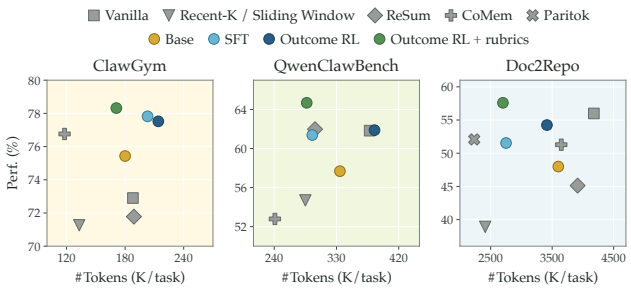


Figure 2: **Training-stage ablation.** ClawGym and QwenClawBench use Hermes + Qwen3.6-35B-A3B; Doc2Repo uses OpenCode + DeepSeek-V4-Flash-0731. Colored circles denote training variants; gray markers denote baselines. Tokens (K/task) count executor input/output in the Claw panels and executor plus compactor input/output in Doc2Repo.

4.4 Information Fidelity under Compression

To understand the information preservation underlying the observed performance and token-use gains, we adopt the analysis of Zerhoudi et al. (2026) on AgentArtifactCorpus, a collection of repository-level agent configurations. We measure preservation of (i) *Constraint*, behavioral rules and conditions; (ii) *Procedural*, instructions and execution order; (iii) *Belief*, facts and identifiers; (iv) *Preference*, soft requirements; and (v) *Episodic*, past events and observations. All averages these five metrics; definitions and evaluation details appear in Appendix C. In Table 3, `Compact0` achieves the highest All score (82.1) while retaining 31% of the original tokens, compared with CoMem's 77.9 at 35% and Paritok's 77.3 at 39%. It leads on three knowledge types, with constraint preservation of 82.1 versus the strongest baseline's 69.8, although its procedural score (82.7) trails Paritok (88.3). ReSum retains the fewest tokens (24%) but has the lowest All score (69.2). Its weaker constraint retention is consistent with rolling summarization, which reads the previous summary and new history without revisiting older raw messages (Wu et al., 2025). CAL retains

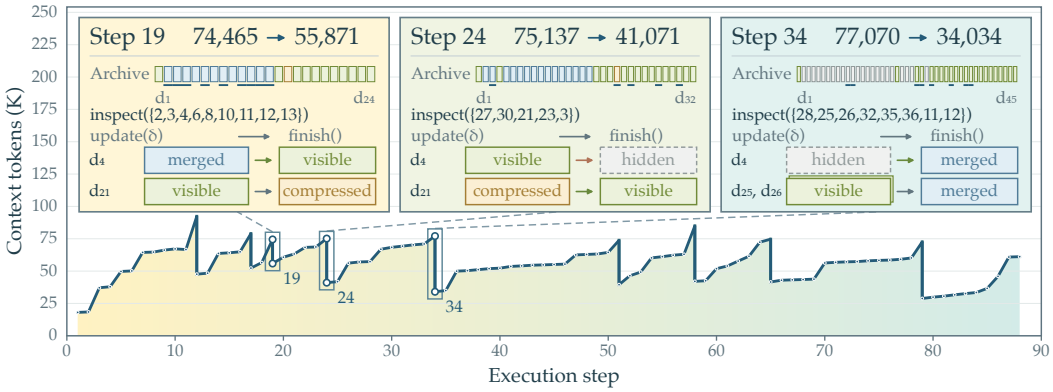


Figure 4: **CAL state dynamics.** The curve shows serialized executor-context size in one ClawArena session; vertical drops mark nine compactions. Insets expand steps 19, 24, and 34 using CAL terminology, with before/after token counts and selected record transitions.

the original archive for inspection and selective restoration. Overall, `Compact0` achieves higher information fidelity than CoMem and Paritok with a smaller working context.

4.5 Framework Analysis

CAL State Dynamics. We examine how CAL preserves access to earlier information during execution. Figure 3 tracks 115 grouped records over 88 ClawArena steps and 171 records over 83 streaming QA steps. Hidden records re-enter summaries 24 and 338 times, respectively, while compressed or merged records return verbatim 2 and 15 times. Figure 4 highlights three ClawArena compactions that reduce context size by 25.0%, 45.3%, and 55.8%. At step 19, d_4 is restored from a merged summary while d_{21} is compressed; at step 24, d_{21} returns verbatim and d_4 is hidden. At step 34, the archived original of d_4 contributes to a new merged summary. These decisions free space selectively while keeping earlier information recoverable verbatim or through new summaries.

Efficiency Analysis. Readers may wonder whether `Compact0` is costly to run, since its multi-turn compaction can be invoked repeatedly within a session. In fact, more management calls need not mean substantially longer execution: Tables 1 and 2 show competitive overall wall-clock times, and the detailed breakdown below reveals substantial gains at a modest compaction cost. Table 8 decomposes executor and compactor costs on BrowseComp+. Relative to Vanilla, `Compact0` improves performance by $10.0/5.0$ points and reduces total tokens by $23.6\%/11.5\%$ with Qwen3.6-35B-A3B/DeepSeek-V4-Flash-0731, respectively. Although compactor calls take $60/23$ seconds per task, end-to-end time increases by only $4.9\%/4.0\%$, indicating modest latency overhead, which we consider acceptable given users' focus on cost and performance.

4.6 Case Study: From Compaction Decisions to Execution

Figures 5 and 6 compare independent runs on two QwenClawBench tasks. For audio-player design, CAL-based compaction retains the completion status and pending checks: the executor validates both deliverables and finishes in 23 steps. ReSum produces the files but keeps re-reading until step 50, while Recent-K produces neither. For world-setting generation, preserving the remaining work and the shared 1M-character constraint yields all seven required artifacts in 20 steps. ReSum leaves four files missing, and Recent-K introduces a conflicting 3M value.

5 Conclusion

We presented `Compact0`, a dedicated compaction model for long-horizon agentic execution. CAL separates the preserved interaction archive from the executor-visible context, enabling selective retention, compression, and recovery through a shared interface across agent harnesses. Supervised initialization and reinforcement learning with outcome rewards and temporal rubrics train the compactor for continued execution while keeping executors fixed. Experiments show task-dependent performance–efficiency gains, while recorded trajectories demonstrate that earlier evidence can be restored or reused after compaction. This work frames compaction as a reusable capability learned from downstream execution, with selective fidelity and recoverability at its core.

AI USE STATEMENT

In this work, we used OpenAI Codex to aid and polish writing, retrieve and discover related work, support research ideation and execution, and draft sections of the paper. Specific uses included translation, refinement of methodological descriptions and mathematical notation, figure preparation, and code development. The use of LLMs for CAL demonstrations and rubric judgments is described in Section 3.3. We have reviewed all AI-assisted work and take responsibility for the final content of this work, including text, claims, code, and figures produced with the aid of generative AI.

ETHICS STATEMENT

We have not identified distinctive ethical risks arising from this work beyond those generally associated with LLM-based agents, including unreliable outputs, bias, privacy concerns, and potential misuse. These considerations remain relevant when deploying context-management models in real-world systems.

REPRODUCIBILITY STATEMENT

We commit to releasing model weights and training data/scripts upon acceptance. Nevertheless, we have carefully provided details for better reproducibility. Algorithm 1 provides pseudocode for CAL-based compaction at inference time. The context interface and training objective are specified in Sections 3.2 and 3.3. Appendix E documents experimental settings and hyperparameters. Training data sources, evaluation protocols, and analysis procedures are described in Appendices A to C, respectively.

REFERENCES

Deepak Akkil, Ravi Kokku, Karthik Vikram, Tamer Abuelsaad, Aditya Vempaty, and Satya Nitta. Emergence World: A Platform for Evaluating Long-Horizon Multi-Agent Autonomy, 2026. URL https://arxiv.org/abs/2606.08367v1.

AlphaEvolve Team. AlphaEvolve: A Gemini-powered coding agent for designing advanced algorithms, 2025. URL https://deepmind.google/blog/alphaevolve-a-gemini-powered-coding-agent-for-designing-advanced-algorithms/. Accessed September 5, 2026.

Anomaly. OpenCode: Session Storage Schema. GitHub source code and developer documentation, 2026. URL https://github.com/anomalyco/opencode/blob/7c2199d84a5830f70a8250731a42ff958145b4d6/packages/core/src/session/sql.ts. Revision 7c2199d, accessed September 6, 2026.

Anthropic. Models Overview, 2026. URL https://platform.claude.com/docs/en/models/overview. API documentation. Accessed September 5, 2026.

Fei Bai, Huatong Song, Shuang Sun, Daixuan Cheng, Yike Yang, Chuan Hao, Renyuan Li, Feng Chang, Yuan Wei, Ran Tao, Bryan Dai, Jian Yang, Wayne Xin Zhao, and Ji-Rong Wen. ClawGym: A Scalable Framework for Building Effective Claw Agents, 2026. URL https://arxiv.org/abs/2604.26904.

Guoxin Chen, Fanzhe Meng, Jiale Zhao, Minghao Li, Daixuan Cheng, Huatong Song, Jie Chen, Yuzhi Lin, Hui Chen, Xin Zhao, Ruihua Song, Chang Liu, Cheng Chen, Kai Jia, and Ji-Rong Wen. BeyondSWE: Can Current Code Agent Survive Beyond Single-Repo Bug Fixing?, 2026. URL https://arxiv.org/abs/2603.03194.

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.

Yingyun Cui, Yi Xie, Piaohong Wang, Jiawei Ma, Bo Liu, and Liangliang Cao. PERFOPT-Bench: Evaluating Coding Agents on Software Performance Optimization, 2026. URL https://arxiv.org/abs/2607.07744.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean Hendryx, Zifan Wang, Vijay Bharadwaj, Jeff Holm, Raja Aluri, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. SWE-Bench Pro: Can AI Agents Solve Long-Horizon Software Engineering Tasks?, 2025. URL https://arxiv.org/abs/2509.16941.

Jingzhe Ding, Shengda Long, Changxin Pu, Huan Zhou, Hongwan Gao, Xiang Gao, Chao He, Yue Hou, Fei Hu, Zhaojian Li, Weiran Shi, Zaiyuan Wang, Daoguang Zan, Chenchen Zhang, Xiaoxu Zhang, Qizhi Chen, Xianfu Cheng, Bo Deng, Qingshui Gu, Kai Hua, Juntao Lin, Pai Liu, Mingchen Li, Xuanguang Pan, Zifan Peng, Yujia Qin, Yong Shan, Zhewen Tan, Weihao Xie, Zihan Wang, Yishuo Yuan, Jiayu Zhang, Enduo Zhao, Yunfei Zhao, He Zhu, Liya Zhu, Chenyang Zou, Ming Ding, Jianpeng Jiao, Jiaheng Liu, Minghao Liu, Qian Liu, Chongyang Tao, Jian Yang, Tong Yang, Zhaoxiang Zhang, Xinjie Chen, Wenhao Huang, and Ge Zhang. NL2Repo-Bench: Towards Long-Horizon Repository Generation Evaluation of Coding Agents, 2025. URL https://arxiv.org/abs/2512.12730.

Nikita Gupta, Riju Chatterjee, Lukas Haas, Connie Tao, Andrew Wang, Chang Liu, Hidekazu Oiwa, Elena Gribovskaya, Jan Ackermann, John Blitzer, Sasha Goldshtein, and Dipanjan Das. DeepSearchQA: Bridging the Comprehensiveness Gap for Deep Research Agents, 2026. URL https://arxiv.org/abs/2601.20975.

Kelly Hong, Anton Troynikov, and Jeff Huber. Context Rot: How Increasing Input Tokens Impacts LLM Performance. Chroma Technical Report, 2025. URL https://www.trychroma.com/research/context-rot. Published July 14, 2025. Accessed September 5, 2026.

Yuyang Hu, Hongjin Qian, Shuting Wang, Jiongnan Liu, Ziliang Zhao, Jiejun Tan, Zheng Liu, and Zhicheng Dou. SAM: State-Adaptive Memory for Long-Horizon Reasoning Agent, 2026. URL https://arxiv.org/abs/2605.24468.

Haonian Ji, Kaiwen Xiong, Siwei Han, Peng Xia, Shi Qiu, Yiyang Zhou, Jiaqi Liu, Jinlong Li, Bingzhou Li, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. ClawArena: Benchmarking AI Agents in Evolving Information Environments, 2026. URL https://arxiv.org/abs/2604.04202.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611. Association for Computational Linguistics, 2017. doi: 10.18653/v1/P17-1147. URL https://aclanthology.org/P17-1147/.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A. Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. ACON: Optimizing Context Compression for Long-horizon LLM Agents, 2025. URL https://arxiv.org/abs/2510.00615.

Keyu Li, Junhao Shi, Yang Xiao, Mohan Jiang, Jie Sun, Yunze Wu, Dayuan Fu, Shijie Xia, Xiaojie Cai, Tianze Xu, Weiye Si, Wenjie Li, Dequan Wang, and Pengfei Liu. AgencyBench: Benchmarking the Frontiers of Autonomous Agents in 1M-Token Real-World Contexts, 2026a. URL https://arxiv.org/abs/2601.11044.

Ruizhe Li, Mingxuan Du, Benfeng Xu, Chiwei Zhu, Xiaorui Wang, and Zhendong Mao. DeepResearch Bench II: Diagnosing Deep Research Agents via Rubrics from Expert Report, 2026b. URL https://arxiv.org/abs/2601.08536.

Xiaochuan Li, Ryan Ming, Meng Chu, Shuai Shao, Rong Jin, and Chenyan Xiong. ACM: Agentic Context Management for Long Horizon Tasks, 2026c. URL https://arxiv.org/abs/2607.23809.

Tobias Lindenbauer, Igor Slinko, Ludwig Felder, Egor Bogomolov, and Yaroslav Zharov. The Complexity Trap: Simple Observation Masking Is as Efficient as LLM Summarization for Agent Context Management, 2025. URL https://arxiv.org/abs/2508.21433.

Shukai Liu, Jian Yang, Bo Jiang, Yizhi Li, Jinyang Guo, Xianglong Liu, and Bryan Dai. Context as a Tool: Context Management for Long-Horizon SWE-Agents, 2025. URL https://arxiv.org/abs/2512.22087.

Wenxuan Liu, Zixuan Li, Long Bai, Chunmao Zhang, Fenghui Zhang, Zhuo Chen, Wei Li, Yuxin Zuo, Fei Wang, Bingbing Xu, Xuhui Jiang, Jin Zhang, Xiaolong Jin, Jiafeng Guo, Tat-Seng Chua, and Xueqi Cheng. Towards Knowledgeable Deep Research: Framework and Benchmark, 2026. URL https://arxiv.org/abs/2604.07720.

Miao Lu, Weiwei Sun, Weihua Du, Zhan Ling, Xuesong Yao, Kang Liu, and Jiecao Chen. Scaling LLM Multi-turn RL with End-to-end Summarization-based Context Management, 2025. URL https://arxiv.org/abs/2510.06727.

Alisia Lupidi, Bhavul Gauri, Thomas Simon Foster, Bassel Al Omari, Despoina Magka, Alberto Pepe, Alexis Audran-Reiss, Muna Aghamelu, Nicolas Baldwin, Lucia Cipolina-Kun, Jean-Christophe Gagnon-Audet, Chee Hau Leow, Sandra Lefdal, Hossam Mossalam, Abhinav Moudgil, Saba Nazir, Emanuel Tewolde, Isabel Urrego, Jordi Armengol Estape, Amar Budhiraja, Gaurav Chaurasia, Abhishek Charnalia, Derek Dunfield, Karen Hambardzumyan, Daniel Izcovich, Martin Josifoski, Ishita Mediratta, Kelvin Niu, Parth Pathak, Michael Shvartsman, Edan Toledo, Anton Protopopov, Roberta Raileanu, Alexander Miller, Tatiana Shavrina, Jakob Foerster, and Yoram Bachrach. AIRS-Bench: a Suite of Tasks for Frontier AI Research Science Agents, 2026. URL https://arxiv.org/abs/2602.06855.

Guanghui Min, Liang Wu, Mayank Darbari, Chen Chen, and Liangjie Hong. Toward Reliable Context Compression for Long-Horizon Agents: An Empirical Study of Execution Instability, 2026. URL https://arxiv.org/abs/2608.06503.

Nous Research. Context Compression and Caching, 2026a. URL https://hermes-agent.nousresearch.com/docs/developer-guide/context-compression-and-caching/. Accessed September 5, 2026.

Nous Research. Hermes Agent: Message Storage and Archival. GitHub source code and developer documentation, 2026b. URL https://github.com/NousResearch/hermes-agent/blob/245e48008fa814b3251f50755eb656bd9fb86cb1/hermes_state_messages.py. Revision 245e480, accessed September 6, 2026.

OpenAI. Codex: Rollout Persistence and Context Compaction. GitHub source code and developer documentation, 2026a. URL https://github.com/openai/codex/tree/6af345407d9c2a568da9d01b6c4b81a9e61495c0/codex-rs. Revision 6af3454, accessed September 6, 2026.

OpenAI. Compare Models, 2026b. URL https://developers.openai.com/api/docs/models/compare. API documentation. Accessed September 5, 2026.

OpenClaw Contributors. OpenClaw: Context Engine Interface. GitHub source code and developer documentation, 2026a. URL https://github.com/openclaw/openclaw/blob/5b099995413589f41904cef1413c62239542aa7c/src/context-engine/types.ts. Revision 5b09999, accessed September 6, 2026.

OpenClaw Contributors. OpenClaw: Session Management and Compaction Internals. GitHub source code and developer documentation, 2026b. URL https://github.com/openclaw/openclaw/blob/5b099995413589f41904cef1413c62239542aa7c/docs/reference/session-management-compaction.md. Revision 5b09999, accessed September 6, 2026.

OpenCode. Compaction, 2026. URL https://opencode.ai/v2/docs/compaction. Accessed September 5, 2026.

Krista Opsahl-Ong, Arnav Singhvi, Jasmine Collins, Ivan Zhou, Cindy Wang, Ashutosh Baheti, Owen Oertell, Jacob Portes, Sam Havens, Erich Elsen, Michael Bendersky, Matei Zaharia, and Xing Chen. OfficeQA Pro: An Enterprise Benchmark for End-to-End Grounded Reasoning, 2026. URL https://arxiv.org/abs/2603.08655.

Pi Contributors. Pi: Session Manager and Compaction Entries. GitHub source code and developer documentation, 2026. URL https://github.com/earendil-works/pi/blob/9767ba275f3e9a5ee0f5c5342249b629ab1b2282/packages/coding-agent/src/core/session-manager.ts. Revision 9767ba2, accessed September 6, 2026.

Qwen Team and Alibaba Data. QwenClawBench: Real-user-distribution benchmark for OpenClaw agents, April 2026. URL https://github.com/SKYLENAGE-AI/QwenClawBench.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models, 2024. URL https://arxiv.org/abs/2402.03300.

Dingfeng Shi, Jingyi Cao, Qianben Chen, Weichen Sun, Weizhen Li, Hongxuan Lu, Fangchen Dong, Tianrui Qin, King Zhu, Minghao Liu, Jian Yang, Ge Zhang, Jiaheng Liu, Changwang Zhang, Jun Wang, Yuchen Eleanor Jiang, and Wangchunshu Zhou. TaskCraft: Automated Generation of Agentic Tasks, 2025. URL https://arxiv.org/abs/2506.10055.

Jiayu Shi and Luzhuo Chen. Paritok-4B: Intent-Conditioned Context Compression for Coding Agents, 2026. URL https://arxiv.org/abs/2608.24188.

Jin Su, Runnan Fang, Yeqiu Li, Xiaobin Wang, Shihao Cai, Pengjun Xie, Ningyu Zhang, and Fajie Yuan. U-Fold: Dynamic Intent-Aware Context Folding for User-Centric Agents, 2026. URL https://arxiv.org/abs/2601.18285.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. Scaling Long-Horizon LLM Agent via Context-Folding, 2025. URL https://arxiv.org/abs/2510.11967.

Zirui Tang, Xuanhe Zhou, Yumou Liu, Linchun Li, Yukai Wu, Weizheng Wang, Hongzhang Huang, Wei Zhou, Jun Zhou, Jiachen Song, Shaoli Yu, Jinqi Wang, Zihang Zhou, Hongyi Zhou, Yuting Lv, Jinyang Li, Jiashuo Liu, Ruoyu Chen, Chunwei Liu, GuoLiang Li, Jihua Kang, and Fan Wu. Workspace-Bench 1.0: Benchmarking AI Agents on Workspace Tasks with Large-Scale File Dependencies, 2026. URL https://arxiv.org/abs/2605.03596.

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Dingchu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, Minhao Cheng, Shuai Wang, Hong Cheng, and Jingren Zhou. ReSum: Unlocking Long-Horizon Search Intelligence via Context Summarization, 2025. URL https://arxiv.org/abs/2509.13313v3.

Shijie Xia, Yikun Wang, Zhen Huang, and Pengfei Liu. Diagnosing and Mitigating Context Rot in Long-horizon Search, 2026. URL https://arxiv.org/abs/2606.29718.

Lei Xiong, Kun Luo, Ziyi Xia, Wenbo Zhang, Jin-Ge Yao, Zheng Liu, Jingying Shao, Jianlyu Chen, Hongjin Qian, Xi Yang, Qian Yu, Hao Li, Chen Yue, Xiaan Du, Yuyang Wang, Yesheng Liu, Haiyu Xu, and Zhicheng Dou. AutoResearchBench: Benchmarking AI Agents on Complex Scientific Literature Discovery, 2026. URL https://arxiv.org/abs/2604.25256.

Wanghan Xu, Shuo Li, Tianlin Ye, Qinglong Cao, Yixin Chen, Hengjian Gao, Yiheng Wang, Qi Li, Kun Li, Sheng Xu, Shengdu Chai, Fangchen Yu, Xiangyu Zhao, Zhangrui Zhao, Weijie Ma, Zijie Guo, Koutian Wu, Haoyu Zhou, Haoxiang Yin, Lixue Cheng, Chaofan Hu, Haoxuan Li, Lu Mi, Xuxuan Xie, Yifan Zhou, Ruizhe Chen, Zhiwang Zhou, Xingjian Guo, Yuhao Zhou, Xuming He, Shengyuan Xu, Xinyu Gu, Jiamin Wu, Mianxin Liu, Chunfeng Song, Fenghua Ling, Dongzhan Zhou, Shixiang Tang, Yuqiang Li, Mao Su, Peng Ye, Siqi Sun, Bin Wang, Xue Yang, Zhenfei Yin, Tianfan Fu, Guangtao Zhai, Wanli Ouyang, Bo Zhang, Lei Bai, and Wenlong Zhang. ResearchClawBench: A Benchmark for End-to-End Autonomous Scientific Research, 2026a. URL https://arxiv.org/abs/2606.07591.

Zhangchen Xu, Junda Chen, Yue Huang, Dongfu Jiang, Jiefeng Chen, Hang Hua, Zijian Wu, Zheyuan Liu, Zexue He, Lichi Li, Shizhe Diao, Jiaxin Pei, Jinsung Yoon, Hao Zhang, Mengdi Wang, Radha Poovendran, Misha Sra, Alex Pentland, and Zichen Chen. AutoLab: Can Frontier Models Solve Long-Horizon Auto Research and Engineering Tasks?, 2026b. URL https://arxiv.org/abs/2606.05080.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380. Association for Computational Linguistics, 2018. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations*, 2023. URL https://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.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. DAPO: An Open-Source LLM Reinforcement Learning System at Scale, 2025. URL https://arxiv.org/abs/2503.14476.

Saber Zerhoudi, Jelena Mitrović, and Michael Granitzer. The Compaction Cliff in Long-Running AI Agent Memory, 2026. URL https://arxiv.org/abs/2608.22752.

Alex L. Zhang, Tim Kraska, and Omar Khattab. Recursive Language Models, 2025. URL https://arxiv.org/abs/2512.24601v3.

Yuwei Zhang, Chengyu Dong, Shuowei Jin, Changlong Yu, Hejie Cui, Hongye Jin, Xinyang Zhang, Hamed Bonab, Colin Lockard, Jianshu Chen, Zhenyu Shi, Jingbo Shang, Xian Li, and Bing Yin. CoMem: Context Management with A Decoupled Long-Context Model, 2026. URL https://arxiv.org/abs/2605.30842.

Jiale Zhao, Guoxin Chen, Fanzhe Meng, Wayne Xin Zhao, Ruihua Song, Ji-Rong Wen, and Kai Jia. DeNovoSWE: Scaling Long-Horizon Environments for Generating Entire Repositories from Scratch, 2026. URL https://arxiv.org/abs/2606.10728.

Deyao Zhu, Xin Zhou, Shengling Qin, Xuekai Zhu, Hangliang Ding, Shu Zhong, Zixin Wen, Zhonglin Xie, Chenhui Gou, Linxuan Ren, Yueyang Wang, Junfeng Zhong, Rui Liu, Tian Gao, Yangguang Lin, Jingyuan Zhang, Maojia Song, Xuan Qi, Jinhong Wu, Chenyang Zhang, Yinzhu Piao, Ziru Niu, Hongbin Lin, Lingxiang Meng, Peng Tang, Chengyao Tang, Shanyu Wu, Huanyu Zheng, Yu Liu, Liya Zhu, He Wang, Ming Ding, Ziyu Wan, Hao Liu, Sibo Wang, Haotian Zhu, Xintian Zhang, Nan Chai, Yipeng Liu, Panhao Lai, Sihang Yuan, Zixin Su, Ge Zhang, Wangchunshu Zhou, Yantao Du, Wenhao Huang, and Guang Shi. EdgeBench: Unveiling Scaling Laws of Learning from Real-World Environments, 2026. URL https://arxiv.org/abs/2607.05155.

A Training Data Distribution

Our SFT dataset contains 6,447 CAL compaction trajectories generated by DeepSeek-V4-Pro across coding, deep research, and Claw-style tasks (Table 4). We retain demonstrations from successful task rollouts. Each trajectory represents one compaction episode, and a task can contribute multiple episodes.

Table 4: SFT trajectory distribution and data sources across training domains.

Domain	Training data	Trajectories	Proportion (%)
Coding	DeNovoSWE	2,647	41.06
Deep research	TaskCraft, DeepSearchQA	2,088	32.39
Claw-style	ClawGym-SynData	1,712	26.55
Total		6,447	100.00

Training–evaluation separation. For Claw-style tasks, training uses ClawGym-SynData[2], whereas ClawGym evaluation uses the separate ClawGym-Bench[3]. Their task instances are disjoint. Across all domains, evaluation instances and their trajectories are excluded from both SFT and RL. HotpotQA and TriviaQA are reserved for evaluation, making streaming long-context QA a task type unseen during training. OpenCode and Pi are also reserved for evaluation and contribute neither SFT rollouts nor RL continuations.

Table 5 summarizes episode lengths in the collected compaction logs. A compactor decision round is one model call. Context lengths use the logged `cl100k_base` counts for the executor-visible context before and after compaction. SFT sequence lengths use the backbone tokenizer, with the training limit specified in Table 9.

Table 5: Statistics of collected compaction episodes. Context lengths count only tokens in the executor's context.

Metric	Median	Maximum
Compactor decision rounds	9	44
Session messages	37	295
Context length before compaction	74,725	914,105
Context length after compaction	32,895	494,643

B Evaluation Context Protocol

Evaluation coverage. Table 6 lists the evaluation set sizes. Single-instance benchmarks comprise Doc2Repo (50 tasks from BeyondSWE (Chen et al., 2026)), NL2Repo (Ding et al., 2025), and BrowseComp+ (Chen et al., 2025). Streaming QA combines HotpotQA (Yang et al., 2018) and TriviaQA (Joshi et al., 2017); the remaining streaming benchmarks are ClawGym (Bai et al., 2026), ClawArena (Ji et al., 2026), and QwenClawBench (Qwen Team & Alibaba Data, 2026). ClawArena maintains a separate continuous context for each of its 12 scenes.

We distinguish context management within a task from management across a stream of tasks. Let x_i denote task instance i and T_i its final interaction step. We write $M_{i,t} = (\mathcal{D}_{i,t}, c_{i,t})$ for the context state at step t, comprising the stored interaction record and the current executor-visible view. Let M_{init} be the harness's initial context state, and let $\text{Append}(M, x)$ add a new task request to the record and working context. During execution, the harness appends actions and observations, and the selected management method updates the working view at its compaction boundaries.

Single-instance evaluation. For Doc2Repo, NL2Repo, and BrowseComp+, each task starts a separate session:

$$M_{i,0} = \text{Append}(M_{\text{init}}, x_i). \tag{11}$$

[2] https://github.com/ClawGym/ClawGym-SynData
[3] https://github.com/ClawGym/ClawGym-Bench

Table 6: Evaluation set sizes and context protocols. ClawArena counts task rounds within persistent scenes; the other sizes count task instances.

Benchmark	Evaluation size	Context protocol
Doc2Repo (BeyondSWE)	50 tasks	Single instance
NL2Repo	104 tasks	Single instance
BrowseComp+	100 tasks	Single instance
Streaming QA	149 questions	Streaming
ClawGym	200 tasks	Streaming
ClawArena	337 rounds / 12 scenes	Streaming per scene
QwenClawBench	100 tasks	Streaming

Doc2Repo and NL2Repo require long-horizon repository work, while BrowseComp+ requires extended information seeking. Context management therefore acts on the history accumulated within each instance, and the context is reset before the next instance.

Streaming evaluation. The streaming protocol for HotpotQA, TriviaQA, ClawGym, ClawArena, and QwenClawBench uses an ordered sequence $\mathcal{X} = (x_1, \ldots, x_N)$ processed in one session:

$$M_{1,0} = \mathrm{Append}(M_{\mathrm{init}}, x_1), \qquad M_{i,0} = \mathrm{Append}(M_{i-1,T_{i-1}}, x_i), \quad i > 1. \tag{12}$$

The next request arrives after the preceding task completes. The context record and its managed view persist across task boundaries, including any summaries or retained history, so context management operates over the growing stream. HotpotQA and TriviaQA constitute the long-context QA group in Table 2. This protocol models an ongoing user session containing successive, potentially unrelated requests, with task completion evaluated using each benchmark's scoring rule.

Vanilla context management. Vanilla retains the native compaction routines of OpenCode, Pi, NanoBot, and Hermes. OpenCode summarizes older history near its usable input limit while retaining recent turns; Pi maintains a rolling summary with roughly 20,000 tokens of recent history. NanoBot consolidates accepted history into a checkpoint while preserving newly added messages. Hermes processes older tool outputs and summarizes middle history while preserving system instructions and recent context. These summaries use the executor model. For ReAct, context reaching 65,536 tokens (64K) triggers deterministic truncation: the earliest agent turns are discarded until the context fits within 32,768 tokens (32K).

Comparison protocol. Table 7 documents the activation rules, budgets, and call limits used in evaluation. Matching every trigger and token limit would simplify experimental control, but these choices are part of the methods themselves: Paritok is trained for intent-conditioned segment compression, including tool outputs, while CoMem couples its memory model to an asynchronous update schedule (Shi & Chen, 2026; Zhang et al., 2026). Replacing these mechanisms with a uniform schedule, or disabling their specialized optimizations, could disadvantage a baseline by changing the operating regime it was designed and trained for. We therefore retain the methods' native management logic and make our concrete settings explicit. Our aim is to give each baseline a faithful evaluation as a complete context-management system, comparing task performance and end-to-end time while accounting for executor and compactor token use separately in Table 8.

An agent turn comprises an assistant action and its associated tool results. Recent-K's QA fallback reduces the retained window from two turns to one when needed to fit the context. Paritok uses distinct tool-output and history-compaction paths, retaining its specialized block processing. For `Compact0`, 48,000 tokens is a soft target for selectable older history; protected recent turns and key inputs contribute separately to the final executor context. The decision-round limit does not include nested fragment-summary calls, whose token and time costs are included in compactor usage.

ReSum uses rolling summarization through CAL with Qwen3.6-35B-A3B (since its official checkpoint is unavailable), a 65,536-token trigger, and a 50-step limit shared with Vanilla.

CoMem uses `summary-sft-qwen3-4b` with $k = 2$ and asynchronous summary generation, preserving overlap with executor inference and activation at the original KSO boundary. Its input truncation and output limits are specified in Table 7. End-to-end measurements retain both the cost of management and any latency benefit from overlap. This protocol evaluates the performance–

Table 7: Context-management configurations. Token quantities specify input limits, output limits, or soft targets as indicated; they are not a shared post-compaction context cap.

Method	Activation / selection	Budget	Compaction calls
Vanilla	Native harness compaction; ReAct truncates at 65,536 tokens.	Harness-specific; ReAct retains $\leq 32{,}768$ tokens.	Native scheduling; ReAct uses truncation.
Recent-K	Keep the latest 5 agent turns per request; QA uses $2 \rightarrow 1$ on overflow, without pinning the first question.	Turn-based; no token target.	None.
ReSum	Context $\geq 65{,}536$ tokens.	Summary output $\leq 4{,}096$ tokens.	One summary per trigger; up to 3 attempts.
CoMem	Asynchronous update every 2 turns; no token threshold.	Input truncated to 95,000 tokens; summary output $\leq 2{,}048$.	One summary per update; up to 3 attempts.
Paritok	Check tool outputs per request; compress 512–50,000-token blocks, passing larger blocks through. History path activates above 65,536 tokens.	History chunks of about 3,000 tokens; output $\leq 3{,}000$ tokens per call.	One call per selected block; up to 3 retries.
`Compact0`	Context $\geq 65{,}536$ tokens; protect the latest 5 turns and key inputs.	Soft target of 48,000 tokens for older history; protected content counted separately.	Up to 40 decision rounds per trigger; fragment summaries additional; up to 3 attempts per call.

resource trade-off of each complete method; it does not isolate compressor quality under an identical budget.

C Analysis Protocol

Sampling and annotation. The fidelity study uses a 20-document pilot followed by 100 evaluation configurations, deduplicated by content and separated by repository from development data. Documents are sampled across length and configuration families, then split into records preserving headings, code blocks, and scope. All methods receive the same ordered source records, including headings and code blocks, in the compressible context. Preservation is scored against the original source in the model-visible compressed output; the presence of a fact in the archive alone earns no credit. Recovery behavior is analyzed separately through CAL state transitions. Gold annotations identify atomic information, type, conditions, and source evidence independently of model outputs. Output judgments require textual evidence and distinguish equivalent preservation, weakening, contradiction, and omission; negation, scope, and exact parameters are assessed explicitly.

Knowledge types. A constraint specifies a required or prohibited behavior; preserving it requires retaining its negation, triggering conditions, exceptions, and scope. A procedure specifies an executable sequence, including prerequisites, ordering, commands, and parameters. A belief is a factual assertion about the task or environment, including names, paths, and values whose exact identity matters. A preference expresses a soft priority or stylistic choice that permits alternatives. An episodic item records a past action or observation, including its outcome and temporal context. These definitions follow Zerhoudi et al. (2026); annotations split mixed statements into atomic items before assigning types.

Knowledge preservation. We use the five knowledge types of Zerhoudi et al. (2026): constraints (C), procedures (P), beliefs (B), preferences (F), and episodic knowledge (E). For document d, let $A_{d,T}$ be its annotated atoms of type T. With $D_T = \{d : |A_{d,T}| > 0\}$, the preservation rate of

method m in Table 3 is

$$\mathrm{Pres}_T(m) = \frac{100}{|D_T|} \sum_{d \in D_T} \frac{\sum_{a \in A_{d,T}} \mathbb{1}[a \text{ is equivalently preserved by } m]}{|A_{d,T}|}. \tag{13}$$

Following the unweighted aggregation in Zerhoudi et al. (2026), $\mathrm{All}(m) = \frac{1}{5} \sum_{T \in \{C,P,B,F,E\}} \mathrm{Pres}_T(m)$. A missing output receives zero preservation.

Recorded trajectory. Figure 4 uses the persistent ClawArena scene `hil_s1`. Context size follows the logged `cl100k_base_json_estimate` for executor messages and tool definitions, with pre- and post-compaction values placed at the same request. Native read, window-editing, and delivery operations are mapped to CAL's `inspect`, `update`, and `finish` vocabulary. Record states follow the accepted window and explicit summary-source membership; the insets show selected transitions. The step-34 recovery of d_4 uses its archived original in a 12-record summary, rather than restoring it verbatim.

State-history reconstruction. Figure 3 showcases two continuous sessions: ClawArena with DeepSeek-V4-Flash/NanoBot (88 requests, 115 grouped records, nine compactions) and streaming QA with Qwen3.6-35B-A3B/NanoBot. Each column records the installed state after a request and any committed compaction; rows retain archive order, grouping assistant turns with their tool results and retaining standalone user records. Original records present in the window are visible; sources explicitly linked to single- or multi-record summaries are compressed or merged; other existing records are hidden. Original presence takes precedence when representations overlap, and source-free notes receive no inferred provenance. White cells precede record creation. Recovery counts are record-level transitions: ClawArena has 24 hidden-to-merged events; QA has 334 hidden-to-merged and four hidden-to-compressed events. The two and 15 verbatim restorations originate from compressed or merged states. Repeated recovery of the same record counts separately; these counts characterize the selected sessions.

Efficiency accounting. Table 8 reports the Hermes runs on BrowseComp+ for both executors. Executor and compactor columns separately record input/output tokens and model-call time; total tokens sum both components. End-to-end task time is measured independently and includes tool execution and orchestration, with asynchronous model calls potentially overlapping, so it is not the sum of the two model-time columns. The executor-token, performance, and task-time entries match Table 2; compactor usage is additionally included here to expose the full token cost.

Table 8: Executor and compactor costs on BrowseComp+ with Hermes. Tokens are in thousands (K/task); times are in seconds per task. Executor and compactor times measure model calls, whereas task time measures end-to-end execution.

Method	Perf. (%)	Executor		Compactor		Total tokens	Task time
		Tokens	Time	Tokens	Time		
Executor: Qwen3.6-35B-A3B							
Vanilla	35.0	1721	397	0	0	1721	616
Recent-K	39.0	689	234	0	0	689	559
ReSum	41.0	1097	327	60	15	1157	666
CoMem	36.0	867	292	120	24	987	853
`Compact0`	45.0	960	243	355	60	1315	646
Executor: DeepSeek-V4-Flash-0731							
Vanilla	51.0	608	764	0	0	608	1025
Recent-K	54.0	386	713	0	0	386	980
ReSum	57.0	585	837	36	5	621	1010
CoMem	50.0	478	1268	75	18	553	1527
`Compact0`	56.0	435	660	103	23	538	1066

Ablation controls and accounting. Within each panel, the four training variants share test tasks, executor, harness, trigger and target budgets, management-call limits, and termination rules. Baseline points use the configurations in the corresponding main table. Token accounting in Figure 2 follows the corresponding main table: executor input/output for ClawGym and QwenClawBench,

and executor plus compactor input/output for Doc2Repo, including sketches, inspection follow-ups, and retries. Execution wall time includes context management with each method's scheduling policy; evaluation judging and the extra rubric-training cost are recorded separately. Variants run interleaved at matched concurrency.

D CASE STUDIES

Preserving progress for termination. In Figure 5, the retained status ledger identifies both completed outputs and the remaining checks. The executor validates and finishes in 23 steps (85.78). ReSum produces both files but continues re-reading through step 50; Recent-K repeatedly rediscovers requirements and writes neither file.

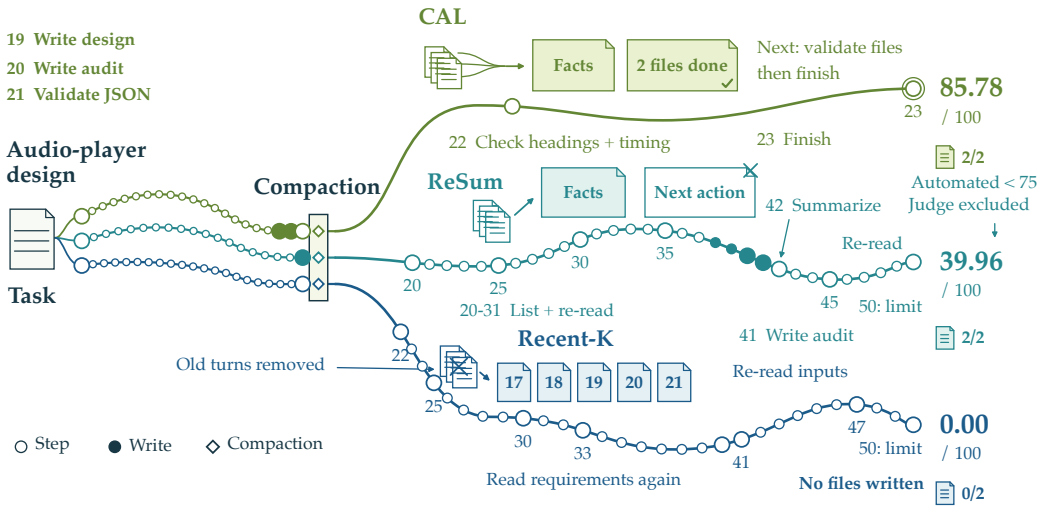


Figure 5: **Audio-player design.** Retaining completed work and the next action supports validation and termination. Branches show independent runs; scoring and step conventions are detailed in Appendix C.

Preserving constraints across artifacts. In Figure 6, the retained checklist and 1M-character plan guide completion of the remaining three files, yielding all seven required artifacts in 20 steps (98.45). ReSum leaves four files missing. Recent-K removes the master-document turn from context before step 50, then writes a conflicting 3M value; the original file remains on disk.

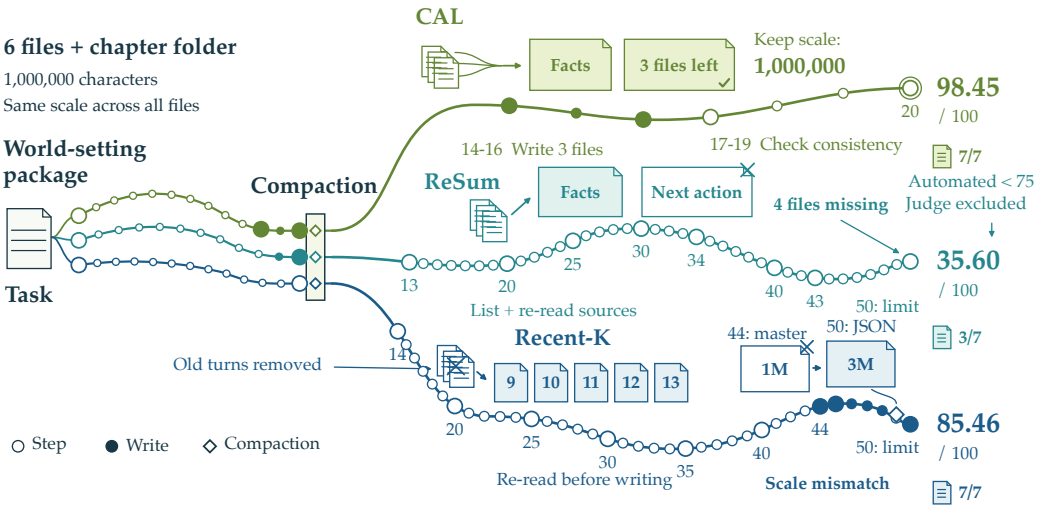


Figure 6: **World-setting generation.** Retaining pending deliverables and a shared scale constraint supports complete, consistent outputs. The crossed master record denotes context eviction; seven artifacts comprise six files and one directory.

E Experimental Details

Inference algorithm. Algorithm 1 details one native compaction call in Section 3.2. CAL preserves the original archive while the compactor inspects records and revises their presentation. A candidate view is installed only after satisfying the token budget and native message protocol; the executor then resumes from the same environment state. The archive and view metadata persist across later compactions.

Supervised initialization. We perform full-parameter SFT of the language-model component of Qwen3.6-35B-A3B with Megatron-SWIFT; the vision modules and aligner remain frozen. Table 9 lists the configuration. The resulting SFT checkpoint initializes both RL variants.

Reinforcement learning. Table 10 specifies the RL configuration for the objective in Equation (10). Each rollout batch contains $B = 24$ checkpoints with $m = 8$ continuations each, yielding 192 continuations. One RL epoch processes two disjoint mini-batches of 12 checkpoints (96 continuations) each. Within each mini-batch, we accumulate all micro-batch gradients before one optimizer update, giving two updates per rollout batch. The old-policy log probabilities remain fixed across both updates, so probability-ratio clipping can become active in the second update. The learning rate increases linearly over the first five optimizer updates and then remains at 10^{-6}.

The two reward channels are normalized separately within each group of eight continuations, using $\varepsilon = 10^{-6}$ and the masking rule below. We give equal weight to immediate and subsequent rubric scores ($\alpha = 0.5$) and combine outcome and temporal channels with $\lambda_{\text{out}} = 1$ and $\lambda_{\text{temp}} = 0.5$. The outcome-only ablation sets $\lambda_{\text{temp}} = 0$.

Reward masking. A `null` rubric score marks unavailable supervision. We mask the entire temporal channel of a continuation if any of its four criteria is `null`, preserving the fixed criterion weights and α in Section 3.3. Let $\mathcal{V}_i$ contain the continuations with four numerical scores and $n_i = |\mathcal{V}_i| \in \{0, \dots, 8\}$ be the effective temporal group size. A numerical score of zero remains valid. For $n_i \geq 2$, we compute r_{ij}^{temp} as defined in the main text and its group mean and population standard deviation using only $j \in \mathcal{V}_i$. The masked advantage is

$$\hat{v}_{ij}^{\text{temp}} = \begin{cases} \dfrac{r_{ij}^{\text{temp}} - \bar{r}_i^{\text{temp}}}{\sigma_i^{\text{temp}} + \varepsilon}, & j \in \mathcal{V}_i,\ n_i \geq 2,\ \sigma_i^{\text{temp}} > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{14}$$

Algorithm 1: `Compact0` inference through the Context Abstraction Layer.

Input : Task x; trained compactor π_θ; executor e; harness h; native compaction boundary $\mathbf{S}_t = (s_t, \mathcal{D}_t, c_t, b_t)$; CAL adapter $\mathcal{A}_h$, tool environment $\mathcal{F}_{e,h}$, and renderer $\mathcal{R}_h$
Output : Installed context $\tilde{c}_t$ and persistent view $\mathbf{V}^\star$ with execution resumed from s_t, or compaction failure

`/* Import context at the native compaction boundary */`
$(\mathcal{T}, \mathbf{V}_0) \leftarrow \mathcal{A}_h(\mathcal{D}_t, c_t)$ (Equation (3))
Prepare short sketches $\{q_i\}_{i \in \mathcal{I}}$ of archived turns; keep their originals intact
Initialize compactor input η_0 with x, sketches, $\mathbf{V}_0$, and budget b_t; $k \leftarrow 0$
while *compaction call and generation limits have not been reached* **do**
  Sample tool action $u_k \sim \pi_\theta(\cdot \mid \eta_k)$
  if $u_k = \text{inspect}(\mathcal{J})$ **then**
    `/* Read original evidence without changing retention */`
    $r_k \leftarrow \{d_i : i \in \mathcal{J}\}$ from archive $\mathcal{T}$; $\mathbf{V}_{k+1} \leftarrow \mathbf{V}_k$
  else if $u_k = \text{update}(\delta)$ **then**
    `/* Revise the view while preserving the archive */`
    $(\mathbf{V}_{k+1}, r_k) \leftarrow \mathcal{F}_{e,h}(\mathcal{T}, \mathbf{V}_k, u_k)$ (Equation (6))
    Read rendered token usage or edit-validation feedback from r_k
  else if $u_k = \text{finish}()$ **then**
    `/* Validate the candidate before installation */`
    Form candidate c' from $\mathcal{T}$ and $\mathbf{V}_k$ in native message format, preserving fixed system instructions
    if $\ell_{e,h}(c') \le b_t$ *and the native message protocol is valid* **then**
      $\mathbf{V}^\star \leftarrow \mathbf{V}_k$; persist view metadata alongside archive $\mathcal{T}$
      $\tilde{c}_t \leftarrow \mathcal{R}_h(x, \mathcal{T}, \mathbf{V}^\star)$; install the revised context in harness h
      Resume executor e from unchanged environment state s_t with $\tilde{c}_t$
      return $(\tilde{c}_t, \mathbf{V}^\star)$
    else
      $\mathbf{V}_{k+1} \leftarrow \mathbf{V}_k$; $r_k \leftarrow$ budget or protocol feedback requesting revision
  Append (u_k, r_k) to η_k to obtain η_{k+1}; $k \leftarrow k + 1$
return *compaction failure; terminate the continuation*

Table 9: Full-parameter SFT configuration.

Setting	Value
Learning rate / epochs	10^{-5} / 2
Maximum sequence length	86,016 tokens
Global / micro-batch size	16 / 1
Warmup fraction	5%
Tensor / expert parallel size	8 / 8
Precision	BF16

Thus, groups with fewer than two valid temporal rewards, including entirely missing groups, or zero temporal variance contribute no temporal update. Outcome rewards still use all eight continuations. All continuations remain in Equation (10), with its token denominator N_{mb} and reward weights unchanged; we do not rescale by the valid fraction. Missing scores are neither imputed nor re-judged, and available criteria are not reweighted to replace missing ones.

Compaction limits and evaluation. The training-time per-call token limits are listed in Table 10; evaluation triggers, soft targets, and call limits appear in Table 7. Native context admissibility is checked after rendering against the boundary-specific budget b_t with the executor tokenizer, as in Equation (2).

F Comparison of Context-Management Paradigms

Table 11 compares eight representative methods with `Compact0` in management initiation, context organization, and empirical generalization. Active management is initiated by the executor; passive

Table 10: RL and compaction configuration. Token limits use the compactor tokenizer.

Setting	Value
Checkpoints per rollout batch / PPO mini-batch	24 / 12
Continuations per checkpoint m	8
Continuations per rollout batch / optimizer update	192 / 96
PPO epochs per rollout batch	1
Optimizer updates per rollout batch	2
Micro-batch size per GPU	1
Learning rate	10^{-6}
Optimizer	AdamW
AdamW (β_1, β_2) / ϵ	$(0.9, 0.95)$ / 10^{-8}
Weight decay	0.01
Gradient norm clipping / precision	1.0 / BF16
Policy ratio clipping $(\epsilon_{\text{low}}, \epsilon_{\text{high}})$	$(0.20, 0.28)$
Advantage normalization ε	10^{-6}
Temporal scope weight α	0.5
Reward weights $(\lambda_{\text{out}}, \lambda_{\text{temp}})$	$(1.0, 0.5)$
KL / entropy regularization coefficients	0 / 0
Input plus output limit per model call	98,304 tokens
Generation limit per model call	2,048 tokens

management is triggered externally. We distinguish the harnesses and domains actually tested from explicit transfer to settings excluded from training or prompt optimization.

Table 11: Context-management paradigms and evaluated scope. Generalization (gen.): ✓/✗ indicates reported/unreported transfer to a held-out harness or task domain without target-specific optimization. Lists give tested runtimes and task families; "custom" denotes the authors' implementation. SAM combines passive consolidation and active recall.

Method	Initiation	Context representation and access	Harness gen.	Applicable harness	Domain gen.	Tested domain
AgentFold (Ye et al., 2025)	Active	Multi-scale summaries; latest raw interaction.	✗	ReAct	✗	Web research
ACM (Li et al., 2026c)	Active	Compression and query tools; external raw archive.	✗	ReAct	✗	Web research; coding
ReSum (Wu et al., 2025)	Passive	Running summary replaces accumulated history.	✗	ReAct	✗	Web research
SUPO (Lu et al., 2025)	Passive	Same-model summary and context reset.	✗	ReAct	✗	Synthetic coding tool use; web research
ACON (Kang et al., 2025)	Passive	History and observation compression.	✗	ReAct	✗	App/API use; office automation; multi-question QA
TRACE (Min et al., 2026)	Passive	Recurrent summary; recent raw turns.	✗	OpenClaw	✗	App/API use (AppWorld)
CoMem (Zhang et al., 2026)	Passive	Asynchronous summaries; recent-turn buffer.	✓	R2E-Gym; OpenHands; ReAct	✗	Coding; web research (separate models per domain)
SAM (Hu et al., 2026)	Passive / active	Raw pages, compact cues, and intent-driven recall.	✗	Custom SAM loop	✗	Web research
`Compact0`	Passive	CAL archive and reversible visible, compressed, merged, and hidden turn states.	✓	OpenCode; Pi; ReAct; NanoBot; Hermes	✓	Coding; deep research; streaming QA; Claw tasks

G TEMPORAL RUBRIC ANNOTATION PROMPTS

We use DeepSeek-V4-Pro as the fixed annotation LLM for the four temporal rubric criteria in Section 3.3. Prompts 2–3 assess constraint and state fidelity using only compaction-boundary evidence; Prompts 4–5 additionally use subsequent execution and CAL records to assess dependency continuity and outcome relevance. Each prompt requests a JSON judgment with a criterion, status, score, cited evidence, missing evidence, and a brief explanation. For supported judgments, status is `scored` and the score lies in $\{0, 0.25, 0.5, 0.75, 1\}$. Otherwise, status is `insufficient_evidence` and the score is `null`; reward masking is specified in Appendix E.

Prompt 2: Constraint Fidelity (Immediate)

Evaluate constraint_fidelity: how faithfully this compaction preserves the instruction scope and qualifiers governing the executor's continuation.

Use only the task, instructions and clarifications issued by the checkpoint, original boundary records, pre-compaction context, management episode, and installed post-compaction context. Ignore any later continuation or verifier result even if accidentally supplied.

Identify the active obligations before reading the candidate's summaries. Include the requested objective and deliverables, permitted and prohibited actions, file or workspace scope, ordering requirements, exceptions, conditions, numeric limits, source requirements, and user preferences that affect valid completion. Follow the recorded instruction hierarchy and latest applicable clarifications. Distinguish current obligations from superseded requests, descriptions of past work, suggestions, and quoted source material.

For each material obligation, compare what the executor can actually see after compaction with its original meaning. Check negation, modality, quantifiers, exceptions, temporal scope, entity identity, and boundaries such as "only," "unless," "at most," and "after approval." Inspect authored summaries and notes for instructions that broaden, narrow, or contradict the task. An unchanged pinned instruction counts as retained, but a contradictory summary is still a fidelity defect.

Recovery requires special care: keeping an instruction somewhere in the archive is not the same as preserving an immediately binding restriction in the executor's view. Credit a compact reference only if it makes the relevant restriction explicit and the supplied interface supports the required access before acting. Do not infer that the executor will remember an omitted condition.

Apply task-specific meaning without adding new requirements. For repository work, check allowed paths, requested changes, compatibility and testing obligations. For document or workspace tasks, check recipients, dates, permissions and output formats when specified. For research and QA, check entity relations, time qualifiers, evidence restrictions and answer format. In serial sessions, carry forward only instructions whose scope actually persists across tasks.

Select one anchor:

1.00 — All material active constraints retain their operative meaning in the installed view, without conflicting or invented instructions.

0.75 — The core scope and every critical restriction remain intact; a minor qualifier is weakened or less explicit without changing the main permissible actions.

0.50 — A material requirement or qualifier is lost or altered, leaving a partially usable but meaningfully incomplete task specification.

0.25 — A critical restriction, exception, or deliverable is lost or changed in a way that permits substantial deviation from the task.

0.00 — The installed context reverses a central prohibition, substitutes a materially different objective, or removes the core constraints needed to pursue the assigned task.

Use the worst material defect to distinguish adjacent anchors; do not average away a critical violation by listing many trivial retained details. Missing instructions or an incomplete installed view require insufficient_evidence and a null score when they prevent a supported comparison. A no-change view can score highly when preservation is verified; budget efficiency is evaluated separately.

Return the JSON object required by the shared system prompt, with criterion set to constraint_fidelity. Keep later_refs empty. Cite original instructions and their retained, weakened, missing, or conflicting counterparts in the installed view.

Prompt 3: State Fidelity (Immediate)

Evaluate state_fidelity: how accurately this compaction carries forward the established workspace or task state, uncertainty, and unresolved obligations at the checkpoint.

Use boundary-time evidence only. Reconstruct the latest supported state from original messages, tool observations and any checkpoint workspace snapshot supplied. Distinguish issued commands from their results, proposed edits from applied edits, attempted tests from completed tests, and pending operations from confirmed completion. Resolve stale observations using their recorded chronology and scope. A claim in an assistant message is not equivalent to a confirming tool observation.

Identify the state needed to continue from this checkpoint: relevant files and artifacts, edits already made, outputs already produced, tool results, checked conditions, failed approaches, active hypotheses, unresolved errors, pending actions and remaining deliverables. For research or QA, include established facts, source identities, disputed claims and unanswered parts. For serial tasks, track which facts remain current and which belong only to an earlier task.

Compare that state with the installed context. Inspect summaries, notes, retained turns and preserved task messages together. Check that the candidate preserves concrete identifiers and relations required to find or use the underlying evidence. A concise statement of a test's result can suffice; a fabricated "all tests pass" cannot. "The patch is planned," "the patch was applied," and "the patch was verified" describe different states.

A fact retained in protected recent turns need not be repeated in a summary. Omitting obsolete details or a reproducible raw log is acceptable when the needed result, uncertainty and access information remain clear. Archival existence alone does not establish that the executor knows a pending obligation. Record pre-existing errors separately; assess the distortion or repair attributable to this compaction, not an earlier executor mistake.

Select one anchor:

1.00 — All material continuation-relevant facts and open obligations are faithfully represented; verified, tentative, failed and pending states remain distinguishable.

0.75 — The current state and critical obligations remain accurate; a minor detail or noncritical status qualifier is omitted or imprecise.

0.50 — The view remains partly usable, but a material state fact or open obligation is lost, stale, or ambiguously represented.

0.25 — Several important facts are corrupted, or a critical pending action, error, artifact identity or verification status is misrepresented.

0.00 — The view invents or contradicts central state in a way that gives the executor a fundamentally false basis for continuation, such as presenting an unperformed critical operation as completed.

Select the anchor by the consequence of the observed state distortion, not by summary length or the count of correct incidental facts. If the evidence lacks the relevant tool result or checkpoint state, report that uncertainty rather than treating the candidate's statement as either confirmed or fabricated. If this prevents a reliable overall assessment, return insufficient_evidence and null.

Return the shared JSON object with criterion set to state_fidelity. Keep later_refs empty. Cite the boundary observation and the corresponding installed statement or omission. Do not use future workspace changes or later tests to establish what was true at the boundary.

Prompt 4: Dependency Continuity (Retrospective)

Evaluate dependency_continuity: whether evidence needed during continued execution remained available, or was actually recovered, before the executor needed to rely on it.

Inspect the boundary evidence, installed context, subsequent executor trajectory, and later CAL episodes. Focus on dependencies whose information was available at the target checkpoint and relevant to continuing the assigned task. Do not penalize the episode for genuinely new facts introduced only later or for unpredictable requirements from an unrelated future task.

For each material dependency, establish an evidence chain:

1. Locate the original fact, constraint, source, identifier, test result or unresolved issue at the checkpoint.

2. Identify how the target episode retained, summarized, referenced, or omitted it in the installed view.

3. Locate the later action that needed it and determine what the executor could actually see immediately before that action.

4. If recovery occurred, identify the successful retrieval and the request or installed view through which the information reached the executor. Compare that time with the first consequential use.

Manager inspection alone is not executor recovery. An archive entry, search hit or read request without a successful result and delivery does not complete the chain. Later management may recover a previously omitted record; credit the current episode's recovery design only when a retained cue or another observed link connects its decision to that timely recovery. Record an independent later rescue separately. In particular, if later compaction drops evidence that this episode retained adequately, attribute that new loss to the later episode.

Judge completeness at the needed granularity. A source title may be sufficient for navigation but insufficient for an exact date or quotation. A test name may locate a result but does not preserve whether it passed. Re-reading a file is not automatically a defect: judge whether correct information arrived before the dependent action. Repeated unsuccessful retrieval, acting on a distorted summary, and recovery only after a consequential mistake are continuity failures when linked to this episode.

Select one anchor:

1.00 — Every observed material dependency has a supported chain showing adequate retention or recovery before its consequential use, with no attributable interruption.

0.75 — Material dependencies arrive in time; one minor attributable gap causes limited extra search or delay without a consequential incorrect action.

0.50 — Continuity is mixed: an important dependency is lost or recovered late, causing a recoverable mistake or material rework, while substantial dependencies remain supported.

0.25 — A critical dependency is unavailable or distorted when needed, causing substantial derailment or an unresolved error traceable to this episode.

0.00 — The loss or corruption of indispensable evidence breaks the necessary continuation and is directly linked to a decisive failure or invalid result.

High scores require observed use or a demonstrably needed dependency, not merely plausible usefulness. A truncated continuation, an unobserved recovery path, or a trace with no identifiable dependence on boundary information requires insufficient_evidence and a null score. Full logs showing actual failed retrieval support a low score; missing retrieval logs do not.

Return the shared JSON object with criterion set to dependency_continuity. For each decisive finding, cite the boundary source, target installed representation or omission, and later need or recovery. State when another compaction, the executor, or the environment explains the effect instead.

Prompt 5: Outcome Relevance (Retrospective)

Evaluate outcome_relevance: the documented contribution of this episode's context decisions to subsequent task progress or failure.

Use the boundary evidence, accepted context installation, continuation, later CAL history, and available task-verifier evidence. Assess the quality of the context decision through its observed consequences. A clear trace of harmful compaction warrants a low score; the existence of a clear trace is not itself a reason for a high score.

Identify concrete progress markers or failure events: a correctly scoped edit, a verified artifact, a satisfied requirement, a resolved subproblem, a grounded answer, a repeated dead end, an invalid change, or a missed deliverable. Tie each decisive event to specific information that the current episode kept, summarized, omitted or distorted. Require an observable chain from the context decision to what the executor saw, the action it took, and the resulting task effect.

Retention can support progress even if the overall task later fails for an unrelated reason. Conversely, a correct final answer does not erase a demonstrated compaction-induced violation or major detour. The verifier's aggregate score is a separate outcome reward; use a detailed verifier observation only when the trace connects it to a particular context decision.

Distinguish reliance from coincidence. Text appearing in a summary and later in a correct answer can support a link when the action demonstrably uses it; shared vocabulary alone is weak evidence. An executor error despite accurate visible information is not automatically a compaction error. A timeout caused by an external service is not evidence of poor retention. Without a paired intervention, describe supported contribution rather than claiming that no other context could have succeeded.

Account for later episodes. If a later manager replaces the relevant state or independently retrieves omitted evidence, identify that intervention before assigning credit or blame. Attribute cumulative loss to the episode that introduced or materially worsened it when the records allow that distinction. If the sequence cannot be separated, report uncertain attribution instead of assigning the final outcome to the first compaction.

Select one anchor:

1.00 — Specific decisions in this episode demonstrably support a critical verified milestone or several substantive advances, with no material attributable harm in the observed continuation.

0.75 — The evidence links the episode to useful, substantive progress, with at most minor attributable friction or loss.

0.50 — There are supported beneficial and harmful effects of comparable importance, or a material compaction-induced detour is repaired and meaningful progress resumes. This anchor requires observed effects on both the context decision and continuation; it is not a default for uncertainty.

0.25 — The episode demonstrably causes substantial wasted work, a violated requirement or a major incorrect step, outweighing its supported contribution to progress.

0.00 — A specific retention or summarization decision is directly linked to a decisive task failure, invalid deliverable or central constraint violation.

When no material effect can be linked to this episode, or the evidence does not distinguish its effect from later management or other causes, return insufficient_evidence and null. Do not manufacture positive credit from final success or negative credit from final failure. Do not infer lost information solely from an unsuccessful action.

Return the shared JSON object with criterion set to outcome_relevance. Each decisive finding must cite the relevant pre-compaction evidence, installed view, and later action/result or verifier detail. State the direction and significance of the observed contribution in the brief explanation.