EVOSTATE: MULTI-OPERATION CONTEXT-STATE TRANSITIONS FOR LONG-HORIZON LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon LLM agents must preserve action-critical information while tracking which facts remain valid and which requirements remain unresolved. As execution progresses, new evidence can change both the status of retained information and its relevance to subsequent actions. We introduce **EvoState**, a training-free framework for **maintaining evolving context state**. The framework models context state in three complementary parts: Current Trace for ongoing execution evidence, Event Memory for historical outcomes, and Task State for current facts and requirements, each with distinct retention and update rules. Its Multi-Operation Context Editor combines rule-based updates with conditional LLM-based consolidation to maintain these records as new evidence arrives. The Dependency-Aware Context Assembler uses the revised state to prioritize exact dependencies, current facts, and unresolved requirements for the next decision, allowing necessary evidence to exceed a soft token target. Using GLM-5.1, we evaluate EvoState on AppWorld Normal and Challenge, BFCL Long Context, and an interactive exact-match subset of CRMArena-Pro. EvoState lies on the observed success–token Pareto frontier in all four settings, with gains of up to **7.1 percentage points** in success and reductions of up to **29.4%** in tokens per task relative to Full History. Controlled analyses further show improved reference retention and next-action progress.

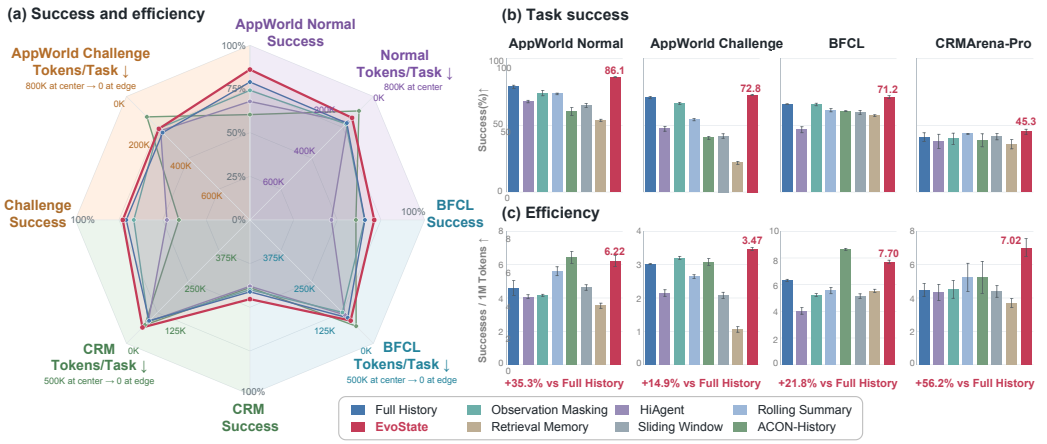


Figure 1: Task success and token efficiency with GLM-5.1. **(a)** Three-run mean success and tokens per task; outward positions indicate higher success or lower expenditure. **(b)** Mean task success. **(c)** Successful tasks per million tokens, computed per run and then averaged. Error bars show sample standard deviations. Annotations in **(c)** give EvoState's relative gains over Full History. Token totals include failed trajectories and the input and output of both acting and context-management calls. BFCL denotes BFCL Long Context; CRM denotes the evaluated CRMArena-Pro subset.

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.

1 INTRODUCTION

Long-horizon LLM agents rely on earlier observations to choose and execute later actions (Yao et al., 2023; Yang et al., 2024). Tool responses supply identifiers, runtime bindings, and intermediate results (Schick et al., 2023; Trivedi et al., 2024); later interactions may replace those values or reveal additional requirements. Useful evidence must persist across searches, corrections, and changes in local objectives. Replaying the complete history incurs repeated input cost, and models can struggle to use evidence reliably in long inputs (Liu et al., 2024). Effective context management must reduce redundant history replay while preserving information needed for subsequent actions.

Prompt compression, selective retention, and summaries reduce the history presented to the model (Jiang et al., 2023; Lindenbauer et al., 2025; Kang et al., 2026). Memory systems organize experience for reuse (Packer et al., 2023; Park et al., 2023; Hu et al., 2025), while recent methods revise records or learn maintenance policies (Xu et al., 2025; Zhou et al., 2026). Continued execution also requires tracking how new evidence changes retained information: which values remain current, which requirements remain unresolved, and which exact references later actions may still use.

The difficulty is that **retained information changes in validity and relevance during execution**. An observation can supply a reusable identifier, replace a previous fact, and change the assessed status of a requirement. The resulting records have different lifetimes: the current value can change while its history remains useful, and an exact dependency can outlast the surrounding exploration. Its entity association or variable binding must remain available for reuse. Maintaining these records requires coordinated updates that also guide the next input. We investigate the following question: **How can an agent maintain context state while reducing redundant history replay?**

We introduce **EvoState**, a training-free framework that maintains evolving context state (Figure 2). It maintains execution evidence in Current Trace, historical outcomes in Event Memory, and current facts and requirements in Task State, each with distinct retention and update rules (§3.2). As new observations arrive, its **Multi-Operation Context Editor** applies compatible edits that preserve reusable evidence while revising facts and requirements (§3.3). The **Dependency-Aware Context Assembler** uses the resulting state to construct the next input (§3.4). Resolved requirements lose their priority as unfinished work, while exact dependencies can remain protected across work units. Necessary evidence may exceed a soft token target within the model's context limit.

With **GLM-5.1** for acting and context management, EvoState combines high task success with reduced token expenditure on AppWorld Normal and Challenge, BFCL Long Context, and an interactive exact-match subset of CRMArena-Pro (Figure 1). It lies on the observed success–token Pareto frontier in all four settings. Relative to Full History, mean success is 2.0–7.1 percentage points higher with 10.5–29.4% fewer tokens per task. Difficulty-stratified AppWorld results show how this trade-off varies across task groups (§4.2). All six ablated variants lower success and raise tokens per success over three runs (§4.3).

Two controlled analyses on AppWorld Normal examine the maintenance and use of evidence (§4.4). Given the same recorded actions and observations, EvoState retains every audited reference at **99.8%** of 463 checkpoints, compared with 89.2% for Single-Role Editing. Given matched memory and environment state, full assembly yields next-action progress at **82.7%** of 202 decisions, compared with 66.3% for Dependency-Agnostic Assembly. The progress difference persists when both inputs contain every audited reference, linking the maintained state to its use in subsequent action. A single-run DeepSeek-V4-Flash evaluation examines the same trade-off (§4.5).

Our contributions are threefold. **(1)** We model **evolving context state** through the distinct retention and update needs of execution evidence, historical outcomes, and current facts and requirements. **(2)** We develop **multi-operation editing and dependency-aware assembly** to maintain this state from observed interactions and construct subsequent model inputs. **(3)** We demonstrate **improved success–token trade-offs** across three benchmarks, with three-run ablations and controlled analyses of reference retention and next-action progress.

2 Related Work

Context Compression and Selective Retention. Prompt compression shortens model inputs by selectively retaining or rewriting earlier context (Jiang et al., 2023; 2024; Pan et al., 2024). For interactive agents, sliding windows keep recent interactions, observation masking removes older tool outputs while preserving the action structure (Lindenbauer et al., 2025), and generative summaries compress longer histories (Wang et al., 2025). ACON (Kang et al., 2026) learns compression guidance from contrasting trajectories to preserve critical information and studies compressor distillation, while AgentFold (Ye et al., 2026) learns folding operations at different temporal scales. EvoState defines evidence-supported update rules for the information retained through compression. New evidence can invalidate a retained value or resolve a requirement, changing its priority in the next input. Exact dependencies remain protected when surrounding records are summarized, and a soft token target allows their inclusion when needed for execution.

Structured Agent Memory. Memory systems store and organize information outside the immediate prompt. MemGPT (Packer et al., 2023) manages transfers across memory tiers, while Generative Agents (Park et al., 2023) combine remembered experiences with reflection and retrieval. HiAgent (Hu et al., 2025) organizes working memory around subgoals and retrieves earlier trajectories; A-MEM (Xu et al., 2025) links notes and revises their contextual descriptions; and SeCom (Pan et al., 2025) combines topical segmentation with compression-based denoising. LightMem (Fang et al., 2026) separates filtering, memory construction, and consolidation to reduce memory-maintenance overhead. LoCoMo (Maharana et al., 2024) and LongMemEval (Wu et al., 2025) evaluate long-term recall and information use across extended conversations. EvoState gives historical records and current state distinct retention and update rules. A fact replacement updates Task State while preserving the earlier outcome in Event Memory and any supporting trace still needed for reuse. Multi-operation editing updates these records together; their revised status guides the next input.

Learned Memory and Context Adaptation. MEM1 (Zhou et al., 2026) learns compact states that combine consolidation with reasoning, while MemAgent (Yu et al., 2026) learns recurrent state updates while processing long inputs segment by segment. ACE (Zhang et al., 2026) keeps model weights fixed and incrementally adapts contextual playbooks through generation, reflection, and curation, accumulating reusable strategies across tasks. Our approach also keeps model parameters fixed, using received observations to update the context state of an ongoing task. These edits determine how records are retained and used in subsequent inputs. A record can remain stored as historical evidence after its value is superseded, while the assembler prioritizes the currently supported value for the next action.

3 Method

EvoState maintains context throughout the action–observation loop (Figure 2). New evidence can change retained values, resolve requirements, or establish dependencies for later actions. The framework realizes context transitions through compatible edits to the maintained state; the revised state then guides dependency-aware assembly of the next model input.

3.1 Problem Formulation

We consider an LLM agent interacting with a stateful environment. An interaction consists of an action and its resulting environment observation. Before action a_t, let g_t be the active request, o_t the latest observation, and S_t the maintained context state. The actor groups interactions under a local objective to define the current work unit w_t. The assembler $\mathcal{A}$ constructs the model input X_t for an acting policy with fixed parameters θ:

$$X_t = \mathcal{A}(g_t, S_t, w_t, o_t), \qquad a_t \sim \pi_\theta(\cdot \mid X_t). \tag{1}$$

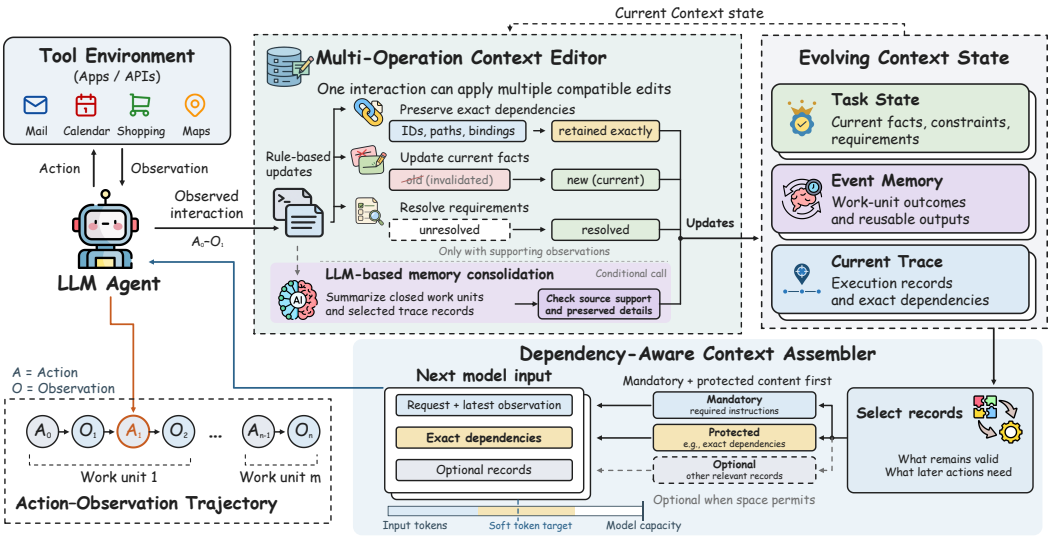


Figure 2: Overview of EvoState. Current Trace, Event Memory, and Task State represent execution evidence, historical outcomes, and current facts and requirements. The editor maintains this state through rule-based updates and conditional LLM-based consolidation. One interaction can combine compatible edits to retain exact evidence, create events, revise facts, and resolve requirements, subject to source and preservation checks. The assembler includes mandatory and protected content, then selects optional records under a soft token target within the model's context limit.

The environment returns o_{t+1}, yielding the interaction $r_{t+1} = (a_t, o_{t+1})$. The editor $\mathcal{E}$ produces an edit script that updates the maintained state:

$$\Delta_{t+1} = \mathcal{E}(S_t, r_{t+1}), \qquad S_{t+1} = \mathrm{Apply}(S_t, \Delta_{t+1}). \tag{2}$$

We omit the editor's request and work-unit arguments for clarity. Both modules use only received instructions, observed interactions, and existing memory. New user messages update g_t while reusable evidence can persist. The actor and consolidation model keep fixed parameters.

3.2 Context State Representation

To preserve earlier evidence as facts and requirements change, EvoState represents context in three complementary parts with distinct update and retention rules:

$$S_t = (C_t, E_t, T_t), \qquad T_t = (K_t, F_t, D_t, U_t, I_t). \tag{3}$$

Current Trace C_t preserves ordered execution evidence, including actions, observations, errors, and corrections. Exact dependencies are values or associations that later actions must reproduce faithfully, including identifiers, parameter values, paths, and variable bindings. Records remain until redundant or adequately replaced; binding updates favor current values during selection.

Event Memory E_t records a closed work unit's objective, observed outcome, reusable outputs, and blockers. Its contents remain fixed while retained, even when later observations supersede a recorded value. Outcomes include progress, failure, and unresolved work.

Task State T_t maintains current constraints K_t, facts F_t, resolved requirements D_t, unresolved requirements U_t, and invalidated values I_t. Entries record sources, evidence status, and update times. New evidence can replace facts or reopen requirements. Hypotheses are marked separately.

The three parts jointly support evidence reuse and state revision. A dependency can appear as a typed state entry and in the trace preserving its assignment or entity associations. Source references link current entries to that execution evidence. Event outputs record observations at closure, while Task State tracks later changes. Appendix A.1 details the representation and update rules.

3.3 MULTI-OPERATION CONTEXT EDITING

A single interaction can change several retained records at once. The **Multi-Operation Context Editor** captures these changes in a script $\Delta_{t+1} = (\delta_1, \ldots, \delta_m)$ of m compatible edits. *Retain records* and *Summarize records* maintain Current Trace; *Create event* records closed-unit outcomes; *Update state*, *Resolve*, and *Invalidate* revise Task State; *Remove* discards redundant or adequately replaced records (Appendix A.2). Predefined rules extract supported edits from recognized fields and execution records without an LLM call:

$$\Delta_{t+1}^{\text{rule}} = \mathcal{E}_{\text{rule}}(S_t, r_{t+1}), \qquad \widetilde{S}_{t+1} = \text{Apply}(S_t, \Delta_{t+1}^{\text{rule}}). \tag{4}$$

LLM-based memory consolidation builds on this updated state. It is triggered at work-unit boundaries or when an ongoing unit accumulates enough new, compressible material (Appendix A.3). A model $\mathcal{C}_\phi$ summarizes selected records $\mathcal{R}_{t+1}$ and proposes edits:

$$\Delta_{t+1}^{\text{llm}} = \begin{cases} \mathcal{C}_\phi(\widetilde{S}_{t+1}, \mathcal{R}_{t+1}), & \text{if consolidation is triggered,} \\ \varnothing, & \text{otherwise.} \end{cases} \tag{5}$$

Consolidation records the preceding unit's outcomes at a boundary, before incorporating evidence from the next unit. Within a unit, it summarizes older traces without closing it. Rules can form events for short units with explicit outcomes. Consolidation reorganizes evidence while the actor handles action selection and task computation. Checked proposals are merged with rule-based edits:

$$\Delta_{t+1} = \text{Merge}\Big(\Delta_{t+1}^{\text{rule}}, \mathcal{V}(\Delta_{t+1}^{\text{llm}}; \mathcal{R}_{t+1}, \widetilde{S}_{t+1})\Big). \tag{6}$$

Here $\mathcal{V}$ checks source support and preserved information, and Merge reconciles operations, prioritizing directly supported changes over conflicting proposals. The script maintains value validity and requirement status. Resolution moves a requirement from U_t to D_t with matching evidence; later evidence can reopen it. Replacement installs a new value under an existing key and records its predecessor in I_t. Revocation withdraws a value, and later evidence can restore it and clear its invalidation. Traces and events may retain superseded values as history, separate from current facts.

An evidence-supported summary may still omit information needed from its source. The editor checks both **source support** and **adequate replacement**. Exact values and their associations receive additional protection, allowing fact updates and event creation to coexist with retained supporting traces. Removal requires redundancy or adequate replacement. Appendices A.2–A.3 detail validation, operation order, and consolidation.

3.4 DEPENDENCY-AWARE CONTEXT ASSEMBLY

State revisions guide subsequent input selection. The assembler selects the next input using updated records and their validity and requirement status. **Mandatory content** $\mathcal{M}_t$ includes the active request, interaction instructions, necessary constraints, workflow and progress information, and the latest observation or selected content from it. **Protected content** $\mathcal{P}_t$ includes exact dependencies, state entries needed to continue execution, unresolved requirements for task completion, and records needed for the current work unit. Remaining traces, facts, and events form the **optional pool** $\mathcal{Z}_t$:

$$X_t = \text{Render}(\mathcal{M}_t, \mathcal{P}_t, \mathcal{Z}_t^\star), \qquad \mathcal{Z}_t^\star \subseteq \mathcal{Z}_t. \tag{7}$$

Mandatory and protected content is included first. Resolving a requirement removes its protection as outstanding work; invalidating a value removes its priority as a current binding. Exact dependencies can remain protected across work units as older noncritical records become optional. **Retention and input inclusion are separate decisions**: stored records remain available even when omitted from the current input. Events are ranked by lexical relevance to the task and unresolved requirements, together with recency. Selected actions remain grouped with their observations.

The **soft token target** B_s regulates input length, but necessary and sufficiently high-priority optional content may exceed it within the model's context limit. Protected content that cannot fit is not silently truncated. The assembler first compacts long observations without altering fields or values, then selects relevant fields and text spans if needed. Selection can omit input content while preserving the original memory record. Appendix A.4 specifies selection and capacity handling.

Figure 3 follows these updates through an AppWorld development task. Conversation evidence is preserved in an event for repayment. The subsequent payment observation is retained while its transaction identifier updates Task State. Before notification, the assembler combines earlier phone-contact evidence with the current login binding, retaining the marked exact records beyond the soft token target. This sequence illustrates how state updates support evidence reuse across work units; Appendix G.1 gives selected execution and memory excerpts.

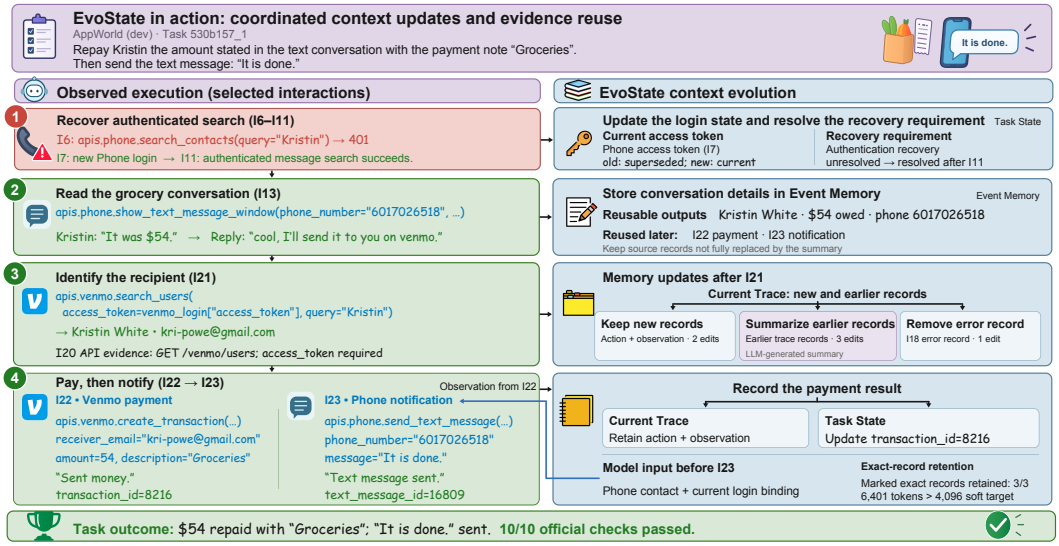


Figure 3: **Multi-operation editing and evidence reuse in an AppWorld task.** Selected interactions (left) induce compatible memory edits (right). Before I23, the assembled input includes earlier contact evidence, the current login binding, and all three marked exact records across work units, exceeding the soft token target. Appendix G.1 provides the detailed trace.

4 Experiments

4.1 Experimental Setup

We evaluate **AppWorld Normal** (168 tasks) and **Challenge** (417 tasks) for stateful application execution (Trivedi et al., 2024), **BFCL Long Context** (200 conversations) for multi-turn function calling (Patil et al., 2025), and an interactive exact-match subset of **CRMArena-Pro** (136 tasks; CRM) for enterprise queries with evolving constraints (Huang et al., 2026). The CRM subset contains four tasks for each combination of 17 task types and two organization types. Baselines are Full History, Sliding Window, Observation Masking (ObsMask), Generative Rolling Summary (Rolling Summary), Step Retrieval Memory (Retrieval Memory), HiAgent, ACON-History, and StructMem (Xu et al., 2026). Appendix B documents task selection, method settings, and interaction limits.

The primary backbone is **GLM-5.1** (GLM-5-Team et al., 2026) for acting and context management, with temperature zero and reasoning disabled. We use three runs with shared task sets and official success criteria. Token totals include acting and context-management input and output, including failed trajectories. Table 1 reports mean success and pooled tokens per success with conditional 95% bootstrap intervals. Appendices B–C give settings, metric definitions, per-run results, and paired tests.

4.2 Main Results

EvoState combines high task success with a favorable success–token trade-off. It lies on each setting's observed Pareto frontier, with mean success 2.0–7.1 percentage points above Full History and 10.5–29.4% fewer tokens per task (Table 1; Figure 4). Holm-adjusted paired tests support BFCL's success gain and token reductions on Normal, Challenge, and CRM (Appendix C.4).

Table 1: Main results with GLM-5.1 over three runs. Cells report mean success or pooled tokens per success (M: million tokens), with conditional 95% bootstrap intervals beneath. Bold marks the best point estimate per metric.

Benchmark	**AppWorld Normal (168)**		**AppWorld Challenge (417)**		**BFCL Long Context (200)**		**CRMArena-Pro subset (136)**	
Method	**Success (%) ↑ 95% CI**	**Tokens/ success (M) ↓ 95% CI**	**Success (%) ↑ 95% CI**	**Tokens/ success (M) ↓ 95% CI**	**Success (%) ↑ 95% CI**	**Tokens/ success (M) ↓ 95% CI**	**Success (%) ↑ 95% CI**	**Tokens/ success (M) ↓ 95% CI**
Agent: GLM-5.1								
Full Context								
Full History	79.0 [71.2,85.9]	0.219 [0.174,0.273]	70.8 [64.7,76.7]	0.331 [0.287,0.384]	65.8 [59.7,71.8]	0.158 [0.136,0.186]	41.2 [35.5,46.8]	0.223 [0.183,0.274]
Selective Retention								
Sliding Window	64.9 [55.4,74.4]	0.216 [0.163,0.289]	42.0 [35.9,48.3]	0.481 [0.397,0.592]	59.8 [53.7,65.7]	0.195 [0.171,0.222]	41.4 [35.8,47.3]	0.228 [0.181,0.288]
Observation Masking	74.2 [65.7,82.3]	0.240 [0.189,0.306]	66.5 [60.3,72.4]	0.312 [0.272,0.362]	65.7 [59.5,71.7]	0.192 [0.169,0.218]	40.0 [34.6,45.3]	0.222 [0.179,0.275]
Generative Compression								
Rolling Summary	73.6 [65.3,81.5]	0.179 [0.143,0.223]	54.3 [48.0,60.6]	0.378 [0.323,0.447]	61.3 [55.2,67.3]	0.180 [0.159,0.205]	43.6 [38.0,49.3]	0.194 [0.159,0.237]
ACON-History	60.3 [51.2,69.2]	**0.156** [0.123,0.199]	40.7 [34.2,47.2]	0.325 [0.269,0.402]	60.7 [54.7,66.5]	**0.116** [0.104,0.130]	38.7 [33.3,44.4]	0.193 [0.162,0.232]
Retrieval								
Retrieval Memory	53.8 [43.7,63.9]	0.283 [0.210,0.389]	21.9 [16.6,27.4]	0.947 [0.723,1.294]	57.3 [51.0,63.5]	0.180 [0.159,0.206]	35.8 [30.1,41.4]	0.270 [0.218,0.340]
Structured Memory								
HiAgent	67.9 [58.7,76.6]	0.246 [0.194,0.313]	47.6 [41.2,53.8]	0.465 [0.392,0.560]	46.7 [40.5,52.8]	0.248 [0.214,0.290]	38.0 [32.8,43.4]	0.232 [0.191,0.285]
StructMem	35.9 [26.6,45.6]	0.798 [0.573,1.167]	16.4 [11.7,21.4]	2.156 [1.589,3.125]	38.8 [33.3,44.3]	0.449 [0.383,0.531]	37.5 [32.4,42.6]	0.370 [0.312,0.443]
Ours								
EvoState	**86.1** [78.2,93.1]	0.161 [0.128,0.203]	**72.8** [67.2,78.2]	**0.288** [0.253,0.330]	**71.2** [65.3,76.7]	0.130 [0.117,0.145]	**45.3** [40.0,50.7]	**0.143** [0.123,0.168]

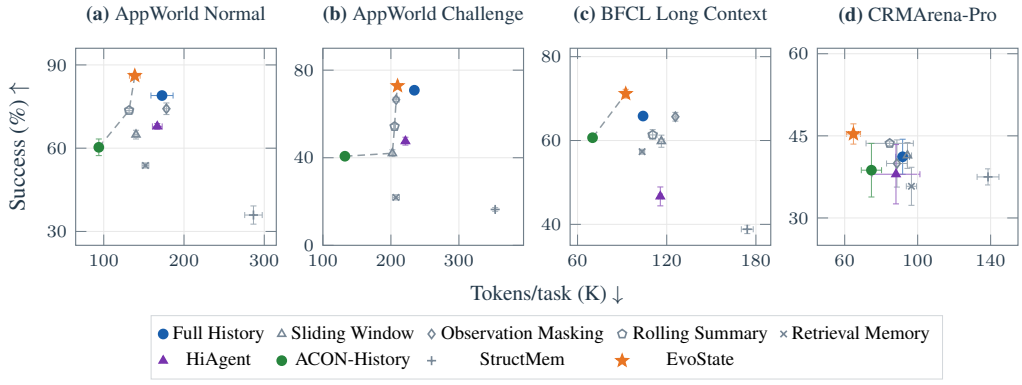


Figure 4: Success versus tokens per task with GLM-5.1. Points show three-run means; whiskers show sample standard deviations. Dashed segments connect nondominated means.

Lower expenditure can trade off against task success. ACON-History uses fewer tokens on AppWorld and BFCL at lower success. On BFCL, it reaches 60.67% at 70.10K tokens per task, compared with EvoState's 71.17% at 92.41K. Pooled tokens per success are lowest for EvoState on Challenge and CRM, and for ACON-History on Normal and BFCL (Appendix C.1).

AppWorld's difficulty labels reflect planning, reasoning, interaction demands, and task-specific hurdles (Trivedi et al., 2024); **higher difficulty need not imply a longer interaction trajectory**. The largest mean success gains over Full History occur on Medium in both splits (Table 2). On Challenge Hard, success is 72.99% versus 76.41%, with 6.6% fewer tokens per task.

Table 2: AppWorld success (%) by difficulty with GLM-5.1: mean ± sample SD over three runs. Appendix C.2 gives all baselines and scenario success.

	Normal ↑			Challenge ↑		
Method	Easy	Medium	Hard	Easy	Medium	Hard
Full History	89.5 ± 1.8	81.9 ± 3.2	67.2 ± 2.4	79.6 ± 2.1	59.3 ± 2.4	$\mathbf{76.4} \pm 1.0$
ObsMask	$\mathbf{95.3} \pm 2.7$	75.7 ± 1.2	54.0 ± 5.7	79.2 ± 3.7	58.0 ± 4.2	68.4 ± 2.1
HiAgent	84.8 ± 2.7	65.3 ± 6.7	54.5 ± 0.9	69.0 ± 0.8	40.0 ± 2.0	45.5 ± 3.9
EvoState	$\mathbf{95.3} \pm 1.0$	$\mathbf{93.1} \pm 1.2$	$\mathbf{72.5} \pm 0.9$	$\mathbf{81.0} \pm 1.6$	$\mathbf{68.7} \pm 0.7$	73.0 ± 1.3

4.3 Ablation on AppWorld Normal

Restricting editing and assembly reduces task success. All six variants cover Normal over three runs, with full EvoState as the reference (Figure 5; Appendix D.1). Restricting each update to one memory component (**Single-Role Editing**, N3) lowers success by 7.34 percentage points with only 1.45K fewer tokens per task. **Dependency-Agnostic Assembly (N6)** causes the largest decrease: 19.44 points, with 11.75K more tokens per task. This variant retains state storage and editing, showing the importance of using the maintained state to protect and prioritize context.

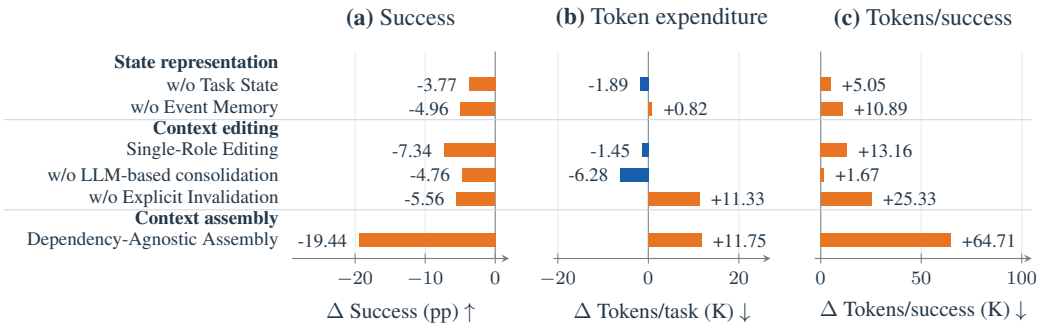


Figure 5: Three-run mean changes from full EvoState on Normal: **(a)** success (percentage points), **(b)** tokens per task, and **(c)** tokens per success. Blue indicates improvement; orange, degradation.

Removing Task State or Event Memory lowers success by 3.77 and 4.96 points at similar per-task cost. Work-unit prompting lowers Full History's mean success (76.39% versus 78.97%).

The remaining editing ablations examine consolidation and invalidation. Without LLM-based consolidation, success and tokens per task decrease, while mean tokens per success increase by 1.67K. Without explicit invalidation, success falls by 5.56 percentage points, mainly on Medium and Hard, and token costs rise. **All six variants increase mean tokens per success** (Figure 5c). We next examine reference retention after editing and task progress after assembly.

4.4 State Maintenance and Actionable Evidence Reuse

Multi-operation editing preserves references reused by later actions. Full EvoState and **Single-Role Editing (N3)** independently build memory from empty states using the same recorded actions and observations from all 168 Normal tasks at seed 0. We measure **complete reference retention** at acquisition checkpoints: all previously acquired tool-returned values reused later as arguments must remain recoverable as typed values or usable variable bindings.

EvoState retains all audited references at 462 of 463 checkpoints (99.8%), versus 413 (89.2%) for N3: a 10.6-percentage-point difference (95% CI: [6.2, 15.4]). The gap reaches 16.3 points at the 301 checkpoints requiring joint recovery of multiple references (Figure 6a). Across all checkpoints, 49 pass only with EvoState and none only with N3. Compatible edits preserve a state entry together with its supporting observation or variable assignment, keeping complementary references recoverable.

Dependency-aware assembly helps convert maintained evidence into task progress. We hold the recorded actions and observations, maintained state, and environment fixed before the first attempted

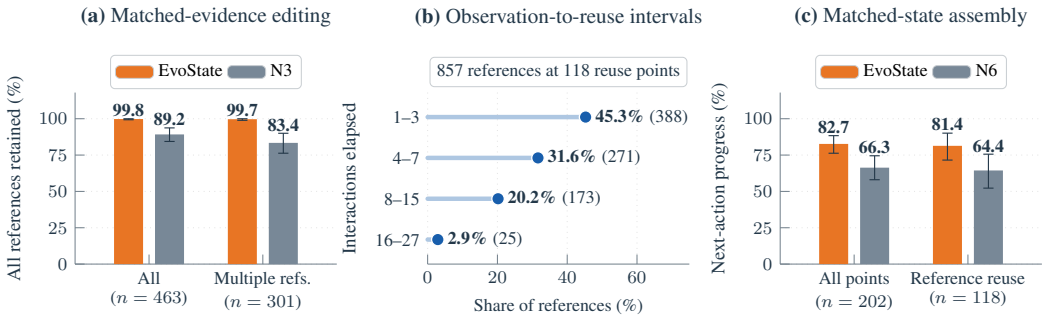


Figure 6: Context maintenance on Normal: **(a)** complete reference retention under matched evidence, **(b)** intervals between observation and reuse, and **(c)** next-action progress under matched state.

call to each task-goal API. Across 202 paired decisions from 154 tasks, full assembly advances at least one official check on 167 decisions (82.7%), versus 134 (66.3%) for **Dependency-Agnostic Assembly (N6)**: a 16.3-point difference (95% CI: [9.3, 23.6]).

At 118 decisions, 857 references are reused up to 27 interactions after observation (Figure 6b). All are available in both inputs; full assembly advances 96 decisions versus 76 for N6 (Figure 6c). Across all 202 pairs, execution errors without progress occur on 13 versus 25 actions. Full assembly uses 8.84K tokens per next-action request versus 7.17K for N6, including input and output. This comparison evaluates the assembly policy, including context selection, prioritization, and token allocation. Together, these analyses show how editing preserves reusable references and how assembly helps the actor use maintained evidence. Appendix C.3 details the audit criteria and paired outcomes.

4.5 Additional Backbone Evaluation

With DeepSeek-V4-Flash (DeepSeek-AI, 2026) for acting and context management, EvoState retains a **favorable success–token trade-off** across all three settings (Table 3). Relative to Full History, its largest success gain is on BFCL (48.0% to 60.0%), with tokens per task falling from 188.75K to 125.84K. ACON-History uses fewer tokens at lower success throughout; Appendix E gives the full results.

Table 3: DeepSeek-V4-Flash results, one run per task: success (%) / tokens per task (K). Bold marks the best value per metric.

Method	Normal Success ↑ / Tokens/task ↓	Challenge Success ↑ / Tokens/task ↓	BFCL Long Context Success ↑ / Tokens/task ↓
Agent: DeepSeek-V4-Flash			
Full History	70.24 / 253.59	47.96 / 409.57	48.00 / 188.75
Observation Masking	60.71 / 231.68	41.25 / 325.55	53.00 / 161.04
HiAgent	47.62 / 227.06	26.14 / 316.30	57.00 / 166.86
ACON-History	28.57 / **138.51**	14.87 / **177.68**	52.50 / **97.90**
EvoState	**76.79** / 208.07	**50.60** / 292.01	**60.00** / 125.84

5 Conclusion

EvoState maintains **evolving context state** through multi-operation editing and dependency-aware assembly. Evidence-supported edits update value validity and requirement status, guiding subsequent input selection. Results place EvoState on the observed success–token Pareto frontier, with high task success and fewer tokens per task than Full History. Controlled comparisons show better reference retention under matched evidence and greater next-action progress under matched state. Evaluation covers API-based tasks and two backbones; extending context-state maintenance to multimodal agents remains open.

AI USE STATEMENT

AI tools assisted with literature search and organization, experimental analysis, manuscript refinement, figure preparation, LaTeX typesetting and layout, and code implementation.

REFERENCES

DeepSeek-AI. DeepSeek-V4: Towards Highly Efficient Million-Token Context Intelligence, 2026.

Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Huajun Chen, and Ningyu Zhang. LightMem: Lightweight and Efficient Memory-Augmented Generation. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/a05b72653ec5b473732129829ae04195-Abstract-Conference.html.

GLM-5-Team, Aohan Zeng, Xin Lv, et al. GLM-5: from Vibe Coding to Agentic Engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. HiAgent: Hierarchical Working Memory Management for Solving Long-Horizon Agent Tasks with Large Language Model. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 32779–32798. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1575. URL https://aclanthology.org/2025.acl-long.1575/.

Kung-Hsiang Huang, Akshara Prabhakar, Onkar Thorat, Divyansh Agarwal, Prafulla Kumar Choubey, Yixin Mao, Silvio Savarese, Caiming Xiong, and Chien-Sheng Wu. CRMArena-Pro: Holistic Assessment of LLM Agents Across Diverse Business Scenarios and Interactions. *Transactions on Machine Learning Research*, 2026. URL https://openreview.net/forum?id=EPlpe3Fx1x.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 13358–13376. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.825. URL https://aclanthology.org/2023.emnlp-main.825/.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LongLLMLingua: Accelerating and Enhancing LLMs in Long Context Scenarios via Prompt Compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1658–1677. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.91. URL https://aclanthology.org/2024.acl-long.91/.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. ACON: Optimizing Context Compression for Long-horizon LLM Agents. In *ICML 2026*, June 2026. URL https://www.microsoft.com/en-us/research/publication/acon-optimizing-context-compression-for-long-horizon-llm-agents/.

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.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638. URL https://aclanthology.org/2024.tacl-1.9/.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating Very Long-Term Conversational Memory of LLM Agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13851–13870. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.747. URL https://aclanthology.org/2024.acl-long.747/.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as Operating Systems. arXiv preprint arXiv:2310.08560, 2023. URL https://arxiv.org/abs/2310.08560.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. LLMLingua-2: Data Distillation for Efficient and Faithful Task-Agnostic Prompt Compression. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 963–981. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.57. URL https://aclanthology.org/2024.findings-acl.57/.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Xufang Luo, Hao Cheng, Dongsheng Li, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Jianfeng Gao. SeCom: On Memory Construction and Retrieval for Personalized Conversational Agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/e56f394bbd4f0ec81393d767caa5a31b-Abstract-Conference.html.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative Agents: Interactive Simulacra of Human Behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. Association for Computing Machinery, 2023. doi: 10.1145/3586183.3606763. URL https://doi.org/10.1145/3586183.3606763.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 48371–48392. PMLR, 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems*, volume 36, pp. 68539–68551. Curran Associates, Inc., 2023. doi: 10.52202/075280-2997. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A Controllable World of Apps and People for Benchmarking Interactive Coding Agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.850. URL https://aclanthology.org/2024.acl-long.850/.

Qingyue Wang, Yanhe Fu, Yanan Cao, Shuai Wang, Zhiliang Tian, and Liang Ding. Recursively Summarizing Enables Long-Term Dialogue Memory in Large Language Models. *Neurocomputing*, 639:130193, 2025. doi: 10.1016/j.neucom.2025.130193. URL https://doi.org/10.1016/j.neucom.2025.130193.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking Chat Assistants on Long-Term Interactive Memory. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/d813d324dbf0598bbdc9c8e79740ed01-Abstract-Conference.html.

Buqiang Xu, Yijun Chen, Jizhan Fang, Ruobin Zhong, Yunzhi Yao, Yuqi Zhu, Lun Du, and Shumin Deng. StructMem: Structured Memory for Long-Horizon Behavior in LLMs. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 122–146. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-short.12. URL https://aclanthology.org/2026.acl-short.12/.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic Memory for LLM Agents. In *Advances in Neural Information Processing Systems*, volume 38, pp. 17577–17604. Curran Associates, Inc., 2025. doi: 10.52202/085713-0593. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/19909c36f51abc4856b4560aff3d36d6-Abstract-Conference.html.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652. Curran Associates, Inc., 2024. doi: 10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

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

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, Pengjun Xie, Fei Huang, Jingren Zhou, Siheng Chen, and Yong Jiang. AgentFold: Long-Horizon Web Agents with Proactive Context Folding. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/fa948624dfde013671e72c1a7ca4aebc-Abstract-Conference.html.

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, and Hao Zhou. MemAgent: Reshaping Long-Context LLM with Multi-Conv RL-based Memory Agent. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/4264ee4376776907c0b87ed70b959585-Abstract-Conference.html.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Y Zou, and Kunle Olukotun. Agentic Context Engineering: Evolving Contexts for Self-Improving Language Models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/8a94ff6f922d995d7d3f4ebf4143e442-Abstract-Conference.html.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Bryan Kian Hsiang Low, and Paul Liang. MEM1: Learning to Synergize Memory and Reasoning for Efficient Long-Horizon Agents. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/5fc8b3bdfbb9167b5144df5d3fae4616-Abstract-Conference.html.

APPENDIX CONTENTS

A Method details 14
A.1 Complete context-state representation 14
A.2 Edit semantics and ordering 15
A.3 Consolidation triggers and prompts 15
A.4 Ranking, selection, and context limits 16
B Evaluation protocol 17
B.1 Task sets and evaluation criteria 17
B.2 Execution environment and method settings 18
B.3 Statistical units and uncertainty 18
C Complete main-experiment results 19
C.1 Per-run outcomes and conditional intervals 19
C.2 Benchmark-defined task groups 22
C.3 State maintenance and actionable evidence reuse 24
C.4 Exploratory paired significance tests 25
D Component ablations 28
D.1 Variant definitions and measurements 28
E Cross-backbone evaluation 31
E.1 Model identities and inference settings 31
E.2 Within-backbone results 31
F Execution and cost measurements 32
G Qualitative trajectories 34
G.1 Evidence reuse across work units 35
G.2 A complete multi-operation editing cycle 36
G.3 Organizing evidence and tracking remaining work 39
G.4 Preserving evidence across ordered actions 42
G.5 File identities across work-unit boundaries 43
G.6 Completed effects and pending work 44

A METHOD DETAILS

A.1 COMPLETE CONTEXT-STATE REPRESENTATION

The maintained context is $S_t = (C_t, E_t, T_t)$, with $T_t = (K_t, F_t, D_t, U_t, I_t)$ as defined in Section 3.2. Its contents may overlap: Task State exposes current status, Current Trace preserves supporting execution content, and Event Memory records outcomes observed at work-unit closure. Each follows distinct retention and update rules.

Current Trace. Local records retain actions, observations, errors, corrections, or exact dependencies, together with their source interaction and validity. Exact dependencies include identifiers, paths, parameter values, entity associations, and runtime bindings needed for faithful reuse. Records remain available across work-unit boundaries when their necessary content has not been adequately replaced.

Recognizing exact dependencies. The editor identifies exact dependencies from observed interactions and existing memory. Rule-based updates retain recognized values and their supporting records, while LLM-based memory consolidation can propose additional source-supported dependencies, subject to the checks in Appendix A.2. Retained dependencies remain linked to their sources and entity associations.

Event Memory. An event records a closed work unit's objective, observed outcome, reusable outputs, blockers, and supporting interactions. Outcomes include failure, blockage, abandonment, submission, and progress as well as success. Events remain historical accounts when related current facts change; maintenance may remove them but does not rewrite their contents to reflect later evidence.

Task State. Constraints K_t capture restrictions; facts F_t retain observed or verified values; D_t and U_t distinguish resolved from unresolved requirements; I_t identifies values no longer treated as current. Each slot carries a key, value, source, evidence status, and update time; requirements also specify evidence for resolution. Hypotheses cannot establish completion. Contrary evidence can reopen a requirement or restore a previously invalidated value as current.

State keys and update scope. Within a task, entries are addressed by their Task State component and stored key. Generic identifier and location fields are qualified by their producing tool, and authentication-token keys by their application; more specific fields retain their field names. Entity associations remain in the source records or structured mappings. When one response contains multiple entity identifiers under the same field, the rules retain them as structured collection evidence rather than collapsing them into a single scalar fact. Collection-coverage requirements are identified by the tool and non-pagination query arguments, allowing successive pages of the same query to contribute to one requirement.

Replacement and collection accumulation have different meanings. A new value for an existing fact key replaces its predecessor and records the old value as invalidated. Collection coverage instead accumulates observed pages and member identifiers within the same query scope. An additional page contributes evidence without making earlier members obsolete; terminal evidence resolves the corresponding coverage requirement. The pagination example in Appendix G.3.1 illustrates how an incomplete collection remains open until later observations establish its end.

Requirement creation and resolution. Requirements enter Task State through explicit structured observations, rule-based recognition of remaining work, or source-supported proposals during LLM-based memory consolidation. Task requirements are grounded in the received request and observed unfinished work. Rules also create execution-local requirements, such as recovery after repeated failures of the same tool or coverage of an incomplete collection. An entry retains its source and the evidence needed for resolution, expressed as a verification condition or structured description of what remains to be established.

Resolution is specific to the requirement. A task condition needs observations supporting the requested effect; collection coverage needs terminal evidence for the corresponding query. A later

successful call matching the blocked operation's scope can resolve its recovery requirement. LLM-proposed resolutions are checked against the supporting observations before moving an entry from U_t to D_t. Restoring a blocked capability does not establish completion of the user request: in Appendix G.4, successful playlist retrieval resolves the recovery condition while backup and account deletion remain to be performed.

A.2 Edit semantics and ordering

Table 4 gives the seven operations and their evidence conditions. **One interaction can induce multiple compatible edits**, jointly preserving an exact dependency, revising a fact, invalidating its predecessor, and resolving a matching requirement.

Table 4: Editing operations and their evidence requirements. Multiple compatible operations may follow one observed interaction.

Operation	Meaning and condition
Retain records	Retain an observed execution record or exact dependency in Current Trace.
Summarize records	Replace selected trace records with a faithful compact representation while preserving necessary dependencies and requirements.
Create event	Record a closed work unit's supported outcome, reusable outputs, and blockers; closure does not imply task success.
Update state	Insert or revise a fact, constraint, or requirement using supported evidence; distinguish hypotheses from verified values.
Resolve	Mark an unresolved requirement as resolved only when its evidence condition is met.
Invalidate	Stop treating a value as current after replacement, deletion, expiry, or contradiction.
Remove	Discard redundant or adequately replaced material without losing an unresolved requirement or required exact dependency.

Source observations take precedence over unsupported LLM proposals; hypotheses cannot overwrite verified facts or resolve requirements. After reconciling redundant or contradictory operations, edits are applied in the order of invalidation, state update, resolution, event creation, retention, summarization, and removal.

Algorithm 1 applies each candidate update to a copy and commits it only after validation. Unsupported LLM proposals are removed before merging. If the combined update fails, a separately validated rule-only update is attempted, without accepting a closure dependent on the rejected proposal. If both fail, stored state remains unchanged and the received interaction remains recorded. The provisional state $\widetilde{S}$ supplies consolidation input and is not committed in advance.

Validation checks source support, exact dependencies, evidence for resolution, replacement coverage, and consistency of current and invalidated values. Citing a source alone does not justify removing it: a replacement must preserve its necessary content. These checks address specified forms of information loss without establishing semantic equivalence for arbitrary text.

A.3 Consolidation triggers and prompts

LLM-based consolidation summarizes closed work units or older records within an ongoing unit. Boundaries follow the actor's optional declarations of local objectives; short units with explicit outcomes can form events without an LLM call. Consolidation within an ongoing unit leaves it open.

The reference token budget for deciding when to consolidate is 4,096 tokens. **Exceeding it alone does not trigger a model call**: selected material must offer useful compression, and recent inputs are not resubmitted without sufficient new content. Within an ongoing work unit, a call based on accumulated material requires at least 5,120 tokens of eligible records, at least 1,024 tokens of useful new material, and an eight-interaction interval. Consolidation at work-unit boundaries separately checks whether a model call is useful. These criteria regulate context-management calls, not the acting model's input or output limit.

Algorithm 1 Multi-operation context editing with evidence checks

Require: Current stored state S; received evidence $\mathcal{R}$; active request g; work-unit boundary information b
Ensure: Validated state, or unchanged S; the received interaction remains recorded

RECORDINTERACTION($\mathcal{R}$)
$\Delta^{\text{rule}} \leftarrow$ RULEBASEDUPDATES$(S, \mathcal{R}, g, b)$
$\widetilde{S} \leftarrow$ APPLYTOCOPY$(S, \Delta^{\text{rule}})$
$\Delta^{\text{llm}} \leftarrow \varnothing$
if SHOULDCONSOLIDATE$(\widetilde{S}, \mathcal{R}, b)$ **then**
    $\mathcal{R}' \leftarrow$ SELECTRECEIVEDEVIDENCE$(\widetilde{S}, \mathcal{R}, b)$
    $\Delta^{\text{llm}} \leftarrow \mathcal{C}_\phi(\widetilde{S}, \mathcal{R}', g, b)$
    $\Delta^{\text{llm}} \leftarrow$ CHECKSOURCESANDPRESERVEDDETAILS$(\Delta^{\text{llm}}, \mathcal{R}', \widetilde{S})$
end if
$\Delta \leftarrow$ RECONCILEANDORDER$(\Delta^{\text{rule}}, \Delta^{\text{llm}})$
$S' \leftarrow$ APPLYTOCOPY(S, Δ)
if VALIDATEUPDATES$(\Delta, S, S', \mathcal{R}, b)$ **then**
    return S'
end if
$(\Delta^{\text{fallback}}, b^{\text{fallback}}) \leftarrow$ RULEONLYUPDATES$(S, \mathcal{R}, g, b)$
$S^{\text{rule}} \leftarrow$ APPLYTOCOPY$(S, \Delta^{\text{fallback}})$
if VALIDATEUPDATES$(\Delta^{\text{fallback}}, S, S^{\text{rule}}, \mathcal{R}, b^{\text{fallback}})$ **then**
    return S^{rule}
end if
return REJECT(S)

Consolidation instructions. The prompt supplies the objective, memory after rule-based updates, latest action, and observation, and asks for the smallest evidence-supported edit. Its constraints are:

- Preserve the objective's scope, entities, relations, quantities, and completion conditions. Use only supplied evidence; an action request does not prove its outcome.
- Preserve exact values and their associations. Keep uncertainty explicit, and resolve only a requirement whose stated evidence condition has been satisfied.
- Do not select the actor's next action, invent recovery work, or compute new task answers. Previously executed and observed computations may be retained.
- Within an ongoing work unit, summarize redundant trace records without closing the unit. At a work-unit boundary, summarize the preceding unit's supported outcomes and blockers in one event.
- Distinguish citing a source from fully replacing it. Preserve a source whenever the proposed event or summary does not cover its necessary content.

Compatible edits remain permitted; unchanged memory requires no operation.

A.4 RANKING, SELECTION, AND CONTEXT LIMITS

Algorithm 2 selects mandatory ($\mathcal{M}$), protected ($\mathcal{P}$), and optional ($\mathcal{Z}$) content. Mandatory content includes the request, instructions, applicable constraints, and latest observation or selected content from it. Protected content covers valid exact dependencies, state entries and records needed to continue execution, and unresolved task requirements. Other traces, facts, and events are optional candidates.

Actions are grouped with their corresponding observations and presented in interaction order. If a work unit grows beyond twice the 4,096-token soft target, selection prioritizes exact dependencies, interactions needed to continue execution, the latest action, and recent errors. Other recent interaction groups are selected around the soft target; older groups remain stored as optional candidates. Thus protection does not extend indefinitely to every interaction in a long unit.

Events are ranked using term overlap with the current query and a recency preference. For event e closed at interaction t_e, let $O(e, q)$ denote query-term overlap with its objective, outcome, reusable

outputs, and blockers, and let $B(e, q)$ denote overlap with its blockers. The score is

$$\text{score}(e, q) = 3O(e, q) + 2B(e, q) + \frac{1}{1 + t_{\max} - t_e},$$

where $t_{\max}$ is the most recent event closure. Ties favor more recent events. Ranking supplies candidate order; it does not override validity or dependency requirements.

For a long observation, the assembler first uses compact formatting that preserves all fields and values. If still too long, it selects relevant fields, identities, dependencies, and text spans. This selection can omit information from the current input **without deleting the original record**. Source removal remains a separate editing decision.

Let H be the actual model context capacity and R the output reserve. The complete model request, including instructions and tool descriptions, must satisfy

$$\text{tokens}(X_t) + R \leq H.$$

The 4,096-token **selection target is soft**: mandatory and protected content, and optional records prioritized by the selection policy, may exceed it. The model's context limit remains a hard constraint. If necessary protected content cannot fit, execution reports a capacity failure instead of silently truncating it.

In Algorithm 2, $\|$ appends candidate groups and H_{in} reserves space for output and fixed request overhead. Selection is adjusted if the complete request exceeds capacity. Beyond the soft target, optional content is prioritized by dependency links, shared entity or tool identifiers, unresolved requirements, current recent values, and query overlap relative to length. Overlap alone cannot admit older raw interactions without a dependency link beyond the target. Rendering preserves interaction order, with the latest observation last.

Algorithm 2 Dependency-aware context assembly

Require: Request g; state S; work unit w; latest observation o; soft token target B_s; available input capacity H_{in}
Ensure: Selected model input or an explicit capacity failure; stored state S is unchanged

$\widehat{o} \leftarrow o$
if TooLong($\widehat{o}$) **then**
  $\widehat{o} \leftarrow$ CompactFormat(o)
  if TooLong($\widehat{o}$) **then**
    $\widehat{o} \leftarrow$ SelectFieldsAndSpans($\widehat{o}, g, S$)
  end if
end if
$(\mathcal{M}, \mathcal{P}, \mathcal{Z}) \leftarrow$ ClassifyCandidates($g, S, w, \widehat{o}$)
$L \leftarrow \mathcal{M} \,\|\, \mathcal{P}$
if RenderedTokens(L) $> H_{\text{in}}$ **then**
  $X \leftarrow$ Render(L)
  return CapacityFailure(X)
end if
for each interaction bundle or candidate group G in OrderCandidates($\mathcal{Z}, g, w, S$) **do**
  $\widehat{G} \leftarrow$ ShortenIfNeeded(G, g, S)
  $n \leftarrow$ RenderedTokens($L \,\|\, \widehat{G}$)
  if $n \leq H_{\text{in}}$ **and** ($n \leq B_s$ **or** PrioritizedBeyondTarget(G, g, w, S)) **then**
    $L \leftarrow L \,\|\, \widehat{G}$
  end if
end for
return RenderInput(L)

B Evaluation protocol

B.1 Task sets and evaluation criteria

AppWorld uses all 168 Normal tasks (56 scenarios) and 417 Challenge tasks (139 scenarios). A scenario contains three task instances. Easy/Medium/Hard correspond to official difficulty labels

1/2/3: the counts are 57/48/63 for Normal and 72/150/195 for Challenge. These hand-assigned labels consider planning, reasoning, interaction demands, and task-specific hurdles (Trivedi et al., 2024); they do not directly measure the length of an agent's execution trajectory. Strict task success is determined by the official state tests; **scenario success requires all three instances to pass**.

BFCL uses all 200 conversations in `multi_turn_long_context`, ordered by the numeric task-ID suffix. The **complete conversation is the unit of evaluation**. The official multi-turn and irrelevance checkers jointly determine success.

CRM uses **17 exact-match task types**, two organization types, and four tasks in each of 34 strata. The eligible pool has 100 tasks per stratum (3,400 total). Within each stratum, a fixed deterministic pseudorandom ordering selects four tasks; the resulting task set is shared by all methods and run labels. This gives 68 B2B and 68 B2C tasks. The two fuzzy-match types and privacy-rejection records are outside this subset. Success uses the official exact-match reward, without an LLM judge.

B.2 Execution environment and method settings

In the primary evaluation, all methods use API-based GLM-5.1 (GLM-5-Team et al., 2026) inference on the same evaluation server, with **temperature zero and reasoning disabled**. Acting and context-management models remain fixed. Task timeout is 1,800 seconds. No local model training is involved.

AppWorld allows 40 interactions and reserves 8,192 output tokens, without a separate actor output cap. All methods use a 204,800-token capacity on both Normal and Challenge.

BFCL uses a 204,800-token capacity, 176 runner steps, an agent output cap of 1,024 tokens, and an 8,192-token output capacity reserve. The output cap and reserve are distinct settings. The benchmark termination and force-quit rules are preserved.

CRM uses a 204,800-token capacity, a 2,000-token agent output cap, and a 4,096-token reserve. The limits are 210 runner steps, 10 user turns, and 20 agent rounds per user turn. **Initial messages are fixed across methods**; subsequent simulator responses remain action-dependent, with **simulator seed 0 for all agent run labels**.

Table 5: Method settings for the primary GLM-5.1 evaluation. Benchmark-specific capacity and output limits appear in Appendix B.2.

Method	Configuration
Full History	Complete interaction history.
Sliding Window	Six complete interactions.
Observation Masking	Keep the latest ten observations verbatim and all actions; replace older observations with placeholders.
Rolling Summary	Compression threshold 4,096 tokens; preserve the three latest interactions.
Retrieval Memory	Five selected interactions, including the two most recent; cosine retrieval over fixed 2,048-dimensional text-feature representations.
HiAgent	Memory size 100; two declaration retries; summary output cap 512.
ACON-History	History-only compression; threshold 4,096; retain one latest turn; compressor output cap 4,096.
EvoState	Rule-based updates and conditional LLM-based memory consolidation; soft targets of 4,096 for assembly, consolidation decisions, and long observations; consolidation under the criteria in Appendix A.3; source and dependency checks.

B.3 Statistical units and uncertainty

Table 1 reports three-run mean success and the **pooled ratio of total method tokens to total successes**, with conditional 95% bootstrap intervals. Boldface marks the best point estimate. Appendix C.1 also gives tokens per task, individual runs, and sample standard deviations with variance divisor $3 - 1$.

We additionally compute conditional percentile bootstrap intervals using 20,000 replicates and a fixed random seed. **AppWorld resamples scenarios**, preserving all three instances and their three

run labels together. BFCL resamples conversations with all run labels together. CRM resamples four task identities with replacement within each of the original 34 strata, keeping the strata equally weighted. **Each bootstrap draw is shared across methods**, preserving pairing. Intervals use the 2.5th and 97.5th percentiles. **The three run labels are held fixed**; repeated labels are not counted as independent new tasks.

The intervals describe variation under resampling of evaluation units with run labels held fixed; sample standard deviations describe variation across runs. Table 6 compares EvoState with Full History: success-difference intervals include zero on Challenge and CRM. Appendix C.4 gives exploratory paired tests with Holm adjustment. Unadjusted percentile intervals and adjusted tests use different constructions and may yield different decisions.

Table 6: EvoState versus Full History: paired effects [conditional 95% bootstrap intervals] and two-sided cluster-test p-values. Holm adjustment jointly covers both metrics and all baselines within each setting. Positive effects favor EvoState. Bold $p_{\mathrm{H}} < 0.05$ denotes an exploratory conditional result.

Setting	Δ SR (pp) [CI]	p_{H}	Token cut (%) [CI]	p_{H}
AppWorld Normal	7.14 [1.19, 13.29]	0.115	19.66 [15.05, 24.23]	$\mathbf{6.00 \times 10^{-5}}$
AppWorld Challenge	2.00 [-2.16, 6.31]	1.000	10.53 [4.78, 15.73]	**0.006**
BFCL Long Context	5.33 [1.50, 9.33]	**0.023**	11.22 [0.45, 21.11]	0.059
CRM exact-match	4.17 [-0.25, 8.58]	0.469	29.40 [21.49, 36.29]	**0.001**

C Complete main-experiment results

C.1 Per-run outcomes and conditional intervals

Figure 7 compares EvoState with every baseline. **Success gains are percentage-point differences** in three-run means. Cost reductions are $100(1 - C_{\mathrm{EvoState}}/C_{\mathrm{baseline}})$, using mean tokens per task or pooled tokens per success, including failures. The tables below give individual runs, sample standard deviations, and conditional intervals.

	(a) Success Gain (pp) ↑				**(b)** Tokens/task Reduction (%) ↑				**(c)** Tokens/success Reduction (%) ↑			
	Normal	Chall.	BFCL	CRM	Normal	Chall.	BFCL	CRM	Normal	Chall.	BFCL	CRM
Full History	+7.1	+2.0	+5.3	+4.2	+19.7	+10.5	+11.2	+29.4	+26.3	+13.0	+17.9	+35.9
Sliding Window	+21.2	+30.8	+11.3	+3.9	+1.1	−3.8	+20.6	+31.4	+25.5	+40.1	+33.3	+37.3
Observation Masking	+11.9	+6.3	+5.5	+5.4	+22.1	−1.0	+26.6	+26.9	+32.9	+7.7	+32.2	+35.6
Rolling Summary	+12.5	+18.5	+9.8	+1.7	−5.4	−2.1	+16.4	+23.5	+9.9	+23.9	+28.0	+26.4
ACON-History	+25.8	+32.1	+10.5	+6.6	−47.8	−58.5	−31.8	+13.2	−3.5	+11.5	−12.4	+25.9
Retrieval Memory	+32.3	+50.9	+13.8	+9.6	+8.8	−1.2	+10.6	+32.9	+43.0	+69.6	+28.0	+47.1
HiAgent	+18.3	+25.3	+24.5	+7.4	+16.8	+5.2	+20.1	+26.5	+34.5	+38.1	+47.6	+38.4
StructMem	+50.2	+56.4	+32.3	+7.8	+51.6	+40.6	+47.0	+53.2	+79.8	+86.6	+71.1	+61.3

EvoState improvement EvoState disadvantage

Figure 7: EvoState relative to each baseline. Green denotes improvement and orange disadvantage. Normal and Chall. denote the AppWorld splits.

On Challenge, EvoState reduces tokens per success against every baseline even though several methods use fewer tokens per task. On Normal and BFCL, its tokens per success exceed ACON-History by 3.5% and 12.4%, respectively. On CRM, EvoState lies on the observed success–token Pareto frontier.

Table 7: AppWorld Normal: outcomes for each run label.

Method	Run 0	K/task	Run 1	K/task	Run 2	K/task
Full History	132/168	184.97	135/168	157.74	131/168	175.08
Sliding Window	107/168	139.46	112/168	138.62	108/168	142.45
Observation Masking	128/168	179.99	125/168	179.20	121/168	175.05
Rolling Summary	123/168	134.14	125/168	125.94	123/168	134.72
Retrieval Memory	92/168	147.86	89/168	156.21	90/168	151.84
HiAgent	116/168	171.52	113/168	168.72	113/168	159.87
ACON-History	102/168	92.16	96/168	94.85	106/168	94.49
StructMem	60/168	277.39	66/168	283.63	55/168	298.60
EvoState	145/168	130.96	145/168	142.15	144/168	142.88

Table 8: AppWorld Challenge: outcomes for each run label.

Method	Run 0	K/task	Run 1	K/task	Run 2	K/task
Full History	291/417	231.86	297/417	235.52	298/417	236.05
Sliding Window	171/417	203.01	172/417	202.92	183/417	200.56
Observation Masking	281/417	209.21	274/417	202.79	277/417	211.00
Rolling Summary	228/417	206.46	222/417	205.78	229/417	203.99
Retrieval Memory	97/417	202.46	88/417	206.77	89/417	212.86
HiAgent	194/417	221.52	194/417	222.88	207/417	219.71
ACON-History	171/417	132.87	165/417	133.57	173/417	130.74
StructMem	65/417	353.99	68/417	352.68	72/417	353.28
EvoState	303/417	207.82	306/417	210.04	302/417	211.47

Table 9: BFCL Long Context: outcomes for each run label.

Method	Run 0	K/task	Run 1	K/task	Run 2	K/task
Full History	132/200	102.87	131/200	104.65	132/200	104.73
Sliding Window	118/200	118.46	123/200	115.91	118/200	114.83
Observation Masking	132/200	126.66	129/200	126.06	133/200	124.75
Rolling Summary	123/200	112.86	125/200	107.28	120/200	111.48
Retrieval Memory	114/200	101.85	114/200	105.31	116/200	103.10
HiAgent	89/200	116.63	93/200	116.28	98/200	114.15
ACON-History	122/200	69.81	121/200	70.11	121/200	70.36
StructMem	76/200	177.66	77/200	175.11	80/200	169.83
EvoState	143/200	92.75	140/200	92.63	144/200	91.84

Table 10: CRM exact-match: outcomes for each run label.

Method	Run 0	K/task	Run 1	K/task	Run 2	K/task
Full History	59/136	95.57	51/136	91.71	58/136	88.02
Sliding Window	55/136	96.60	54/136	94.50	60/136	92.14
Observation Masking	50/136	93.34	52/136	82.32	61/136	90.20
Rolling Summary	60/136	89.53	59/136	94.49	59/136	69.98
Retrieval Memory	54/136	99.18	45/136	93.62	47/136	96.96
HiAgent	46/136	88.18	49/136	74.94	60/136	101.21
ACON-History	45/136	79.79	57/136	75.41	56/136	68.77
StructMem	51/136	136.43	53/136	133.95	49/136	145.34
EvoState	62/136	60.30	64/136	67.15	59/136	66.91

Table 11: AppWorld Normal: success and K/task show mean ± sample SD across run labels, with conditional 95% bootstrap intervals beneath. K/success shows the conditional interval for the pooled token/success ratio.

Method	Success (%)	K/task	K/success (CI)
Full History	78.97 ± 1.24 [71.23, 85.91]	172.60 ± 13.78 [144.59, 202.65]	[174.36, 273.16]
Sliding Window	64.88 ± 1.57 [55.36, 74.40]	140.18 ± 2.02 [119.39, 162.26]	[162.91, 289.10]
Obs. Masking	74.21 ± 2.09 [65.67, 82.34]	178.08 ± 2.65 [153.25, 204.01]	[189.26, 305.69]
Rolling Summary	73.61 ± 0.69 [65.28, 81.55]	131.60 ± 4.91 [112.87, 152.01]	[143.49, 223.01]
Retrieval Memory	53.77 ± 0.91 [43.65, 63.89]	151.97 ± 4.18 [132.49, 171.81]	[210.03, 388.58]
HiAgent	67.86 ± 1.03 [58.73, 76.59]	166.70 ± 6.08 [141.99, 193.20]	[194.35, 313.50]
ACON-History	60.32 ± 3.00 [51.19, 69.25]	93.84 ± 1.46 [82.53, 105.92]	[123.11, 198.86]
StructMem	35.91 ± 3.28 [26.59, 45.63]	286.54 ± 10.90 [259.39, 313.39]	[572.88, 1167.35]
EvoState	86.11 ± 0.34 [78.17, 93.06]	138.67 ± 6.68 [116.56, 162.71]	[128.12, 203.38]

Table 12: AppWorld Challenge: success and K/task show mean ± sample SD across run labels, with conditional 95% bootstrap intervals beneath. K/success shows the conditional interval for the pooled token/success ratio.

Method	Success (%)	K/task	K/success (CI)
Full History	70.82 ± 0.91 [64.75, 76.66]	234.48 ± 2.28 [212.85, 257.36]	[286.84, 383.55]
Sliding Window	42.05 ± 1.60 [35.89, 48.28]	202.16 ± 1.39 [188.73, 215.60]	[396.61, 591.96]
Obs. Masking	66.51 ± 0.84 [60.27, 72.42]	207.66 ± 4.32 [192.09, 223.81]	[271.54, 361.53]
Rolling Summary	54.28 ± 0.91 [47.96, 60.59]	205.41 ± 1.27 [191.89, 219.09]	[323.13, 446.65]
Retrieval Memory	21.90 ± 1.18 [16.63, 27.42]	207.37 ± 5.23 [195.76, 219.19]	[722.60, 1293.91]
HiAgent	47.56 ± 1.80 [41.25, 53.80]	221.37 ± 1.59 [206.18, 236.92]	[392.45, 560.24]
ACON-History	40.69 ± 1.00 [34.21, 47.24]	132.39 ± 1.48 [125.43, 139.31]	[269.21, 401.51]
StructMem	16.39 ± 0.84 [11.67, 21.42]	353.32 ± 0.65 [338.36, 367.40]	[1588.63, 3125.30]
EvoState	72.82 ± 0.50 [67.23, 78.18]	209.78 ± 1.84 [193.72, 226.41]	[253.17, 329.82]

Table 13: BFCL Long Context: success and K/task show mean ± sample SD across run labels, with conditional 95% bootstrap intervals beneath. K/success shows the conditional interval for the pooled token/success ratio.

Method	Success (%)	K/task	K/success (CI)
Full History	65.83 ± 0.29 [59.67, 71.83]	104.08 ± 1.05 [90.89, 119.52]	[135.59, 185.68]
Sliding Window	59.83 ± 1.44 [53.67, 65.67]	116.40 ± 1.86 [105.28, 127.91]	[171.27, 221.80]
Obs. Masking	65.67 ± 1.04 [59.50, 71.67]	125.82 ± 0.98 [112.90, 139.40]	[168.67, 217.92]
Rolling Summary	61.33 ± 1.26 [55.17, 67.33]	110.54 ± 2.91 [101.00, 120.51]	[159.06, 205.31]

BFCL Long Context (continued)

Method	Success (%)	K/task	K/success (CI)
Retrieval Memory	57.33 ± 0.58 [51.00, 63.50]	103.42 ± 1.75 [94.48, 112.87]	[158.94, 205.89]
HiAgent	46.67 ± 2.25 [40.50, 52.83]	115.69 ± 1.34 [104.89, 126.84]	[213.76, 289.99]
ACON-History	60.67 ± 0.29 [54.67, 66.50]	70.10 ± 0.28 [65.62, 74.71]	[103.63, 129.77]
StructMem	38.83 ± 1.04 [33.33, 44.33]	174.20 ± 4.00 [159.94, 189.07]	[383.33, 531.00]
EvoState	71.17 ± 1.04 [65.33, 76.67]	92.41 ± 0.49 [86.71, 98.29]	[117.25, 144.69]

Table 14: CRM exact-match: success and K/task show mean ± sample SD across run labels, with conditional 95% bootstrap intervals beneath. K/success shows the conditional interval for the pooled token/success ratio.

Method	Success (%)	K/task	K/success (CI)
Full History	41.18 ± 3.21 [35.54, 46.81]	91.77 ± 3.77 [81.32, 103.09]	[182.95, 274.28]
Sliding Window	41.42 ± 2.36 [35.78, 47.30]	94.41 ± 2.23 [81.26, 108.19]	[181.50, 287.54]
Obs. Masking	39.95 ± 4.31 [34.56, 45.34]	88.62 ± 5.67 [77.13, 100.25]	[179.31, 275.12]
Rolling Summary	43.63 ± 0.42 [37.99, 49.26]	84.66 ± 12.96 [74.33, 95.31]	[159.36, 236.99]
Retrieval Memory	35.78 ± 3.47 [30.15, 41.42]	96.58 ± 2.80 [85.42, 108.30]	[217.91, 340.10]
HiAgent	37.99 ± 5.42 [32.84, 43.38]	88.11 ± 13.14 [78.15, 98.59]	[190.66, 284.78]
ACON-History	38.73 ± 4.90 [33.33, 44.36]	74.66 ± 5.55 [68.18, 81.17]	[162.01, 232.04]
StructMem	37.50 ± 1.47 [32.35, 42.65]	138.57 ± 5.99 [125.56, 151.84]	[312.09, 442.78]
EvoState	45.34 ± 1.85 [39.95, 50.74]	64.79 ± 3.89 [59.77, 69.98]	[122.63, 168.21]

C.2 BENCHMARK-DEFINED TASK GROUPS

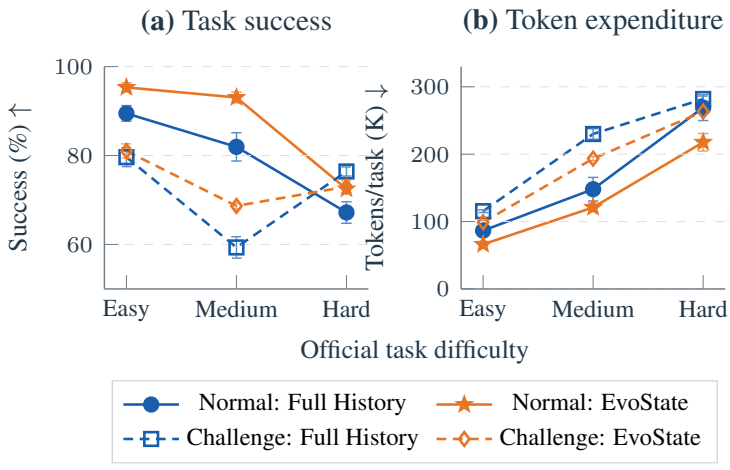


Figure 8: Full History and EvoState by AppWorld difficulty. **(a)** Task success; **(b)** tokens per task. Points show three-run means; whiskers show sample standard deviations.

Success–token trade-offs vary across difficulty groups. The labels describe task demands, while execution length also depends on the agent's actions and recovery attempts. On Normal, EvoState's mean success exceeds Full History in all three groups, most on Medium (11.1 percentage points), with 18.2–23.6% fewer tokens per task (Figure 8). On Challenge, Medium gains 9.3 percentage points and reduces tokens per success by 27.4%. Hard uses 6.6% fewer tokens per task, but success falls from 76.41% to 72.99%, leaving a 2.2% reduction in tokens per success. Challenge scenario success is also slightly lower (55.64% versus 56.83%). CRM likewise shows larger mean success differences in B2B than B2C. The tables below give complete subgroup results.

Token savings also need not come from shorter trajectories. On Challenge and CRM, actor-input savings exceed the added cost of LLM calls for context management despite longer execution, consistent with less context replay (Appendix F). Token reductions do not imply proportional latency or monetary savings.

Table 15: AppWorld Normal: difficulty and scenario success, mean ± sample SD (%). Difficulty counts are given in Appendix B.1.

Method	Easy	Medium	Hard	Scenario
Full History	89.47 ± 1.75	81.94 ± 3.18	67.20 ± 2.42	63.10 ± 2.06
Sliding Window	93.57 ± 2.68	65.97 ± 2.41	38.10 ± 4.20	47.02 ± 1.03
Observation Masking	95.32 ± 2.68	75.69 ± 1.20	53.97 ± 5.72	57.14 ± 5.36
Rolling Summary	92.98 ± 1.75	65.97 ± 4.34	61.90 ± 3.17	53.57 ± 3.57
Retrieval Memory	87.13 ± 2.68	52.08 ± 2.08	24.87 ± 0.92	34.52 ± 1.03
HiAgent	84.80 ± 2.68	65.28 ± 6.70	54.50 ± 0.92	45.83 ± 3.72
ACON-History	80.70 ± 4.64	56.25 ± 11.60	44.97 ± 2.42	39.88 ± 5.15
StructMem	71.35 ± 2.68	25.69 ± 8.67	11.64 ± 0.92	17.86 ± 1.79
EvoState	95.32 ± 1.01	93.06 ± 1.20	72.49 ± 0.92	77.98 ± 1.03

Table 16: AppWorld Challenge: difficulty and scenario success, mean ± sample SD (%). Difficulty counts are given in Appendix B.1.

Method	Easy	Medium	Hard	Scenario
Full History	79.63 ± 2.12	59.33 ± 2.40	76.41 ± 1.03	56.83 ± 1.25
Sliding Window	68.52 ± 2.12	43.33 ± 2.00	31.28 ± 2.24	23.74 ± 1.44
Observation Masking	79.17 ± 3.67	58.00 ± 4.16	68.38 ± 2.07	51.56 ± 0.42
Rolling Summary	71.76 ± 2.12	49.33 ± 2.31	51.62 ± 0.78	36.21 ± 2.31
Retrieval Memory	54.63 ± 2.89	22.44 ± 1.92	9.40 ± 0.59	10.31 ± 1.66
HiAgent	68.98 ± 0.80	40.00 ± 2.00	45.47 ± 3.92	27.58 ± 2.91
ACON-History	75.00 ± 2.78	35.11 ± 3.01	32.31 ± 2.66	26.38 ± 2.91
StructMem	49.54 ± 0.80	17.56 ± 1.68	3.25 ± 1.57	7.43 ± 0.42
EvoState	81.02 ± 1.60	68.67 ± 0.67	72.99 ± 1.29	55.64 ± 1.50

Table 17: CRM organization groups: 68 tasks per organization per run label. Values are mean ± sample SD.

Method	B2B (%)	B2B K/task	B2C (%)	B2C K/task
Full History	40.69 ± 5.16	90.06 ± 6.75	41.67 ± 3.40	93.48 ± 6.16
Sliding Window	41.67 ± 2.25	104.79 ± 14.24	41.18 ± 5.30	84.03 ± 16.06
Observation Masking	41.18 ± 4.41	76.78 ± 3.98	38.73 ± 5.94	100.46 ± 8.73
Rolling Summary	46.08 ± 3.06	89.65 ± 24.64	41.18 ± 2.55	79.68 ± 5.70
Retrieval Memory	37.25 ± 1.70	94.70 ± 13.66	34.31 ± 8.62	98.47 ± 10.71
HiAgent	41.67 ± 5.16	83.69 ± 3.80	34.31 ± 7.40	92.53 ± 23.11
ACON-History	37.75 ± 4.73	82.07 ± 10.42	39.71 ± 5.09	67.24 ± 4.54
StructMem	38.73 ± 1.70	157.36 ± 15.80	36.27 ± 1.70	119.78 ± 9.23
EvoState	47.55 ± 5.94	68.39 ± 4.48	43.14 ± 2.25	61.18 ± 3.33

C.3 State maintenance and actionable evidence reuse

We examine reference retention after editing under matched observations and next-action progress after assembly from the same maintained state. The editing audit checks references reused later in the source trajectory; the assembly comparison executes each arm's own next action.

Matched-evidence editing. We examine seed-0 EvoState trajectories for all 168 Normal tasks in 56 scenarios, including successes and failures. Full EvoState and Single-Role Editing (N3) **each start with empty memory** and process the same recorded actions and observations. Each editor maintains its own state and makes its own consolidation calls. The resulting 336 memory trajectories are evaluated at common evidence checkpoints.

Reference checklist. For the retrospective audit, we trace typed values from recorded tool responses to their later use as call arguments. These include entity identifiers, contact details, file references, and access bindings. Identifiers are matched within their application; contact and file references may cross applications. A checkpoint follows the first acquisition of at least one such reference. Its checklist contains every reference acquired by that point that is used again later in the recorded trajectory. Acquisition, use, and checkpoint membership are determined from tool observations and arguments, **independently of either editor's retention decisions**.

The audit contains 1,941 distinct task-specific references and 463 checkpoints. Of these checkpoints, 301 contain two or more references, and 269 combine newly acquired references with earlier references that remain in use. Repeated checks of a reference with a later use yield 3,107 reference–checkpoint pairs. This construction includes every task, including successes and failures.

Availability criterion. A reference is available when the maintained memory contains its typed value, an equivalent file path, or a visible reference to a live variable with the matching content. The audit decodes structured collections and lossless tabular representations, and accepts named collections of entity identifiers. An untyped number occurring elsewhere in memory is insufficient. Likewise, a variable existing in the execution environment must remain identifiable from memory to count as available. The check covers Current Trace, Event Memory, and Task State before context selection. A checkpoint passes only when **all references on its checklist are available**; it measures complete reference retention at that point.

Table 18: Reference retention under matched evidence. Each entry counts checkpoints retaining every audited reference. Multiple-reference and new-plus-earlier rows are subsets of all checkpoints. Differences are EvoState minus N3 in percentage points.

Checkpoints	EvoState	N3	Difference [95% CI]
All	462/463	413/463	10.6 [6.2, 15.4]
Multiple references	300/301	251/301	16.3 [9.7, 23.4]
New plus earlier references	268/269	221/269	17.5 [10.5, 24.8]

EvoState retains all audited references at 49 checkpoints where N3 does not; there are no checkpoints with the reverse outcome. At the individual-reference level, the counts are 3,100/3,107 for EvoState and 2,648/3,107 for N3. For example, an update may preserve a contact identifier in Task State while the corresponding phone number remains only in the accompanying observation. Restricting the update to one component can lose that complementary evidence. Multi-operation editing can retain the observation and update the binding in the same interaction. Multiple-reference checkpoints require joint recovery of their references, which can reside within one component or across components.

Matched-state assembly. Goal-directed APIs are identified from the requested task effects; for question-answering tasks, the target is answer submission. We select the first attempted call to each target API within a task and merge target calls occurring in the same actor action. Incorrect calls and unsuccessful trajectories remain included. Selection precedes the sampling of either next action.

The trajectories yield 204 decisions in 155 tasks and 55 scenarios. Two decisions in one task do not reproduce identical prefix observations, leaving 202 verified pairs in 154 tasks. Each pair be-

gins with **identical interactions, maintained memory, application state, simulated time, and live execution variables**. One arm uses the full Dependency-Aware Context Assembler. The other uses Dependency-Agnostic Assembly (N6): dependency-protected records and mandatory state constraints become optional candidates, and dependency-specific priority beyond the soft token target is removed. The editor, acting prompt, and tools are unchanged. Each arm generates and executes its own next action, which may contain several tool calls.

After each arm executes its action, progress is scored by **whether at least one additional official task check passes**. Previously passed checks are also reevaluated; neither arm loses a passed check at these decisions. Full assembly advances 41 decisions that N6 does not, while N6 advances eight that full assembly does not. The remaining 153 pairs have the same binary progress outcome.

Table 19: Next-action progress under matched state. Differences are EvoState minus N6 in percentage points. Reference reuse is a subset of the goal-directed decisions.

Decisions	EvoState	N6	Difference [95% CI]
Goal-directed	167/202	134/202	16.3 [9.3, 23.6]
Reference reuse	96/118	76/118	16.9 [8.6, 26.5]

Evidence at reuse. Among the verified decisions, 118 in 97 tasks and 35 scenarios reuse 857 identifiers, contact details, or file references from earlier observations. Repeated uses of the same argument and value within one decision count once. **All 857 are available in both assembled inputs**, either as explicit values or usable variable references. The intervals from first observation to reuse are 1–3 interactions for 388 references, 4–7 for 271, 8–15 for 173, and 16–27 for 25. Within these decisions, 25 advance only with full assembly and five only with N6. Thus, the difference in subsequent progress persists when both inputs retain the audited references.

Configuration and inference. Both comparisons use GLM-5.1 at temperature zero with reasoning disabled and a 4,096-token soft target. Editor calls allow up to 4,096 output tokens; the seed-0 next-action comparison caps actor output at 1,024 tokens. Full editing makes 280 model requests totaling 2.238M tokens; N3 makes 253 requests totaling 1.969M. Across all 202 paired decisions, next-action requests consume 1.786M tokens under full assembly and 1.449M under N6, or 8.84K and 7.17K per request, including input and output. These are **costs per diagnostic request**; Table 1 reports complete-task expenditure.

Intervals use 50,000 percentile-bootstrap resamples of **complete scenario clusters**, retaining all checkpoints or decisions from each sampled scenario. Paired differences are tested by exchanging method labels jointly within each scenario and computing the exact two-sided sign-flip probability. The two primary comparisons are all editor checkpoints and all goal-directed decisions. Their Holm-adjusted p-values are 7.63×10^{-6} and 4.86×10^{-5}, respectively. Subset intervals use the same clustering procedure.

Table 20: Next-action outcomes at the 202 matched goal-directed decisions. Categories are assigned in the displayed order and sum to 202 for each arm.

Outcome	EvoState	N6
At least one official check advances	167	134
Execution error without progress	13	25
Answer submission without progress	2	3
Other action without immediate progress	20	40

C.4 Exploratory paired significance tests

Each EvoState–baseline comparison pairs the same task identities and run labels, including failures. Tests **exchange method labels jointly within each cluster**: an AppWorld scenario, BFCL conversation, or CRM task, retaining its three runs. There are 56, 139, 200, and 136 clusters for Normal, Challenge, BFCL, and CRM, respectively. CRM tasks remain within their organization/type strata.

For success, let d_c be the difference in the numbers of successful records in cluster c. We use the two-sided statistic $|\sum_c d_c|$ and independently flip each cluster's sign under the null. The exact sign-flip distribution includes ties. For tokens, d_c is the difference in total method tokens in that cluster; the same statistic is evaluated with 200,000 Monte Carlo sign-flip draws. The reported Monte Carlo probability is $(b+1)/(200{,}000+1)$, where b counts draws at least as extreme as observed. Tokens include unsuccessful trajectories and all recorded LLM calls for context management. All clusters within a setting have the same number of records, so these statistics are proportional to the corresponding mean differences.

The null assumes joint exchangeability of method labels within independent clusters, conditional on the evaluated records and fixed run labels. These tests are exploratory. **Holm adjustment jointly covers success and token tests** against EvoState within each setting: 16 hypotheses per setting, including both directions of token effects. Tables 21–24 report raw and adjusted probabilities.

Against Full History, **only BFCL passes the adjusted 0.05 threshold for success**; **Normal, Challenge, and CRM pass it for lower token expenditure**. BFCL's positive unadjusted token-reduction interval and nonsignificant adjusted token test reflect the distinct procedures described in Appendix B.3.

Table 21: AppWorld Normal: paired comparisons of EvoState with every baseline (56 clusters, 504 task–run pairs). Positive Δ SR and token cut favor EvoState. p_{H} is Holm-adjusted across both endpoints and all baselines; bold marks $p_{\mathrm{H}} < 0.05$. The tests are conditional and exploratory.

	Success			Tokens per task		
Baseline	Δ SR (pp)	Raw p	p_{H}	Cut (%)	Raw p	p_{H}
Full History	7.14	0.029	0.115	19.66	5.00×10^{-6}	$\mathbf{6.00 \times 10^{-5}}$
Sliding Window	21.23	1.19×10^{-7}	$\mathbf{1.55 \times 10^{-6}}$	1.08	0.773	0.773
Obs. Masking	11.90	1.03×10^{-4}	$\mathbf{5.15 \times 10^{-4}}$	22.13	5.00×10^{-6}	$\mathbf{6.00 \times 10^{-5}}$
Rolling Summary	12.50	5.86×10^{-5}	$\mathbf{3.51 \times 10^{-4}}$	-5.37	0.107	0.213
Retrieval Memory	32.34	2.62×10^{-10}	$\mathbf{3.93 \times 10^{-9}}$	8.75	0.051	0.153
HiAgent	18.25	1.05×10^{-5}	$\mathbf{8.37 \times 10^{-5}}$	16.82	3.00×10^{-5}	$\mathbf{2.10 \times 10^{-4}}$
ACON-History	25.79	6.80×10^{-9}	$\mathbf{9.51 \times 10^{-8}}$	-47.78	5.00×10^{-6}	$\mathbf{6.00 \times 10^{-5}}$
StructMem	50.20	1.14×10^{-13}	$\mathbf{1.82 \times 10^{-12}}$	51.61	5.00×10^{-6}	$\mathbf{6.00 \times 10^{-5}}$

Table 22: AppWorld Challenge: paired comparisons of EvoState with every baseline (139 clusters, 1,251 task–run pairs). Positive Δ SR and token cut favor EvoState. p_{H} is Holm-adjusted across both endpoints and all baselines; bold marks $p_{\mathrm{H}} < 0.05$. The tests are conditional and exploratory.

	Success			Tokens per task		
Baseline	Δ SR (pp)	Raw p	p_{H}	Cut (%)	Raw p	p_{H}
Full History	2.00	0.387	1.000	10.53	7.15×10^{-4}	**0.006**
Sliding Window	30.78	4.50×10^{-22}	$\mathbf{6.30 \times 10^{-21}}$	-3.77	0.135	0.674
Obs. Masking	6.31	0.007	**0.049**	-1.02	0.676	1.000
Rolling Summary	18.55	4.16×10^{-15}	$\mathbf{4.58 \times 10^{-14}}$	-2.13	0.346	1.000
Retrieval Memory	50.92	2.42×10^{-33}	$\mathbf{3.64 \times 10^{-32}}$	-1.16	0.737	1.000
HiAgent	25.26	4.48×10^{-18}	$\mathbf{5.38 \times 10^{-17}}$	5.24	0.014	0.086
ACON-History	32.13	5.08×10^{-22}	$\mathbf{6.60 \times 10^{-21}}$	-58.45	5.00×10^{-6}	$\mathbf{5.00 \times 10^{-5}}$
StructMem	56.43	3.69×10^{-35}	$\mathbf{5.90 \times 10^{-34}}$	40.63	5.00×10^{-6}	$\mathbf{5.00 \times 10^{-5}}$

Table 23: BFCL Long Context: paired comparisons of EvoState with every baseline (200 clusters, 600 task–run pairs). Positive Δ SR and token cut favor EvoState. p_{H} is Holm-adjusted across both endpoints and all baselines; bold marks $p_{\mathrm{H}} < 0.05$. The tests are conditional and exploratory.

	Success			Tokens per task		
Baseline	Δ SR (pp)	Raw p	p_{H}	Cut (%)	Raw p	p_{H}
Full History	5.33	0.011	**0.023**	11.22	0.059	0.059
Sliding Window	11.33	1.96×10^{-6}	$\mathbf{2.55 \times 10^{-5}}$	20.61	5.00×10^{-6}	$\mathbf{6.00 \times 10^{-5}}$
Obs. Masking	5.50	0.006	**0.018**	26.55	5.00×10^{-6}	$\mathbf{6.00 \times 10^{-5}}$
Rolling Summary	9.83	1.61×10^{-4}	$\mathbf{8.03 \times 10^{-4}}$	16.40	5.00×10^{-6}	$\mathbf{6.00 \times 10^{-5}}$
Retrieval Memory	13.83	1.98×10^{-8}	$\mathbf{2.78 \times 10^{-7}}$	10.65	0.003	**0.013**
HiAgent	24.50	7.19×10^{-15}	$\mathbf{1.08 \times 10^{-13}}$	20.12	5.00×10^{-6}	$\mathbf{6.00 \times 10^{-5}}$
ACON-History	10.50	1.19×10^{-5}	$\mathbf{7.16 \times 10^{-5}}$	-31.83	5.00×10^{-6}	$\mathbf{6.00 \times 10^{-5}}$
StructMem	32.33	3.02×10^{-26}	$\mathbf{4.84 \times 10^{-25}}$	46.95	5.00×10^{-6}	$\mathbf{6.00 \times 10^{-5}}$

Table 24: CRM exact-match: paired comparisons of EvoState with every baseline (136 clusters, 408 task–run pairs). Positive Δ SR and token cut favor EvoState. p_{H} is Holm-adjusted across both endpoints and all baselines; bold marks $p_{\mathrm{H}} < 0.05$. The tests are conditional and exploratory.

	Success			Tokens per task		
Baseline	Δ SR (pp)	Raw p	p_{H}	Cut (%)	Raw p	p_{H}
Full History	4.17	0.156	0.469	29.40	7.50×10^{-5}	**0.001**
Sliding Window	3.92	0.175	0.469	31.38	2.00×10^{-4}	**0.002**
Obs. Masking	5.39	0.059	0.234	26.89	4.40×10^{-4}	**0.004**
Rolling Summary	1.72	0.497	0.497	23.48	2.85×10^{-4}	**0.003**
Retrieval Memory	9.56	2.25×10^{-4}	**0.002**	32.92	5.00×10^{-6}	$\mathbf{8.00 \times 10^{-5}}$
HiAgent	7.35	0.007	0.054	26.47	9.00×10^{-5}	**0.001**
ACON-History	6.62	0.017	0.103	13.22	0.025	0.126
StructMem	7.84	0.009	0.062	53.25	5.00×10^{-6}	$\mathbf{8.00 \times 10^{-5}}$

D COMPONENT ABLATIONS

D.1 VARIANT DEFINITIONS AND MEASUREMENTS

The six variants and full-method reference each use all 168 AppWorld Normal tasks in 56 scenarios for run labels 0, 1, and 2, giving 504 task–run records per method.

Interventions. N1 disables Task State writes, reads, and derived task-state recovery. Current Trace exact dependencies and independently evidence-supported events remain available. N2 disables event creation and reading, including generation calls made solely for an event; Current Trace, Task State, and completion control remain. Source records are retained when no authorized event replacement exists.

N3 restricts each logical update batch to one of Current Trace, Event Memory, and Task State. It permits multiple compatible operations within the chosen component; closure, supported state changes, and local updates determine routing. There is no deferred queue that later restores updates to other components. N4 disables LLM-based memory consolidation both within work units and at their boundaries, while retaining rule-based updates across components, events constructed without an LLM call, and the original assembler.

N5 disables explicit invalidation, implicit invalidation archiving, and same-key cleanup that would bypass this intervention. Current slots can still receive new values, while older exact candidates remain eligible. **N6 retains state storage and editing** but demotes dependency-protected records and accumulated mandatory state constraints to optional candidates. It removes the dependency-specific bypass of the assembly soft target while retaining ordinary relevance/recency selection and lossless rendering of selected records requiring exact preservation.

Table 25: Component ablations on AppWorld Normal, with 168 tasks per run. Success and tokens per task are means ± sample standard deviations across three run labels. Differences are relative to the full-method reference. Token totals include input and output tokens from acting and context-management calls on successful and failed tasks.

Component	Variant	Success (%) ↑	Difference (pp) ↑	Tokens/task (K) ↓
Full method	EvoState reference	**86.11 ± 0.34**	—	138.67 ± 6.68
State representation	N1: w/o Task State	82.34 ± 0.91	−3.77	136.78 ± 3.34
	N2: w/o Event Memory	81.15 ± 1.50	−4.96	139.49 ± 1.67
Context editing	N3: Single-Role Editing	78.77 ± 1.50	−7.34	137.22 ± 2.17
	N4: w/o LLM-based consolidation	81.35 ± 0.34	−4.76	132.39 ± 7.11
	N5: w/o Explicit Invalidation	80.56 ± 2.68	−5.56	150.00 ± 2.31
Context assembly	N6: Dependency-Agnostic Assembly	66.67 ± 1.57	−19.44	150.42 ± 1.82

Configuration. Acting and consolidation use GLM-5.1 at temperature zero with reasoning disabled. Actor requests carry run labels 0, 1, and 2; editor calls omit an explicit seed. Limits are 204,800 tokens of capacity, an 8,192-token output reserve, a 4,096-token assembly soft target, 40 actor interactions, and a 1,800-second timeout. N4 makes no consolidation calls.

Metrics. Tables 27–30 report means and sample standard deviations across three runs, using the official task and scenario criteria. **K/success is computed within each run** from all recorded costs, including failures, before summarizing across runs; Table 1 instead uses the pooled ratio.

Token totals include acting and context-management input and output.

Figure 9 shows differences in three-run mean success. N5 losses concentrate in Medium and Hard, while N6 degrades across all three groups. Table 29 gives the means and sample standard deviations.

Table 26: Full History with and without work-unit prompting on AppWorld Normal. Run columns give successful tasks; aggregate metrics show means ± sample SD over three runs.

Method	Run 0	Run 1	Run 2	Success (%) ↑	Tokens/task (K) ↓	Tokens/success (K) ↓
Full History	132/168	135/168	131/168	78.97 ± 1.24	172.60 ± 13.78	218.75 ± 20.19
Full History + work units	126/168	128/168	131/168	76.39 ± 1.50	149.38 ± 10.10	195.53 ± 11.94

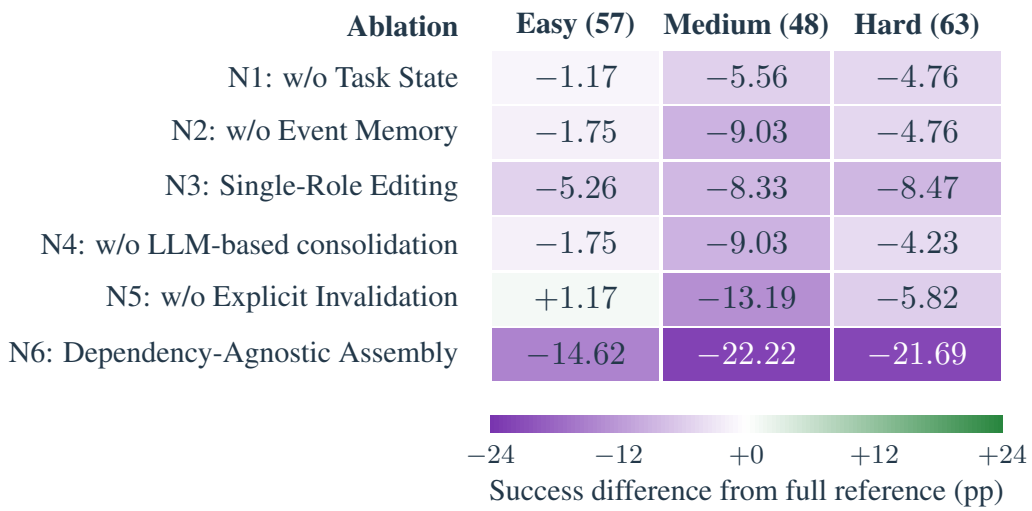


Figure 9: Difficulty-specific success changes from full EvoState, in percentage points. Purple denotes lower success and green higher success on a common zero-centered scale.

Table 27: Additional Normal ablation metrics over three runs (168 tasks per run). Success counts are pooled across runs; scenario goal completion (SGC) and recorded tokens per success (K) show mean ± sample SD. Table 25 gives task-success rates and tokens per task.

Variant	Successes/ task–runs	SGC (%) ↑	Tokens/success (K) ↓
Full reference	434/504	77.98 ± 1.03	161.05 ± 8.12
N1	415/504	67.86 ± 1.79	166.10 ± 2.22
N2	409/504	68.45 ± 2.73	171.94 ± 4.28
N3	397/504	60.12 ± 4.12	174.21 ± 1.22
N4	410/504	68.45 ± 3.72	162.72 ± 8.27
N5	406/504	67.86 ± 4.72	186.38 ± 8.24
N6	336/504	42.26 ± 6.76	225.76 ± 7.93

Interpretation. **All six variants lower mean success**. N6 degrades across all difficulty groups and increases mean actor interactions from 16.50 to 20.64. N5 loses 13.19 and 5.82 success points on Medium and Hard while gaining 1.17 on Easy.

Table 28: Normal ablations by run label. Each row contains all 168 planned tasks. All token quantities are recorded usage, including failed trajectories.

Variant	Run	Successes/ tasks	Success (%) ↑	SGC (%) ↑	Total tokens (M) ↓	K/task ↓	K/success ↓
Full reference	0	145/168	86.31	76.79	22.002	130.96	151.74
Full reference	1	145/168	86.31	78.57	23.882	142.15	164.70
Full reference	2	144/168	85.71	78.57	24.004	142.88	166.70
N1	0	140/168	83.33	69.64	23.584	140.38	168.46
N1	1	138/168	82.14	67.86	22.882	136.20	165.81
N1	2	137/168	81.55	66.07	22.473	133.77	164.04
N2	0	136/168	80.95	71.43	23.750	141.37	174.63
N2	1	134/168	79.76	66.07	23.340	138.93	174.18
N2	2	139/168	82.74	67.86	23.213	138.17	167.00
N3	0	132/168	78.57	55.36	22.862	136.08	173.20
N3	1	130/168	77.38	62.50	22.823	135.85	175.56
N3	2	135/168	80.36	62.50	23.472	139.72	173.87
N4	0	137/168	81.55	64.29	21.873	130.20	159.66
N4	1	137/168	81.55	71.43	23.576	140.33	172.09
N4	2	136/168	80.95	69.64	21.274	126.63	156.43
N5	0	131/168	77.98	64.29	25.615	152.47	195.54
N5	1	140/168	83.33	73.21	25.140	149.64	179.57
N5	2	135/168	80.36	66.07	24.845	147.89	184.04
N6	0	115/168	68.45	50.00	24.917	148.32	216.67
N6	1	111/168	66.07	39.29	25.455	151.52	229.33
N6	2	110/168	65.48	37.50	25.439	151.43	231.27

Table 29: Normal ablations by official difficulty: mean ± sample SD across three run labels. Every run has 57 Easy, 48 Medium, and 63 Hard tasks. Cost is recorded K/task, with accounting consistent with Table 27.

Variant	Easy success (%) ↑	Easy K/task ↓	Medium success (%) ↑	Medium K/task ↓	Hard success (%) ↑	Hard K/task ↓
Full reference	95.32 ± 1.01	66.18 ± 1.98	93.06 ± 1.20	121.03 ± 6.45	72.49 ± 0.92	217.69 ± 13.00
N1	94.15 ± 2.68	63.07 ± 2.41	87.50 ± 5.51	128.15 ± 5.16	67.72 ± 1.83	210.06 ± 4.99
N2	93.57 ± 1.01	62.59 ± 0.69	84.03 ± 3.18	124.31 ± 9.65	67.72 ± 1.83	220.64 ± 2.56
N3	90.06 ± 4.05	68.54 ± 0.87	84.72 ± 1.20	131.38 ± 2.08	64.02 ± 0.92	203.80 ± 6.36
N4	93.57 ± 1.01	66.62 ± 4.38	84.03 ± 3.18	117.05 ± 10.50	68.25 ± 1.59	203.57 ± 7.86
N5	96.49 ± 3.51	70.43 ± 0.86	79.86 ± 5.24	132.64 ± 1.72	66.67 ± 3.17	235.23 ± 4.52
N6	80.70 ± 0.00	77.79 ± 3.84	70.83 ± 7.51	136.67 ± 4.14	50.79 ± 1.59	226.61 ± 5.62

Table 30: Normal ablation costs and execution counts: mean ± sample SD across three run labels. Actor and memory tokens are recorded K/task; memory tokens include input and output of all context-management LLM calls. Model calls include failed requests. N4 retains rule-based updates without context-management LLM calls.

Variant	Actor input (K) ↓	Actor output (K) ↓	Memory tokens (K) ↓	Actor interactions ↓	Model calls ↓
Full reference	121.87 ± 5.76	1.90 ± 0.10	14.90 ± 0.82	16.50 ± 0.40	18.32 ± 0.51
N1	120.74 ± 3.57	1.82 ± 0.08	14.22 ± 0.20	16.76 ± 0.32	18.54 ± 0.27
N2	127.53 ± 1.77	1.94 ± 0.04	10.02 ± 0.11	17.26 ± 0.28	18.51 ± 0.27
N3	121.82 ± 2.01	2.09 ± 0.09	13.31 ± 0.28	17.26 ± 0.26	18.91 ± 0.28
N4	130.52 ± 7.00	1.87 ± 0.12	0.00 ± 0.00	16.89 ± 0.57	17.14 ± 0.58
N5	133.64 ± 2.29	1.84 ± 0.02	14.53 ± 0.23	17.35 ± 0.34	19.50 ± 0.34
N6	129.56 ± 1.78	2.15 ± 0.01	18.71 ± 0.13	20.64 ± 0.23	22.91 ± 0.29

E CROSS-BACKBONE EVALUATION

E.1 MODEL IDENTITIES AND INFERENCE SETTINGS

DeepSeek-V4-Flash (DeepSeek-AI, 2026) is used for acting and all context-management LLM calls, with actor run label 0, temperature zero, and reasoning disabled. Every method has a 204,800-token capacity. Other method settings follow Table 5, and interaction, output, and timeout limits follow Appendix B.2. The environment and official success criteria are unchanged. Costs include input and output from acting and context management. There is **one run per task**, so no cross-run standard deviation is reported.

E.2 WITHIN-BACKBONE RESULTS

Normal, Challenge, and BFCL each include EvoState, Full History, HiAgent, Observation Masking, and ACON-History on the complete task set.

Metrics and findings. Tables 31–33 give success, SGC, and cost using the definitions in Appendix B.3. Successes per million tokens is the reciprocal of pooled tokens per success, scaled by one million. Relative to Full History, EvoState reduces K/success by 24.95%, 32.42%, and 46.66% on Normal, Challenge, and BFCL, respectively.

Table 31: AppWorld Normal: complete single-run results (run label 0; 168 planned tasks per method). Tokens include recorded input and output from acting and context-management calls on successful and failed tasks.

Method	Successes/ tasks	Success (%) ↑	SGC (%) ↑	Total tokens (M) ↓	K/task ↓	K/success ↓	Successes/M tokens ↑
EvoState	129/168	76.79	57.14	34.955	208.07	270.97	3.690
Full History	118/168	70.24	48.21	42.604	253.59	361.05	2.770
HiAgent	80/168	47.62	28.57	38.147	227.06	476.83	2.097
Observation Masking	102/168	60.71	37.50	38.923	231.68	381.59	2.621
ACON-History	48/168	28.57	10.71	23.269	138.51	484.78	2.063

Table 32: AppWorld Challenge: complete single-run results (run label 0; 417 planned tasks per method). Tokens include recorded input and output from acting and context-management calls on successful and failed tasks.

Method	Successes/ tasks	Success (%) ↑	SGC (%) ↑	Total tokens (M) ↓	K/task ↓	K/success ↓	Successes/M tokens ↑
EvoState	211/417	50.60	38.13	121.770	292.01	577.11	1.733
Full History	200/417	47.96	29.50	170.793	409.57	853.96	1.171
HiAgent	109/417	26.14	14.39	131.899	316.30	1210.08	0.826
Observation Masking	172/417	41.25	20.86	135.754	325.55	789.27	1.267
ACON-History	62/417	14.87	6.47	74.092	177.68	1195.03	0.837

Table 33: BFCL Long Context: complete single-run results (run label 0; 200 planned tasks per method). Tokens include recorded input and output from acting and context-management calls on successful and failed tasks.

Method	Successes/ tasks	Success (%) ↑	Total tokens (M) ↓	K/task ↓	K/success ↓	Successes/M tokens ↑
EvoState	120/200	60.00	25.169	125.84	209.74	4.768
Full History	96/200	48.00	37.749	188.75	393.22	2.543
HiAgent	114/200	57.00	33.372	166.86	292.74	3.416
Observation Masking	106/200	53.00	32.209	161.04	303.85	3.291
ACON-History	105/200	52.50	19.581	97.90	186.48	5.362

Table 34: DeepSeek actor evaluation by official AppWorld difficulty, run label 0. All fixed-denominator tasks are retained.

Method	Easy success (%) ↑	Easy K/task ↓	Medium success (%) ↑	Medium K/task ↓	Hard success (%) ↑	Hard K/task ↓
AppWorld Normal: Easy 57 / Medium 48 / Hard 63						
EvoState	94.74	114.86	81.25	210.78	57.14	290.33
Full History	78.95	116.72	72.92	223.92	60.32	400.05
HiAgent	77.19	127.70	47.92	226.26	20.63	317.57
Observation Masking	80.70	128.01	64.58	217.67	39.68	336.16
ACON-History	50.88	98.28	8.33	141.58	23.81	172.57
AppWorld Challenge: Easy 72 / Medium 150 / Hard 195						
EvoState	79.17	146.35	50.67	285.46	40.00	350.84
Full History	66.67	178.94	43.33	404.69	44.62	498.49
HiAgent	59.72	168.77	22.00	311.11	16.92	374.77
Observation Masking	69.44	162.20	38.00	339.70	33.33	374.97
ACON-History	34.72	118.18	12.67	178.95	9.23	198.67

Table 35: DeepSeek evaluation: recorded token costs per task (K) and mean execution counts, run label 0. Memory tokens include input and output of all context-management LLM calls and are included in total method cost. Model assignments appear in Appendix E.1.

Method	Actor input (K) ↓	Actor output (K) ↓	Memory tokens (K) ↓	Actor interactions ↓	Model calls ↓
AppWorld Normal					
EvoState	184.20	2.65	21.21	22.88	25.36
Full History	250.99	2.60	0.00	21.69	21.69
HiAgent	211.80	2.84	12.42	27.57	29.83
Observation Masking	228.59	3.09	0.00	25.00	25.01
ACON-History	121.27	2.11	15.13	28.27	30.86
AppWorld Challenge					
EvoState	259.40	3.15	29.46	29.54	33.11
Full History	406.64	2.94	0.00	26.35	26.36
HiAgent	293.75	3.19	19.36	32.29	69.71
Observation Masking	322.40	3.15	0.00	28.78	56.68
ACON-History	146.56	2.41	28.72	32.35	72.49
BFCL Long Context					
EvoState	119.02	1.84	4.98	13.01	13.72
Full History	186.72	2.02	0.00	13.22	13.23
HiAgent	164.41	1.84	0.61	12.20	12.77
Observation Masking	159.31	1.73	0.00	12.07	12.08
ACON-History	82.68	1.86	13.36	13.05	14.22

EvoState's Hard success is lower than Full History despite lower K/task: 57.14% versus 60.32% on Normal and 40.00% versus 44.62% on Challenge. ACON-History uses fewer tokens per task on all three settings and has lower BFCL K/success.

F Execution and cost measurements

The cost tables separate actor input, actor output, and context-management LLM tokens on all trajectories. Figure 10 shows that **actor-input savings exceed added context-management costs** on Challenge and CRM. Memory tokens include context-management input and output; K denotes 1,000 tokens. CRM simulator costs are separate, and offline initial-message generation is excluded. Table 36 gives total system tokens across the three runs. Comparisons of total cost appear in Figure 7.

Table 36: Total system tokens across three runs with GLM-5.1, including acting and context-management input and output.

Method	AppWorld Normal	AppWorld Challenge	BFCL Long Context	CRMArena-Pro
Full History	86,988,655	293,329,036	62,450,989	37,440,606
Sliding Window	70,649,855	252,901,759	69,841,261	38,519,485
Observation Masking	89,751,014	259,788,344	75,491,846	36,156,830
Rolling Summary	66,326,218	256,967,834	66,322,003	34,543,009
Retrieval Memory	76,592,246	259,414,613	62,050,567	39,406,450
HiAgent	84,018,970	276,937,679	69,411,608	35,947,709
ACON-History	47,293,018	165,620,680	42,057,464	30,459,814
StructMem	144,416,828	442,001,717	104,519,803	56,537,889
EvoState	69,888,167	262,433,879	55,445,071	26,433,638

Table 37: AppWorld Normal: token costs (K/task) and interactions, mean ± sample SD.

Method	Actor in	Actor out	Memory tokens	Interactions
Full History	170.46 ± 13.64	2.14 ± 0.15	0.00 ± 0.00	17.21 ± 1.11
Sliding Window	137.99 ± 1.97	2.19 ± 0.08	0.00 ± 0.00	20.79 ± 0.39
Observation Masking	175.35 ± 2.62	2.73 ± 0.05	0.00 ± 0.00	21.46 ± 0.24
Rolling Summary	101.14 ± 3.25	2.04 ± 0.07	28.42 ± 1.65	17.24 ± 0.46
Retrieval Memory	149.44 ± 4.07	2.53 ± 0.12	0.00 ± 0.00	25.01 ± 0.31
HiAgent	152.84 ± 5.96	2.72 ± 0.20	11.14 ± 0.52	21.35 ± 0.47
ACON-History	82.31 ± 1.14	1.60 ± 0.05	9.92 ± 0.37	20.29 ± 0.20
StructMem	147.83 ± 5.53	2.84 ± 0.21	135.88 ± 5.18	30.04 ± 1.26
EvoState	121.87 ± 5.76	1.90 ± 0.10	14.90 ± 0.82	16.50 ± 0.40

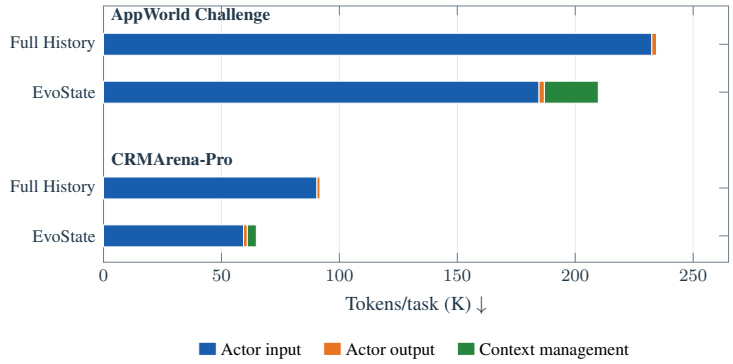


Figure 10: Actor-input, actor-output, and context-management tokens on Challenge and CRM. Bars show three-run means per task, including failures and all context-management LLM calls.

Table 38: AppWorld Challenge: token costs (K/task) and interactions, mean ± sample SD.

Method	Actor in	Actor out	Memory tokens	Interactions
Full History	232.40 ± 2.29	2.08 ± 0.02	0.00 ± 0.00	19.52 ± 0.18
Sliding Window	199.73 ± 1.35	2.43 ± 0.04	0.00 ± 0.00	27.32 ± 0.32
Observation Masking	205.31 ± 4.27	2.35 ± 0.05	0.00 ± 0.00	21.01 ± 0.27
Rolling Summary	148.22 ± 0.80	2.58 ± 0.05	54.61 ± 0.45	25.47 ± 0.24
Retrieval Memory	204.52 ± 5.20	2.84 ± 0.03	0.00 ± 0.00	33.79 ± 0.29
HiAgent	202.87 ± 1.69	2.77 ± 0.02	15.74 ± 0.18	26.10 ± 0.25
ACON-History	115.82 ± 1.24	1.83 ± 0.01	14.74 ± 0.25	28.13 ± 0.31
StructMem	186.19 ± 0.44	2.89 ± 0.06	164.24 ± 0.24	35.03 ± 0.09
EvoState	184.56 ± 1.80	2.33 ± 0.06	22.89 ± 0.04	22.51 ± 0.19

Table 39: BFCL Long Context: token costs (K/task) and interactions, mean ± sample SD.

Method	Actor in	Actor out	Memory tokens	Interactions
Full History	103.14 ± 1.05	0.94 ± 0.00	0.00 ± 0.00	11.26 ± 0.14
Sliding Window	115.33 ± 1.86	1.08 ± 0.01	0.00 ± 0.00	11.65 ± 0.08
Observation Masking	124.77 ± 0.98	1.05 ± 0.00	0.00 ± 0.00	11.11 ± 0.05
Rolling Summary	97.30 ± 2.53	1.10 ± 0.03	12.14 ± 0.37	11.16 ± 0.21
Retrieval Memory	102.35 ± 1.74	1.06 ± 0.01	0.00 ± 0.00	11.64 ± 0.23
HiAgent	106.70 ± 1.03	1.09 ± 0.01	7.90 ± 0.45	11.64 ± 0.13
ACON-History	61.50 ± 0.21	1.00 ± 0.01	7.60 ± 0.25	10.79 ± 0.08
StructMem	103.85 ± 2.32	1.12 ± 0.02	69.23 ± 1.74	12.49 ± 0.18
EvoState	87.35 ± 0.30	1.01 ± 0.00	4.05 ± 0.20	11.40 ± 0.03

Table 40: CRM exact-match: token costs (K/task) and interactions, mean ± sample SD. Totals include simulator costs and exclude offline initial-message generation.

Method	Actor in (K) ↓	Actor out (K) ↓	Memory tokens (K) ↓	Interactions ↓	Simulator (K) ↓	Method + sim. (K) ↓
Full History	90.42±3.75	1.35±0.03	0.00±0.00	7.58±0.19	2.47±0.05	94.24±3.81
Sliding Window	93.00±2.13	1.41±0.11	0.00±0.00	8.46±0.32	2.30±0.13	96.71±2.34
Obs. Masking	87.31±5.53	1.31±0.15	0.00±0.00	7.59±0.34	2.31±0.08	90.93±5.73
Rolling Summary	76.60±10.93	1.40±0.15	6.67±2.05	7.89±0.62	2.45±0.12	87.12±13.06
Retrieval Memory	95.04±2.84	1.55±0.17	0.00±0.00	9.15±0.50	2.17±0.06	98.75±2.80
HiAgent	83.30±11.48	1.47±0.16	3.34±1.55	7.77±0.56	2.37±0.09	90.47±13.22
ACON-History	60.51±3.40	1.31±0.06	12.84±2.28	8.71±0.34	2.41±0.05	77.06±5.59
StructMem	82.56±3.26	1.49±0.13	54.52±2.62	9.25±0.56	2.46±0.13	141.03±6.11
EvoState	59.35±3.15	1.56±0.10	3.88±0.68	8.61±0.26	2.38±0.13	67.17±3.95

G QUALITATIVE TRAJECTORIES

This section presents seven AppWorld train/dev cases illustrating complementary forms of context maintenance. All use GLM-5.1 at temperature zero with reasoning disabled. The cases complement the test-set evaluation of success and cost. The first expands Figure 3; the second presents a complete memory-editing cycle.

Panels reproduce task instructions, executed programs, responses, and accepted memory. Narrative interactions are one-based; quoted indices retain their original notation. Formatting is adjusted, authentication literals are redacted, and omissions are marked or identified as excerpts. Gray panels show execution content and blue panels maintained memory. Actor computations and consolidation of their observed results are distinguished throughout.

Context requirement	**Illustrative scenarios**	**Section**
Evidence reuse across work units	Recover an owed amount from messages, repay it, and notify the recipient using retained contact and login evidence.	G.1
Multi-operation editing	Follow artists of classical songs: state insertion, local retention, LLM-based summarization, and removal in one update.	G.2
Organizing evidence	Complete a paginated collection; preserve album groups for later comparison.	G.3
Ordered effects	Export a complete music backup before deleting the account.	G.4
Exact identity across work units	Use newly renamed paths for later file moves.	G.5
Scope and remaining work	Separate completed library removals from pending playlist removals.	G.6

G.1 EVIDENCE REUSE ACROSS WORK UNITS

AppWorld dev, task `530b157_1`. The trajectory passes all ten official checks in 24 actor interactions. Contact details below belong to the benchmark environment.

User request

Kristin paid for my grocery recently as my payment cards were not working at the time. Send them the owed money with a description note "Groceries" as per my phone text conversation, and then send them a phone text message, "It is done.".

Conversation evidence and Event Memory. After correcting unsuccessful queries, the agent finds the amount and payment channel in Kristin's messages. Event Memory retains the discovered debt as reusable evidence with a progress outcome; payment remains pending.

1. Retrieve the conversation (interaction 13)

Executed call

```
print(apis.phone.show_text_message_window(
    access_token=login_result["access_token"],
    phone_number="6017026518",
    min_datetime="2023-05-17T13:00:00",
    max_datetime="2023-05-17T14:00:00"))
```

Observed messages: selected sender and message fields

```
{"sender": {"name": "Matthew Blackburn"},
 "message": "hey, how much was yesterday's grocery?"}
{"sender": {"name": "Kristin White"},
 "message": "It was $54."}
{"sender": {"name": "Matthew Blackburn"},
 "message": "cool, I'll send it to you on venmo."}
{"sender": {"name": "Kristin White"},
 "message": "great, thanks!"}
```

Event Memory after the retrieval work unit: selected fields

```
"objective": "Find Kristin's contact and text conversation to determine
owed amount",
"outcome": "progress",
"reusable_outputs": {
    "contact_id": 824,
    "owed_amount": 54,
    "owed_to": "Kristin White",
    "phone_number": "6017026518"
}
```

Payment evidence and Task State. The agent sends the observed amount with the requested note. The returned transaction identifier **becomes an observed fact in Task State**, while notification remains pending.

2. Repay the debt (interaction 22)

Executed call

```
print(apis.venmo.create_transaction(
    access_token=venmo_login["access_token"],
    receiver_email="kri-powe@gmail.com",
    amount=54,
    description="Groceries"
))
```

Observed response


```
{"message": "Sent money.", "transaction_id": 8216}
```


Task State: selected fields of the resulting fact


```
"key": "transaction_id", "value": 8216,
"evidence_status": "observed", "status": "active"
```


Reusing earlier evidence for the final action. The notification call **reuses the earlier phone number and login binding** across the intervening payment unit and consolidation. Its returned identifier is stored separately from the transaction identifier.

Before interaction 23, the model input retains the Phone login result from interaction 7, the account-credential lookup result from interaction 16, and the Venmo login result from interaction 19, **all marked for exact preservation**.

Current Trace before notification: fields from the earlier contact observation


```
"contact_id": 824,
"first_name": "Kristin", "last_name": "White",
"email": "kri-powe@gmail.com", "phone_number": "6017026518"
```


3. Notify Kristin (interaction 23)

Executed call

```
print(apis.phone.send_text_message(
    access_token=login_result["access_token"],
    phone_number="6017026518",
    message="It is done."
))
```

Observed response


```
{"message": "Text message sent.", "text_message_id": 16809}
```


Task State: selected fields of the resulting fact


```
"key": "text_message_id", "value": 16809,
"evidence_status": "observed", "status": "active"
```


The task completes at interaction 24 after three consolidation calls. The sequence connects a reusable event, updated action identifiers, and retained contact and login evidence to subsequent execution.

G.2 A COMPLETE MULTI-OPERATION EDITING CYCLE

G.2.1 FROM PLAYLISTS TO THE ARTISTS OF CLASSICAL SONGS

AppWorld train, task b7a9ee9_1. The trajectory passes all 4 official checks in 12 actor interactions.

Task instruction

Follow all artists of all classical-genre songs in any of my playlists on Spotify.

This case shows **one complete maintenance cycle**, from the playlist observation through accepted edits to the next input. The observation supplies a state value and remains in Current Trace while older records are condensed.

Interactions	Observed task progression
1–2	Inspect Spotify operations and the playlist, song, and artist-following interfaces.
3–5	Obtain account credentials, inspect authentication, and establish the Spotify session.
6	Retrieve all eight playlists; apply the five accepted edits detailed below.
7–8	Inspect song details and deduplicate the playlist contents into 53 song identities.
9–10	Inspect those songs, identify five artists of classical songs, and confirm the follow operation.
11–12	Follow all five artists, observe five confirmations, and submit completion.

Executed action, interaction 6

```
access_token = login_result["access_token"]
playlists =
  apis.spotify.show_playlist_library(access_token=access_token,
  page_limit=20, page_index=0)
print(playlists)
```

playlist_id	Observed song_ids (all eight lists)
366	[43, 62, 149, 216, 236, 243, 278, 290]
365	[50, 52, 84, 266, 283]
371	[25, 30, 100, 112, 121, 128, 139, 177, 256, 267]
370	[72, 108, 135, 149, 151, 216]
369	[49, 82, 96, 104, 188, 257, 290]
368	[21, 31, 97, 152, 206, 209, 266, 291]
367	[15, 37, 39, 66, 90, 136, 210, 222, 229]
372	[82, 171, 222, 235, 286, 320]

The accepted script contains **five operations of four types**. The first three are rule-based updates; the final two are LLM-proposed edits included in the same update. Before this interaction, Task State has no song_ids fact. Afterwards, that observed fact contains all eight lists above, preserving their values and grouping.

One observation, five memory edits

Accepted operation	Content and scope
UPSERT_STATE	Write song_ids as the eight observed lists; evidence_status = observed, status = active.
KEEP_LOCAL	Retain the action that obtained the playlists, including the access_token binding.
KEEP_LOCAL	Retain the complete playlist observation in Current Trace.
COMPRESS_LOCAL	Replace five older records with the LLM-generated summary quoted below: the application-list observation (interaction 1), documentation action and observation (2), authentication-documentation action (4), and login observation (5).
EVICT	Remove the superseded application-list action from interaction 1.

One observation, five memory edits (continued)

Accepted LLM-generated summary (complete text)

Spotify login completed (step 4); access_token obtained. API docs for show_playlist_library, show_playlist, show_song, follow_artist reviewed (step 1). show_playlist_library called (step 5) and returned all 8 playlists (IDs 366, 365, 371, 370, 369, 368, 367, 372) owned by Jason Simpson; song_ids extracted to authoritative state. Collection is terminal (8 results, page_limit=20).

The eight returned playlists under a page limit of twenty support the summary's terminal-collection statement. The actor then performs genre filtering and deduplication.

Memory before	Accepted change	Memory after
No song_ids fact	Add the observed song_ids fact	All eight song lists in Task State
Five older local records	LLM-based memory consolidation	One summary of earlier records
Application-list action	Removal	Absent from Current Trace
Exact login dependency	Preserved despite being cited as source evidence	The same exact dependency remains available
Current action and observation	Two local insertions	Both retained with their relationship intact
12 Current Trace records	Five compressed, one removed, three added	9 Current Trace records

The LLM-proposed edit **cites the exact login record as a source but does not replace it**. That record remains available after the broader login observation is summarized. The next model input contains both the resulting summary and the song_ids fact in Task State, so these memory updates support the subsequent action.

Executed action, interaction 8

```
all_song_ids = [[43, 62, 149, 216, 236, 243, 278, 290], [50, 52, 84,
  266, 283], [25, 30, 100, 112, 121, 128, 139, 177, 256, 267], [72,
  108, 135, 149, 151, 216], [49, 82, 96, 104, 188, 257, 290], [21, 31,
  97, 152, 206, 209, 266, 291], [15, 37, 39, 66, 90, 136, 210, 222,
  229], [82, 171, 222, 235, 286, 320]]
unique_song_ids = list(set(sid for sublist in all_song_ids for sid in
  sublist))
print(f"Total unique song IDs: {len(unique_song_ids)}")
print(f"Song IDs: {sorted(unique_song_ids)}")
```

Observed response, interaction 8

```
Total unique song IDs: 53
```

The actor retrieves the song details and filters by classical genre at interaction 9, producing the following artist identities.

Observed response, interaction 9

```
Classical song found: Lost in the Twilight of Hope (ID: 82), Genre:
  classical
   Artist: Felix Blackwood (ID: 33)
Classical song found: Momentary Bliss (ID: 139), Genre: classical
  Artist: Zoe Mitchell (ID: 10)
Classical song found: Love's Final Resting Place (ID: 151), Genre:
  classical
  Artist: Mia Sullivan (ID: 12)
Classical song found: The Enigma of Love (ID: 152), Genre: classical
  Artist: Mia Sullivan (ID: 12)
Classical song found: The Paradox of Our Love (ID: 216), Genre:
  classical
  Artist: Oceanic Odyssey (ID: 21)
```

Observed response, interaction 9 (continued)

```
Classical song found: Within the Walls of Solitude (ID: 266), Genre:
  classical
   Artist: Evelyn Rose (ID: 28)
Classical song found: The Maze of Forgotten Whispers (ID: 267), Genre:
  classical
   Artist: Evelyn Rose (ID: 28)

Total classical artists to follow: 7
Unique artist IDs: [33, 10, 12, 21, 28]
```

Executed action, interaction 11

```
unique_artist_ids = [33, 10, 12, 21, 28]
for artist_id in unique_artist_ids:
    result = apis.spotify.follow_artist(artist_id=artist_id,
      access_token=access_token)
    print(f"Followed artist ID {artist_id}: {result}")
```

Observed response, interaction 11

```
Followed artist ID 33: {'message': 'Artist followed.'}
Followed artist ID 10: {'message': 'Artist followed.'}
Followed artist ID 12: {'message': 'Artist followed.'}
Followed artist ID 21: {'message': 'Artist followed.'}
Followed artist ID 28: {'message': 'Artist followed.'}
```

Completion is submitted at interaction 12, after all five artist-following calls are confirmed.

G.3 Organizing evidence and tracking remaining work

G.3.1 Keeping an incomplete collection open across pagination

AppWorld train, task aa8502b_1. The trajectory passes all 4 official checks in 12 actor interactions.

Task instruction
Follow all the artists who have sung at least one song I have liked on Spotify.

After a full first page, the summary **leaves collection coverage incomplete**. The actor retrieves the next page and an empty terminal page before constructing the artist set.

Accepted LLM-generated summary after interaction 5 (complete text)

Spotify API docs: show_liked_songs (GET /spotify/liked_songs, requires access_token; optional page_index default 0, page_limit default 5 max 20, sort_by default -liked_at; response has song_id, title, album_id, album_title, duration, artists[{id, name}], liked_at). follow_artist (POST /spotify/following_artists/{artist_id}, requires artist_id, access_token). Login obtained access_token. Page 0 of liked songs (page_limit=20) returned 20 rows; artist IDs observed: 2,4,5,6,7,9,11,12,13,14,15,27. Pagination incomplete—must fetch remaining pages to collect all artist IDs.

Executed action, interaction 6

```
liked_songs_page2 =
  apis.spotify.show_liked_songs(access_token=access_token,
  page_limit=20, page_index=1)
print(liked_songs_page2)
```

Executed action, interaction 7

```
liked_songs_page3 =
  apis.spotify.show_liked_songs(access_token=access_token,
  page_limit=20, page_index=2)
print(liked_songs_page3)
```

Observed response, interaction 7

```
[]
```

Observed response, interaction 9

```
Unique artist IDs to follow: [1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
  14, 15, 16, 22, 27, 30, 33, 34]
Total unique artists: 20
```

Event Memory after artist extraction: selected reusable outputs

```
{
  "artist_ids": [1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
    22, 27, 30, 33, 34],
  "total_unique_artists": 20
}
```

Executed action, interaction 11

```
for aid in sorted(artist_ids):
    try:
        result = apis.spotify.follow_artist(access_token=access_token,
          artist_id=aid)
        print(f"Followed artist {aid}: {result}")
    except Exception as e:
        print(f"Artist {aid}: {e}")

print("Done following all artists from liked songs.")
```

Observed response, interaction 11

```
Artist 33: Response status code is 422:
{"message":"You are already following this artist."}
Artist 34: Response status code is 422:
{"message":"You are already following this artist."}
Done following all artists from liked songs.
```

The actor corrects an intermediate list-access error and handles already-followed artists before completion at interaction 12.

G.3.2 Preserving album membership while choosing one song per group

AppWorld dev, task 6171bbc_1. The trajectory passes all 7 official checks in 21 actor interactions.

Task instruction

Make me a Spotify playlist called "My Most Played Album Songs" containing only the most-played song from each album in my album library.

The summary **retains eight album-to-song associations** for the pending within-album comparisons. The actor computes each maximum and inserts the selected songs into a new playlist.

Accepted LLM-generated summary after interaction 8 (complete text)

Spotify API docs explored: show_album_library (GET /spotify/library/albums, requires access_token, supports pagination/sort), show_album (GET /spotify/albums/{album_id}), show_song (GET /spotify/songs/{song_id}, returns play_count), show_song_privates (GET /spotify/songs/{song_id}/privates, returns liked/reviewed/downloaded). Supervisor password retrieved. Login completed yielding Bearer access_token. Album library fully retrieved (8 albums, terminal page): album 15 (Whispers in the Wind, song_ids [70,71,72,73]), album 13 (Starlight Serenades, [63,64,65,66]), album 18 (Echoes of Eternity, [80,81,82]), album 10 (Dreamscape Delights, [47-53]), album 11 (Synaptic Serenity, [54,55,56,57]), album 16 (Electric Dreamscape, [74,75,76]), album 9 (Mystical Crescendo, [44,45,46]), album 7 (Vibrant Visions, [33,34,35,36]). Remaining: call show_song for each song_id to get play_count, then determine max per album.

Observed response, interaction 13

```
Album 'Whispers in the Wind': Most played song ID = 73 (play_count =
  482)
Album 'Starlight Serenades': Most played song ID = 66 (play_count =
  975)
Album 'Echoes of Eternity': Most played song ID = 80 (play_count = 863)
Album 'Dreamscape Delights': Most played song ID = 51 (play_count =
  991)
Album 'Synaptic Serenity': Most played song ID = 54 (play_count = 845)
Album 'Electric Dreamscape': Most played song ID = 74 (play_count =
  846)
Album 'Mystical Crescendo': Most played song ID = 45 (play_count = 713)
Album 'Vibrant Visions': Most played song ID = 36 (play_count = 974)

Most played song IDs to add to playlist: [73, 66, 80, 51, 54, 74, 45,
  36]
```

Executed action, interaction 15

```
playlist = apis.spotify.create_playlist(title="My Most Played Album
  Songs", access_token=access_token)
print(playlist)
```

Observed response, interaction 15

```
{"message": "Playlist created.", "playlist_id": 654}
```

Task State after playlist creation: selected fields

```
{
  "key": "playlist_id",
  "value": 654,
  "evidence_status": "observed",
  "status": "active"
}
```

Executed action, interaction 20

```
most_played_song_ids = [73, 66, 80, 51, 54, 74, 45, 36]
playlist_id = 654
access_token = login_result["access_token"]

for song_id in most_played_song_ids:
    result = apis.spotify.add_song_to_playlist(playlist_id=playlist_id,
      song_id=song_id, access_token=access_token)
    print(f"Added song {song_id}: {result}")
```

Observed response, interaction 20 (confirmation excerpts)

```
Added song 73: {'message': 'Song added to the playlist.'}
Added song 66: {'message': 'Song added to the playlist.'}
[...]
Added song 36: {'message': 'Song added to the playlist.'}
```

Identifier 654 is reused for all eight insertions, following correction of a collection-format error and renewed inspection of the album list.

G.4 PRESERVING EVIDENCE ACROSS ORDERED ACTIONS

G.4.1 A COMPLETE MUSIC BACKUP BEFORE ACCOUNT DELETION

AppWorld train, task 6104387_3. The trajectory passes all 10 official checks in 34 actor interactions.

Task instruction
Export a unique list of all the songs in my song and album library and all playlists in my Spotify account into "~/backups/spotify_songs.csv" file in my file system. The file should have headers, "Title" and "Artists" and artists should be separated by "|". Terminate my account after this backup is complete.

The actor combines the song, album, and playlist libraries into `all_songs`, recovering from collection and authentication errors. File creation precedes account deletion.

Interactions	Observed progression
1–10	Discover interfaces, authenticate, and paginate through libraries.
11–18	Inspect albums and playlists; correct argument, syntax, and key errors.
19	Successfully assemble the deduplicated collection of 70 songs.
20–22	Inspect file operations and construct a CSV with 70 song rows.
23–27	Correct file-system authentication and successfully create the backup file.
28–33	Confirm the deletion interface, correct authentication, and delete the account.
34	Submit completion after both effects have been observed.

Observed response, interaction 19

```
Total unique songs: 70
```

Accepted resolution after successful collection: selected fields

```
{
  "op": "RESOLVE",
  "key": "recovery_required:spotify.show_playlist",
  "value": true
}
```

The resolution establishes **recovery of playlist retrieval**; backup and deletion remain pending. The `all_songs` and `csv_content` bindings carry collected content into those operations.

Observed response, interaction 22

```
CSV has 70 song rows
First few lines:
Title,Artists
A Symphony Written in Tears,Aria Sterling
Blind Trust,Isabella Cruz
Veiled Sorrow,Grace Harrison
Azure Skies,Liam Palmer
```

Executed action, interaction 27

```
fs_token = fs_login["access_token"]

apis.file_system.create_directory(directory_path="~/backups",
  access_token=fs_token, recursive=True)

apis.file_system.create_file(file_path="~/backups/spotify_songs.csv",
  access_token=fs_token, content=csv_content, overwrite=True)
print("CSV file created successfully!")
```

Observed response, interaction 27

```
CSV file created successfully!
```

Executed action, interaction 33

```
spotify_access_token = spotify_login["access_token"]
result = apis.spotify.delete_account(access_token=spotify_access_token)
print(result)
```

Observed response, interaction 33

```
{"message": "User account with all data belonging to it have been
  deleted."}
```

File creation completes at interaction 27, account deletion at 33, and task submission at 34.

G.5 File identities across work-unit boundaries

G.5.1 Use the new file paths after renaming

AppWorld dev, task 68ee2c9_1. The trajectory passes all 5 official checks in 11 actor interactions.

Task instruction
In my file system, add the prefix "YYYY-MM-DD_" to all file names in the ~/downloads/ directory, based on their creation dates, and then move all files not from this year to ~/trash/.

Renaming changes the exact paths that later moves must use. The actor first gathers creation dates for 100 files, then stores each original path, renamed path, and year in renamed_files. At the handoff, the event records the completed rename phase while the later move still consumes the exact new paths.

Observed response, interaction 8

```
Number of files: 100
File: /home/nancy/downloads/DIY_home_repair_guide.docx, Created:
  2022-12-04T10:24:57
File: /home/nancy/downloads/art_inspiration_sketches.zip, Created:
  2022-07-07T09:27:11
File: /home/nancy/downloads/bike_lights.jpg, Created:
  2023-03-30T03:02:29
```

Executed action, interaction 9 (rename and binding excerpt)

```
new_filename = f"{date_prefix}_{filename}"
new_file_path = f"{directory}/{new_filename}"

result = apis.file_system.move_file(
    access_token=access_token,
    source_file_path=file_path,
    destination_file_path=new_file_path
)
renamed_files.append({
    'original': file_path,
    'new': new_file_path,
    'year': int(date_prefix.split('-')[0])
})
```

Observed response, interaction 9

```
Renamed: DIY_home_repair_guide.docx ->
  2022-12-04_DIY_home_repair_guide.docx
Renamed: art_inspiration_sketches.zip ->
  2022-07-07_art_inspiration_sketches.zip
```

Observed response, interaction 9 (continued)

```
Renamed: bike_lights.jpg -> 2023-03-30_bike_lights.jpg
[...]
Total files renamed: 100
```

Event Memory after renaming: selected fields

```
{
  "objective": "Rename files with date prefix",
  "outcome": "progress",
  "reusable_outputs": {
    "files_renamed_count": 100,
    "method": "file_system.move_file within same directory",
    "rename_pattern": "YYYY-MM-DD_<original_filename>",
    "source_directory": "~/downloads/"
  }
}
```

Executed action, interaction 10

```
import os

files_to_move = [f for f in renamed_files if f['year'] != current_year]
print(f"Files to move to trash (not from {current_year}):
  {len(files_to_move)}")

for f in files_to_move:
    filename = os.path.basename(f['new'])
    destination = f"~/trash/{filename}"
    result = apis.file_system.move_file(
        access_token=access_token,
        source_file_path=f['new'],
        destination_file_path=destination
    )
    print(f"Moved to trash: {filename}")

print(f"\nTotal files moved to trash: {len(files_to_move)}")
```

Observed response, interaction 10

```
Files to move to trash (not from 2023): 27
Moved to trash: 2022-12-04_DIY_home_repair_guide.docx
Moved to trash: 2022-07-07_art_inspiration_sketches.zip
[...]
Total files moved to trash: 27
```

Event Memory records the completed rename phase, while the next action uses the `renamed_files` binding to move 27 files by their updated paths. This case illustrates **reuse of a runtime binding across a work-unit boundary**.

G.6 Completed effects and pending work

AppWorld train, task ce359b5_1. The trajectory passes all 8 official checks in 32 actor interactions.

Completed effects and remaining work

Task instruction
Remove all songs from my Spotify song library and playlists that were released before 2021 year.

The summary **distinguishes completed library removals from pending playlist removals**, preserving the exact playlist-to-song mapping through an authorization failure and recovery.

Completed effects and remaining work (continued)

Observed response, interaction 22

```
Songs to remove from library: [294, 301, 208, 110, 199]
Songs to remove from playlists: {195: [97, 159], 194: [143], 193:
  [112], 192: [223, 290], 196: [163, 225, 248]}
```

Observed response, interaction 23 (confirmation excerpts)

```
Removed song 294 from library: {'message': 'Song removed from
  library.'}
[...]
Removed song 199 from library: {'message': 'Song removed from
  library.'}
```

Accepted LLM-generated summary, interaction 23: completed and pending effects excerpt

Library removals completed: [294, 301, 208, 110, 199]. Remaining playlist removals: {195: [97, 159], 194: [143], 193: [112], 192: [223, 290], 196: [163, 225, 248]}.

Observed response, interaction 24

```
Exception: Response status code is 401:
{"message":"You are either not authorized to access this spotify API
  endpoint or your access token is missing, invalid or expired."}
```

Executed action, interaction 31 (mapping and call excerpt)

```
playlist_removals = {195: [97, 159], 194: [143], 193: [112], 192:
  [223, 290], 196: [163, 225, 248]}

for playlist_id, song_ids in playlist_removals.items():
    for song_id in song_ids:
        result = apis.spotify.remove_song_from_playlist(␣
          playlist_id=playlist_id, song_id=song_id,
          access_token=access_token)
```

Observed response, interaction 31 (confirmation excerpts)

```
Removed song 97 from playlist 195: {'message': 'Song removed from
  playlist.'}
Removed song 159 from playlist 195: {'message': 'Song removed from
  playlist.'}
[...]
Removed song 248 from playlist 196: {'message': 'Song removed from
  playlist.'}
```

The mapping is reused for five playlists before completion at interaction 32. The excerpt demonstrates retention of these fields; it does not establish exactness of every summary field.