RECONBENCH: EVALUATING CONTEXT COMPRESSION THROUGH EXECUTABLE AGENT RESUMPTION

Anonymous authors
Paper under double-blind review

ABSTRACT

During a long task, an agent accumulates a history of decisions, tool calls, and observations. Compressing this history reduces the context passed to a successor, but may also remove information needed to finish the task. We introduce RECONBENCH, a benchmark that evaluates context compression through task completion after an interruption. Starting from a verified successful execution, we select a checkpoint, restore its workspace, compress the preceding interaction history into a memory, and ask a fresh agent to complete the task. A task-specific verifier evaluates the final workspace. The benchmark pool contains 2,694 successful trajectories organized into four workflow families. Our main study evaluates 32,400 continuation runs across interruption stages, compressor scales, memory methods, and budgets within three source/continuation-agent strata. Among verifier-valid runs, 57.3% complete the task, with a mean normalized score of 0.834. Success rises from 49.4% at an early checkpoint to 67.8% at a late checkpoint, and from 54.2% with a 2B compressor to 59.6% with a 9B compressor. Memory methods differ by only a few percentage points, and larger allowances produce neither proportional increases in memory length nor consistently higher success. These findings show why compression should be evaluated through both the task it preserves and the work required after handoff.

1 INTRODUCTION

An agent working on a long task accumulates more than a transcript. It makes decisions, tests hypotheses, calls tools, and changes files in a workspace. If it reaches a context limit or another agent takes over, the successor needs enough information to understand what has happened and what remains to be done. Passing the complete history can be expensive; passing too little may force the successor to repeat work or act on an outdated understanding of the workspace.

This exposes a gap between two evaluation traditions. Prompt-compression work tests static retrieval or question answering (Jiang et al., 2024; Xu et al., 2024); related approaches study learned latent compression (Ge et al., 2024). Agent benchmarks test end-to-end tool use (Liu et al., 2024b; Wang et al., 2024), including web and software environments (Zhou et al., 2024; Jimenez et al., 2024) and desktop settings (Xie et al., 2024). These benchmarks usually start from the original task specification and do not ask whether a successor can finish after a mid-task handoff.

We propose *executable agent resumption*: evaluating a compressed history by asking a fresh agent to continue the task. RECONBENCH begins with a verified successful execution, called the *source trajectory*. At a selected checkpoint, the interaction history before interruption is the *prefix*. A compressor turns this prefix into a bounded text *memory*. The successor receives the original task, the checkpoint's workspace snapshot, and the memory, then uses tools to finish the task. A verifier checks the resulting workspace for success and partial completion. Figure 1 shows what crosses this handoff boundary.

In our protocol, *Full Prefix* is the verified source oracle (100% by construction). *Raw-Prefix Continuation* gives a fresh agent the uncompressed prefix and checkpoint state, isolating agent replacement from compression loss.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

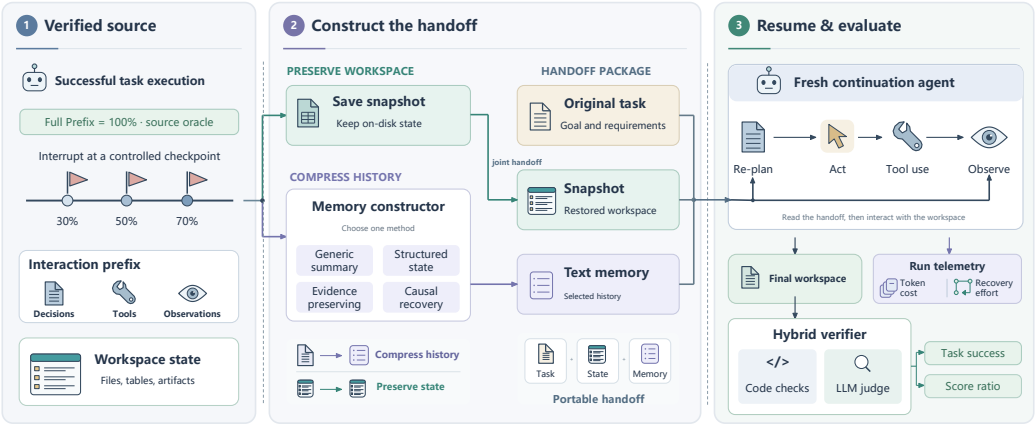


Figure 1: The RECONBENCH protocol. The successful source execution supplies an oracle and a controlled checkpoint. One of four memory constructors compresses the interaction prefix; only the workspace snapshot, task, and compressed memory cross the handoff boundary. Hybrid verification assesses the final workspace, while run telemetry records token cost and recovery effort.

Our main study evaluates 32,400 continuation runs and finds that 57.28% of verifier-valid runs finish the task. Later checkpoints and larger compressors show the clearest improvements; memory methods and nominal budgets have smaller effects.

Our contributions are:

- We formulate context compression for long-horizon agents as an executable state-handoff problem and define resumption fidelity as task completion after interruption.
- We introduce RECONBENCH, a protocol that combines verified source trajectories, controlled interruption, workspace snapshots, portable memories, fresh-agent continuation, and verifier-based scoring.
- We provide a benchmark pool of 2,694 verified trajectories across four task families and a 32,400-cell factorial study covering interruption stage, agent and compressor scale, memory construction, and budget.
- We find larger differences in success across interruption stages and compressor scales than across memory methods or budgets, and show why evaluations should report both allowed and realized memory length.

2 RELATED WORK

Agent evaluation. AgentBench and MINT evaluate language agents across environments and tasks involving tools and feedback (Liu et al., 2024b; Wang et al., 2024). WebArena and GAIA cover web interaction and general assistance (Zhou et al., 2024; Mialon et al., 2024); SWE-bench and OSWorld target software and desktop operation (Jimenez et al., 2024; Xie et al., 2024). SWE-agent, AgentDojo, and MLE-bench extend coverage to software engineering, security, and machine-learning engineering (Yang et al., 2024; Debenedetti et al., 2024) (Chan et al., 2025). Their standard evaluations run an agent from the start of a task. RECONBENCH uses task completion while placing the evaluation boundary at a checkpoint.

Context compression. LLMLingua and LongLLMLingua reduce prompt length using token-level or learned importance signals (Jiang et al., 2023; 2024); LLMLingua-2 applies data distillation (Pan et al., 2024). RECOMP and the in-context autoencoder compress retrieved or conversational context (Xu et al., 2024; Ge et al., 2024); Gist Tokens and AutoCompressor learn compact latent summaries (Mu et al., 2023; Chevalier et al., 2023). These methods target static-query utility, whereas our benchmark asks whether a compressed object lets a new agent choose tools, interpret observations, and reach a verifiable terminal state.

Agent memory. Reflexion and ExpeL extract verbal feedback or reusable experience from trajectories (Shinn et al., 2023; Zhao et al., 2024). Generative Agents and MemoryBank study persistent memory (Park et al., 2023; Zhong et al., 2024); LongMem and HippoRAG study retrieval-augmented memory (Wang et al., 2023; Gutierrez et al., 2024). LoCoMo and LongMemEval evaluate long-term interactive recall (Maharana et al., 2024; Wu et al., 2025). RECONBENCH complements these systems with a checkpoint setting in which memory quality is determined by executable completion under a fixed workspace state.

Long-context efficiency. Lost in the Middle shows that more context does not guarantee reliable use (Liu et al., 2024a). LongBench and RULER expose differences between nominal and effective context length (Bai et al., 2024; Hsieh et al., 2024). StreamingLLM and YaRN improve long-context handling (Xiao et al., 2024; Peng et al., 2024), while LongRoPE and SnapKV target extended positions and cached keys (Ding et al., 2024; Li et al., 2024). These approaches address model or runtime capacity; we measure the behavioral consequence of selecting a compact state handoff.

3 RECONBENCH

3.1 DESIGN PRINCIPLES

A useful handoff benchmark must let us compare memories while holding the task and checkpoint fixed. We follow four design principles. *Executable evaluation* measures whether the successor finishes the task. *Checkpoint consistency* aims to give each memory condition the same workspace at a given interruption point. *Source feasibility* requires a verified successful source execution. *Paired comparison* evaluates multiple memories derived from the same prefix with the same task and verifier. Together, these principles make differences between handoffs easier to interpret.

These principles focus evaluation on information needed to act: current state, unresolved decisions, and the next action. A summary can restate the goal yet omit a failed command, existing artifact, or ruling constraint. RECONBENCH therefore evaluates memories through execution rather than reference-summary similarity.

3.2 PROBLEM DEFINITION

Let a task be τ, a source agent produce an interaction trajectory $\xi = (u_0, a_0, o_0, \ldots, a_T, o_T)$, and $V(\tau, w_T)$ be a deterministic or judge-backed verifier over the final workspace w_T. A source trajectory is admitted only when $V(\tau, w_T) = 1$ (or its normalized score reaches the benchmark pass threshold). For an interruption fraction $\rho \in \{0.3, 0.5, 0.7\}$, the prefix ends at step $k = \lfloor \rho T \rfloor$. The environment state after executing the prefix is serialized as a workspace snapshot s_k.

A memory constructor C maps the visible prefix $\xi_{\le k}$ to a text package $m = C(\xi_{\le k}; b, h)$, where b is a nominal budget and h denotes the construction method. A continuation agent π receives (τ, s_k, m) and generates the suffix execution $\hat{\xi}_{>k}$, producing workspace $\hat{w}_T$. The primary metric is

$$\mathrm{ResumptionSuccess}(\tau, \rho, C, \pi) = \Pr\left[V(\tau, \hat{w}_T) = 1 \mid s_k, C(\xi_{\le k})\right]. \tag{1}$$

Because all source trajectories pass verification, this is also a direct measure of capability retention relative to the verified source reference. A score ratio $r = V_{\mathrm{score}}(\tau, \hat{w}_T)/V_{\mathrm{score}}(\tau, w_T)$ captures partial progress when the verifier supplies a graded score.

3.3 DECOMPOSING THE HANDOFF STATE

The handoff combines three kinds of information. The *task contract* specifies the goal, constraints, and available tools. The *workspace state* contains files, edits, generated artifacts, and environment metadata. The *interaction history* records plans, observations, rejected alternatives, unresolved errors, and intended next steps. Some of this history is visible in the workspace, while some explains how to interpret it. At interruption k, the handoff is

$$H_k = (\tau,\ s_k,\ C(\xi_{\le k}))\,. \tag{2}$$

The original task is supplied again, and the workspace is restored directly. The interaction history is the part compressed into memory. This gives the memory a specific role: preserve the information

Table 1: Benchmark and main-study scale. Maxima are computed over the verified source pool; experimental cells refer to the continuation study.

Statistic	Value
Verified source trajectories	2,694
Task families	4
Maximum assistant turns per source	29
Maximum tool calls per source	28
Maximum response tokens per source	22,504
Sources in the main study	300
Conditions derived per source	108
Planned continuation cells	32,400

that helps the successor interpret the files and choose its next action. For example, a saved output can show what was computed, while the memory records whether that output has been checked and which requirements remain unresolved.

The snapshot and memory are complementary. A snapshot answers *what exists now*; a memory can answer *why it exists, whether it is trusted, and what should happen next*. A file may reveal a partially implemented solution but not the failed tests that motivated it. A log may contain an error without indicating which hypothesis has already been ruled out. Resumption fidelity measures whether the two channels jointly restore enough actionable state for the successor to close this gap.

3.4 Verified Source Trajectories

The benchmark pool contains 2,694 verified successful trajectories. We organize its workloads into four broad task families by their primary operational objective: *data processing and reconciliation*, which transforms scattered records into consistent outputs; *rule auditing and compliance*, which checks evidence against explicit constraints; *operational coordination and resource management*, which reconciles schedules, inventories, and resource requirements; and *technical diagnosis and artifact maintenance*, which inspects logs, configurations, or software artifacts to produce corrective outputs. This workflow taxonomy describes the work required of the agent across application domains. Appendix A.2 expands the definitions and their implications for memory.

Each source has at most 29 assistant turns, 28 tool calls, and 22,504 response tokens. The verified endpoint establishes task feasibility and supplies the Full Prefix reference. The main study uses 300 source trajectories, with 100 from each source-model stratum, representing 151 unique task identifiers. Each source supplies three checkpoints and every compressor, memory-method, and budget combination. Table 1 summarizes the pool and experimental sample.

3.5 Interruption and Workspace Snapshot

At each interruption, the harness replays the prefix to materialize the continuation workspace. The standard handoff supplies the task, restored workspace, and compressed memory; the Raw-Prefix control supplies the uncompressed prefix instead. Replay observations and file counts audit the checkpoint (Appendix A.5).

3.6 Memory Construction Methods

We compare four prompt-and-pipeline constructions: `generic_summary`, a general-purpose summary of the prefix; `causal_recovery`, which emphasizes why previous actions were taken and what remains; `evidence_preserving`, which retains observations and concrete evidence; and `structured_state`, which writes an explicit state representation containing progress, artifacts, constraints, and next actions. Each constructor is run with nominal budgets of 10%, 30%, and 50% of the source-history allowance. The budget is an upper bound on the generated memory, not a promise that the constructor will fill it.

The memory is ordinary text, so a different model can consume it without access to the compressor's internal state. The study evaluates both compressors and continuation agents at 2B, 4B, and 9B scales.

3.7 Resumption Protocol and Metrics

Workspace-based verification. We evaluate the final workspace against task-specific criteria. Deterministic code checks structured properties, such as required artifacts, valid schemas, and numerical constraints; an LLM judge assesses unstructured content, such as whether a report or message satisfies the task's semantic requirements. These criterion outcomes yield the task's graded score and pass decision. The evaluation target is the produced artifact state, so the compressed memory is assessed through what the successor accomplishes. Appendix A.5 describes the division of responsibility and recorded outputs.

For each source trajectory, the harness executes the following protocol:

1. verify the complete source trajectory and select an interruption fraction;
2. replay the prefix and save the workspace snapshot;
3. construct one bounded memory from the visible prefix;
4. restore the snapshot for a fresh continuation agent and execute to termination;
5. run the task verifier and record binary success, score ratio, token usage, turns, tool calls, and recovery effort.

We report four complementary outcomes. *Success* is the final task-verifier pass indicator, and *verifier score* is the reported score divided by its maximum. Scoring evaluates the resulting workspace and uses the configured 9B judge where required. *End-to-end token cost* adds compression API prompt and completion tokens to the successor's prompt and response tokens across its tool-use sequence. *Observed recovery effort*, our operational proxy for recovery debt, comprises continuation turns, tool calls, tokens, and limit exhaustion. These measurements jointly describe correctness, partial completion, inference cost, and post-handoff work; Appendix B specifies the aggregation rules.

A failed task and a failed evaluation are recorded separately. A continuation is valid when execution produces a usable terminal trace and workspace; a verifier result is valid when scoring completes and returns a usable outcome. Success is calculated over verifier-valid runs. Appendix C reports the counts for each stage.

4 Experimental Setup

Research questions. We ask whether a compressed handoff preserves completion, how this varies with checkpoint, compressor scale, method, and budget, and what work follows handoff. We also include a raw-prefix continuation control to separate agent-replacement effects from compression loss.

Design. The experiment has three source/continuation-agent strata, each containing 100 source trajectories. Every source is evaluated with three compressors, four memory methods, three budgets, and three interruption stages, giving $3 \times 100 \times 3 \times 4 \times 3 \times 3 = 32{,}400$ planned runs. Comparisons of compressors, methods, and budgets are paired within the same source and interruption stage.

Models and sampling. The runs use Qwen3.5 2B, 4B, and 9B configurations. Each source-model stratum is paired with a same-scale continuation agent, and all compressors are evaluated within every stratum. Continuation uses temperature 0.6. Success and score use verifier-valid continuations; execution-cost summaries use continuation-valid runs. The compression ledger covers all 32,400 constructed memories.

Cost accounting. We distinguish visible-history length, requested memory allowance, realized memory, and API usage. Compression cost counts both reading the prefix and generating the memory; continuation cost accumulates prompt and response tokens over the successor's execution. For

Table 2: Main marginal results. Success is computed over verifier-valid runs; score ratio is normalized to the verified source endpoint.

Factor	Setting	Success (%)	Score ratio
Reference controls			
	Full Prefix (source oracle)	100.00	1.000
	Raw-prefix continuation	91.70	0.965
Interruption stage	30%	49.43	0.781
	50%	54.62	0.830
	70%	67.77	0.890
Compressor scale	2B	54.23	0.816
	4B	57.97	0.842
	9B	59.63	0.844
Memory method	generic summary	57.96	0.836
	causal recovery	57.77	0.835
	evidence preserving	57.63	0.838
	structured state	55.74	0.827
Nominal budget	10%	56.83	0.825
	30%	57.71	0.837
	50%	57.29	0.839

the current ledger, we add mean compression cost to mean continuation cost within each factorial cell and then average cells equally. Appendix G reports the full breakdown.

Statistical analysis. We report pooled success rates and descriptive Wilson intervals. To account for runs sharing a source, we also bootstrap the 300 source trajectories, keeping all conditions from each sampled source together. The same procedure provides intervals for score, memory length, tokens, turns, and tool calls. We additionally resample task identifiers to account for tasks represented in more than one source stratum. The appendix reports clustered intervals and paired contrasts; exploratory comparisons supplement the main factor summaries.

5 Results

5.1 Overall Resumption Fidelity

Across the 32,382 verifier-valid runs, 18,547 pass the task verifier, giving a retained success rate of 57.28% (descriptive 95% Wilson CI [56.74, 57.81]) and mean normalized score 0.834. Continuations average 11 assistant turns, 14.6 tool calls, and 101k tokens. Table 2 summarizes the factorial results and reference controls; Figure 2 shows the factor differences.

The controls clarify the gap: raw-prefix continuation reaches 91.7% success (0.965 score), close to the Full Prefix oracle and well above compressed handoff (57.3%, 0.834). The difference therefore reflects both agent replacement and information loss from compression.

The 300 source trajectories correspond to 151 unique task identifiers, some represented in multiple source strata. The pooled rate gives equal weight to each valid run. Giving each task equal weight instead yields 52.52% success (task-bootstrap 95% CI [48.27, 56.63]). We use the pooled rate to summarize the experimental grid and the task-level average to show sensitivity to task composition. Appendix F provides the weighting details.

We next examine which handoffs preserve more of the source's verified capability, starting with interruption stage and compressor scale.

5.2 Later Checkpoints Are Easier to Resume

Success increases most clearly with interruption stage. At 30%, 50%, and 70% of the source trajectory, success is 49.43%, 54.62%, and 67.77%, with mean normalized scores of 0.781, 0.830, and

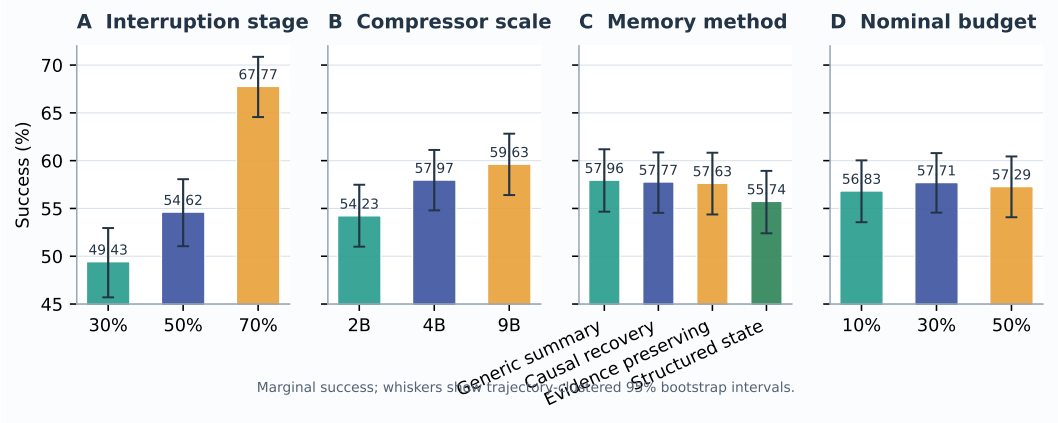


Figure 2: Main factors associated with executable resumption. Interruption stage has the largest observed spread, compressor scale has a smaller monotonic association, and method and nominal-budget marginals are tightly grouped.

0.890. Continuation tokens fall from 114,871 at the earliest checkpoint to 88,487 at the latest, while mean turns fall from 11.77 to 10.22. Later checkpoints leave a shorter portion of the source execution to complete, and the successor both succeeds more often and uses fewer tokens. This makes checkpoint position an important part of any comparison between memory methods.

5.3 Larger Compressors Preserve More Task Success

The 2B, 4B, and 9B compressors achieve 54.23%, 57.97%, and 59.63% success, respectively, with mean normalized scores of 0.816, 0.842, and 0.844. The success increase is larger from 2B to 4B than from 4B to 9B. All three compressors process the same source checkpoints, so these comparisons hold the source and continuation agent fixed. Mean continuation tokens decrease from 108,632 for the 2B compressor to 97,036 for the 9B compressor. In this study, larger compressors are associated with both higher success and less subsequent execution.

5.4 Memory Format and Budget Have Smaller Effects

The four memory methods have similar aggregate success rates. Generic summaries achieve 57.96%, causal recovery 57.77%, evidence preserving 57.63%, and structured state 55.74%. These differences are smaller than those across interruption stages or compressor scales. Evidence preserving has the highest mean normalized score (0.838), although generic summaries have the highest binary success. The two metrics capture different aspects of performance: how often every requirement is met and how much of the task is completed on average.

Success does not increase consistently with the nominal budget: the 10%, 30%, and 50% allowances yield 56.83%, 57.71%, and 57.29% success. The memories actually produced help interpret this result. The 10% allowance is nearly filled (364 tokens; 92.6% utilization), whereas the 30% and 50% allowances produce 662 and 750 tokens, using 64.3% and 45.3% of their available space. Increasing the allowance therefore does not increase memory length proportionally. Figure 3 shows why both quantities are needed to describe a compression setting.

The budget trend differs by method: causal recovery rises from 56.78% to 58.43%, evidence preserving from 55.88% to 58.78%, while generic summary stays near 58%. Structured state is longest at 50% (911 tokens) but reaches 54.10% success. Appendix E gives the complete breakdown.

5.5 Token Cost and Recovery Debt

The compression ledger averages 4,240 source-history tokens and 592 realized memory tokens. Constructing a memory consumes 4,560 API prompt tokens and 592 completion tokens, or 5,152 tokens in total. Averaging the 324 groups defined by the experimental factors gives approximately

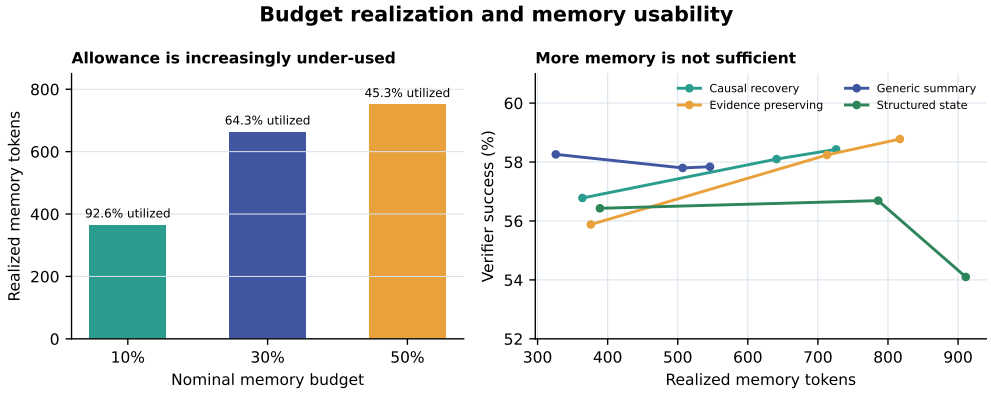


Figure 3: Nominal memory allowance versus realized memory and resumption success. Constructors increasingly under-use larger allowances, while success remains nearly flat.

101,323 continuation tokens and 106,475 tokens for construction plus continuation. Compression accounts for 4.84% of this combined cost. Most tokens are therefore spent executing the remaining task, which makes downstream effort important even when the handoff memory is short. Appendix G gives the breakdown by compressor, budget, and stage.

Post-handoff effort also decreases at later checkpoints. The 70% stage averages 10.22 turns and 13.65 tool calls, compared with 11.77 turns and 15.53 calls at 30%. These measurements describe how much work the successor performs. Reporting them alongside success makes it possible to distinguish a run that finishes efficiently from one that spends substantial effort without meeting the task requirements.

5.6 Replay Fidelity and Failure Modes

The metadata reveals a mechanism stronger than nominal memory length: state fidelity dominates budget. Among verifier-valid runs with no recorded replay observation mismatch, success is 61.83%; runs with at least one mismatch succeed at 38.76%, a 23.07-point gap that persists at every stage (55.55% vs. 15.43% at 30%, 59.45% vs. 34.87% at 50%, and 71.39% vs. 56.53% at 70%). Conditioning on matched replay, the 10%, 30%, and 50% budgets yield 61.824%, 61.826%, and 61.842% success. Thus the relevant bottleneck is faithful executable state, not allowance size; a high-level pass rate can otherwise conflate memory quality with environment restoration.

The methods are also complementary rather than globally rankable. In 8,082 matched blocks with valid outcomes for all four methods, all four succeed in 31.91% and none succeeds in 19.30%; the remaining 48.79% contain both successes and failures. The retrospective union of the four methods succeeds in 80.70%, versus 58.03% for Generic Summary on the same support. Although this union is not a deployable single-run baseline, the pattern indicates that different representations preserve different actionable facts. It motivates adaptive or ensemble memory selection instead of searching for one universally best format.

Failure termination points to a third mechanism: obligation closure. Of 13,835 verifier-valid failures, 81.98% end with no further tool call; 52.17% miss exactly one verifier criterion, and 22.95% achieve a score ratio of at least 0.9. Many failures therefore reach a nearly complete state but stop before checking or satisfying the last requirement. Preserving completion criteria and unresolved obligations is a concrete design target. Appendix H gives the full distributions.

6 Discussion

Memory should support the next decision. A successor uses memory to decide what to inspect, which result to trust, and what action to take. Useful content can therefore include a failed approach, an unresolved constraint, or the reason a file was changed. Section headings such as “current state” and “next step” may make this information easier to find, but the small aggregate differences between

methods suggest that headings alone are not enough. A useful next experiment would compare the information preserved within each format.

Compression is a state-preservation problem. The replay split and the matched-budget analysis point to the same conclusion: once the executable workspace is faithful, extra text allowance has little aggregate value. The central design problem is therefore to preserve the facts that determine the next action and the remaining verifier obligations, rather than to maximize summary length.

Heterogeneity favors adaptive memory. Similar marginal method scores conceal substantial block-level disagreement: each representation rescues some source/checkpoint conditions that others miss. A future system could select or combine memories using checkpoint state, task family, or early successor feedback, with selection evaluated by executable completion rather than summary similarity.

Checkpoint position changes the remaining task. At a later checkpoint, more work is already reflected in files and other artifacts. The successor has a different starting state and a shorter remaining execution horizon. The observed increase in success should therefore inform checkpoint selection as well as memory design. Reporting interruption stage allows readers to tell whether a method works throughout a task or mainly near its completion.

Memory length and execution cost serve different purposes. The allowance describes available space, realized memory length describes the handoff, and continuation tokens describe the work that follows. Our results show why none of these alone is a sufficient cost measure. A system should compare the total inference cost needed to achieve a given level of task completion, including the cost of constructing its memory.

Implications for agent systems. The benchmark suggests managing workspace state and conversational memory together. Persistent files preserve concrete work; memory can preserve decisions, evidence, and unfinished obligations. A system designer can then evaluate compressor choice, memory budget, and checkpoint timing against the same task-completion objective. This gives context management a measurable goal: reduce the cost of carrying history forward while preserving the ability to finish.

7 LIMITATIONS

Our results are conditional on 300 successful source trajectories covering 151 tasks, with source and continuation-agent scales coupled. Single runs per condition leave method differences and success-set overlap sensitive to sampling; the retrospective method union requires an equal-compute repeated-seed control. Replay consistency does not establish snapshot identity or causality, and aggregate failure profiles do not identify lost information. Recovery effort includes necessary remaining work, while token counts do not directly measure monetary cost.

8 CONCLUSION

We introduced RECONBENCH, an executable benchmark for context compression in long-horizon agents. By interrupting verified successful trajectories, checkpointing workspace state, compressing interaction history, and testing a fresh agent on the remaining task, the benchmark turns "good memory" into an observable behavioral property. In 32,400 continuation cells, 57.3% of verifier-valid runs completed the task and retained 0.834 of the source score on average. Progress stage and compressor capacity were the most prominent factors, while memory format and nominal budget had smaller and non-monotonic effects. The results suggest that future context-management systems should optimize for portable, action-relevant state and evaluate it through resumption, cost, and recovery effort together.

AI USE STATEMENT

This work used generative AI tools to assist with analysis-script development, and checks of aggregate results. The benchmark protocol, experimental runs, verifier outputs, and reported numerical results were generated by the documented evaluation pipeline and checked against saved metadata artifacts. The authors reviewed AI-assisted text and code, and take responsibility for the final manuscript, claims, and artifacts.

ETHICS STATEMENT

The study evaluates software agents in isolated workspaces and does not involve human subjects or human-participant data. The benchmark uses task artifacts and generated execution traces; any release should follow the licenses and privacy constraints of the underlying artifacts. The main risk is overinterpreting automated verifier scores as general agent reliability, which we address by reporting validity audits, replay mismatches, partial scores, and limitations of judge-backed criteria.

REPRODUCIBILITY STATEMENT

The main paper specifies the handoff protocol, factorial design, metrics, and primary aggregates. The appendix documents source sampling, checkpoint and verifier behavior, validity denominators, clustered uncertainty, failure profiles, and the token ledger. The accompanying analysis scripts generate the scalar metadata tables without reading the large conversation or workspace payloads; the submission supplementary material should include these scripts, the small aggregate artifacts, environment instructions, and an anonymized artifact manifest.

REFERENCES

Yushi Bai et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *Annual Meeting of the Association for Computational Linguistics*, 2024.

John S. Chan et al. Mle-bench: Evaluating machine learning agents on machine learning engineering. In *International Conference on Learning Representations*, 2025.

Alexis Chevalier et al. Adapting language models to compress contexts. In *Conference on Empirical Methods in Natural Language Processing*, 2023.

Edoardo Debenedetti et al. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for llm agents. In *Advances in Neural Information Processing Systems*, 2024.

Yiran Ding et al. Longrope: Extending llm context window beyond 2 million tokens. In *International Conference on Machine Learning*, 2024.

Tao Ge et al. In-context autoencoder for context compression in a large language model. In *International Conference on Learning Representations*, 2024.

Bernal Jimenez Gutierrez et al. Hipporag: Neurobiologically inspired long-term memory for large language models. In *Advances in Neural Information Processing Systems*, 2024.

Cheng-Ping Hsieh et al. Ruler: What's the real context size of your long-context language models? In *Conference on Language Modeling*, 2024.

Hui Jiang et al. Llmlingua: Compressing prompts for accelerated inference of large language models. In *Conference on Empirical Methods in Natural Language Processing*, 2023.

Hui Jiang et al. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. In *Annual Meeting of the Association for Computational Linguistics*, 2024.

Carlos E. Jimenez et al. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, 2024.

Yuhong Li et al. Snapkv: Llm knows what you are looking for before generation. In *Advances in Neural Information Processing Systems*, 2024.

Nelson F. Liu et al. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024a.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu-Cheng Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, 2024b.

Adyasha Maharana et al. Evaluating very long-term conversational memory of llm agents. In *Annual Meeting of the Association for Computational Linguistics*, 2024.

Grégoire Mialon et al. Gaia: A benchmark for general ai assistants. In *International Conference on Learning Representations*, 2024.

Jesse Mu et al. Learning to compress prompts with gist tokens. In *Advances in Neural Information Processing Systems*, 2023.

Zizheng Pan et al. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024.

Joon Sung Park et al. Generative agents: Interactive simulacra of human behavior. In *ACM Symposium on User Interface Software and Technology*, 2023.

Bowen Peng et al. Yarn: Efficient context window extension of large language models. In *International Conference on Learning Representations*, 2024.

Noah Shinn et al. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

Weizhi Wang et al. Augmenting language models with long-term memory. In *Advances in Neural Information Processing Systems*, 2023.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. In *International Conference on Learning Representations*, 2024.

Di Wu et al. Longmemeval: Benchmarking chat assistants on long-term interactive memory. In *International Conference on Learning Representations*, 2025.

Guangxuan Xiao et al. Streamingllm: Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024.

Tianbao Xie et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, 2024.

Fangyuan Xu et al. Recomp: Improving retrieval-augmented language models with context compression and selective augmentation. In *International Conference on Learning Representations*, 2024.

John Yang et al. Swe-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, 2024.

Aoran Zhao et al. Expel: Llm agents are experiential learners. In *AAAI Conference on Artificial Intelligence*, 2024.

Wanjun Zhong et al. Memorybank: Enhancing large language models with long-term memory. In *AAAI Conference on Artificial Intelligence*, 2024.

Shuyan Zhou et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024.

A Benchmark Construction Details

A.1 Factorial Expansion

The main study starts from 300 verified source trajectories, with 100 drawn from each of the 2B, 4B, and 9B source strata. For every source, we create three interruption checkpoints. At each checkpoint, three compressor scales instantiate four memory-construction prompts at three budgets. This produces $3 \times 3 \times 4 \times 3 = 108$ handoff conditions per source and 32,400 planned continuation runs overall. Table 3 lists the axes explicitly.

Table 3: Factorial axes in the main continuation study.

Axis	Levels	Count
Source/continuation stratum	Qwen3.5 2B, 4B, 9B	3
Source trajectories per stratum	Verified successful samples	100
Interruption stage	30%, 50%, 70%	3
Compressor scale	Qwen3.5 2B, 4B, 9B	3
Memory construction	Generic, structured, evidence, causal	4
Nominal memory budget	10%, 30%, 50%	3
Conditions per source trajectory		108
Total planned continuation runs		32,400

Every derived condition retains a stable cluster identifier based on the source-model stratum and source sample. This identifier links runs that share the same task and source trajectory while allowing their compressor, memory method, budget, and stage to vary. It is used both for paired comparisons and for cluster-level resampling.

A.2 Task Families and Handoff Requirements

The four families group workflows by their primary objective. They summarize operational patterns across domains, rather than defining industry categories or experimental strata. A workflow can involve several capabilities; its dominant deliverable determines the most useful descriptive family.

Data processing and reconciliation. The agent extracts, cleans, joins, and aggregates records into a consistent result. Typical operations include reconciling transactions, matching records across files, and calculating totals from noisy logs. A useful handoff retains the authoritative inputs, transformation rules, intermediate outputs, and unresolved discrepancies, so the successor can continue without repeating completed processing.

Rule auditing and compliance. The agent checks records or artifacts against eligibility rules, permissions, thresholds, or contractual constraints. Typical outputs identify violations and the evidence supporting each decision. The memory must preserve the applicable rule version, exceptions, checked cases, and outstanding evidence; an omitted exception can invalidate an otherwise plausible result.

Operational coordination and resource management. The agent combines availability, demand, and constraints to manage schedules, inventories, deliveries, or staffing requirements. The handoff must preserve current commitments, unresolved conflicts, resource limits, and changes already applied to the workspace. This family highlights the importance of distinguishing completed decisions from proposed actions.

Technical diagnosis and artifact maintenance. The agent inspects logs, dependencies, configurations, or other technical artifacts to identify faults or produce corrective outputs. The memory should retain observed symptoms, tested hypotheses, modifications, and remaining validation checks. Such information connects the current workspace to the reasoning needed for the next action.

A.3 Memory Construction Objectives

The four constructors use the same visible interaction prefix but organize it around different information priorities. `generic_summary` provides a general account of progress without enforcing a task-state schema. `structured_state` separates goals, completed work, artifacts, constraints, open issues, and next actions. `evidence_preserving` prioritizes concrete tool observations, outputs, errors, and facts needed to justify later decisions. `causal_recovery` emphasizes the chain from attempted action to observed consequence to revised plan. All memories are ordinary text; no hidden activation, KV cache, or source-agent state is transferred.

The nominal budget is derived from the visible source-history allowance and applied as an upper bound during construction. The harness records source-history tokens, target tokens, actual memory tokens, API prompt/completion tokens, construction latency, retry count, and any post-hoc truncation. This allows the analysis to distinguish an intended compression rate from the memory actually delivered to the successor.

A.4 Checkpoint and Continuation Lifecycle

For each interruption, the harness identifies the source prefix, replays its environment-affecting actions, and materializes a checkpointed workspace. A continuation is infrastructure-valid only when the restored workspace exists, a continuation trace is present, and execution reaches a terminal recorded state. Scoring occurs after continuation in a separate verifier step. A verifier record is valid only when scoring completes without an infrastructure error and returns a usable pass outcome. Invalid infrastructure rows remain in the coverage audit but are excluded from behavioral denominators.

The supplied entry points use a 9B judge configuration where judge-backed verification is needed. Continuation and scoring can be resumed without overwriting completed jobs. Replay mismatch metadata, termination reason, elapsed time, and token counts are retained for diagnosis. The main analysis does not silently convert missing or crashed verifiers into behavioral failures.

A.5 Verifier Record and Snapshot Audit

The verification design uses workspace grading and hybrid checks. Structured outputs are parsed and checked by code: the relevant questions include whether required artifacts exist, whether their fields satisfy the expected schema, and whether extracted values meet task-specific constraints or numerical tolerances. Semantic criteria apply to unstructured artifacts, such as reports, summaries, or messages, and are assessed by an LLM against the task requirements. This division assigns exact checks to executable validators and meaning-dependent checks to the semantic judge. The grader combines the resulting criterion scores according to the task rubric and reports an aggregate score, maximum score, and pass outcome.

For resumption evaluation, grading is applied after the successor finishes, against the original task requirements. The source and resumed runs are therefore assessed on the same completion objective. A fluent memory or final response alone is insufficient when required workspace artifacts remain incorrect or incomplete. The normalized score captures partial satisfaction of the rubric, while the pass outcome records whether the full task meets its acceptance criterion.

Scoring records expose an aggregate pass indicator and a normalized score, together with criterion-level outcomes when available. The analysis uses the reported pass flag; if it is absent, a score ratio of at least 0.999 supplies the fallback pass decision. All 32,382 verifier-valid records carry `needs_judge=true`, and the launch configuration specifies a 9B judge. This flag records judge involvement at the configuration or task level; it does not identify the evaluation mechanism of each individual criterion.

The snapshot audit records replay observation mismatches and initial/restored file counts. Within each source checkpoint these recorded properties are stable across memory conditions. File-count agreement measures inventory consistency; it does not establish content identity. Replay-matched and mismatch subsets are reported separately in Appendix F.

B METRIC DEFINITIONS

Let $\mathcal{I}_V$ be the set of verifier-valid continuation runs and let $y_i \in \{0, 1\}$ be the verifier pass for run i. Retained success is

$$\widehat{S} = \frac{1}{|\mathcal{I}_V|} \sum_{i \in \mathcal{I}_V} y_i. \tag{3}$$

When the verifier exposes a graded score v_i and maximum $v_i^{\max}$, the per-run score ratio is $r_i = v_i / v_i^{\max}$, and the reported verifier score is the mean of r_i over valid runs. If the verifier supplies no explicit pass flag, a score ratio of at least 0.999 is treated as passing.

For source-history length L_i and realized memory length M_i, the realized memory ratio is M_i/L_i. Budget utilization is M_i/B_i, where B_i is the allowed memory-token budget. These quantities are distinct: the nominal 10%, 30%, and 50% labels define B_i, while all length analyses use M_i.

For an end-to-end handoff, token cost can be decomposed as

$$T_i^{\mathrm{E2E}} = T_i^{\mathrm{comp,prompt}} + T_i^{\mathrm{comp,completion}} + T_i^{\mathrm{cont,prompt}} + T_i^{\mathrm{cont,response}}. \tag{4}$$

The end-to-end ledger combines compression means over all constructed memories in a cell with continuation means over continuation-valid runs in that cell, then averages the 324 cells equally. It therefore differs slightly from a pooled per-run estimator and uses slightly different validity populations for its two components. Memory tokens are part of the compressor output and are consumed within continuation prompts; they are not added as a separate fifth component. The ledger covers memory construction and successor inference, excluding source collection, verifier inference, and environment execution.

We summarize observed recovery effort using four quantities:

$$\mathbf{D}_i = \left(N_i^{\mathrm{turn}}, N_i^{\mathrm{tool}}, T_i^{\mathrm{cont}}, \mathbb{1}[\text{limit exhausted}]\right). \tag{5}$$

This vector measures work performed after handoff. A scalar recovery-debt measure can later weight its components by latency or monetary cost, but the vector preserves interpretable units and avoids choosing deployment-specific prices in the benchmark definition.

C COVERAGE AND VALIDITY AUDIT

Table 4 shows that the full planned grid was discovered and that infrastructure loss is negligible. Behavioral success uses the 32,382 verifier-valid rows as its denominator. The 13,835 valid non-passing runs are genuine behavioral outcomes, not missing records.

Table 4: Coverage and validity audit for the main study.

Outcome	Count	Fraction of planned
Planned/discovered rows	32,400	100.00%
Continuation-valid rows	32,386	99.96%
Verifier-valid rows	32,382	99.94%
Verifier-passing rows	18,547	57.24%
Verifier-valid non-passing rows	13,835	42.70%
Continuation-invalid rows	14	0.04%
Verifier-invalid rows	18	0.06%

D CLUSTERED UNCERTAINTY INTERVALS

The Wilson interval in the main text describes the pooled binary proportion. Table 5 reports 95% intervals from resampling the 300 source trajectories. All runs derived from a sampled source are kept together, preserving their dependence. These intervals reflect variation across source trajectories. Appendix F additionally groups sources sharing a task identifier.

Table 5: Trajectory-clustered bootstrap 95% intervals for verifier success. Point estimates shown here are cluster-weighted bootstrap means and can differ slightly from pooled marginals in Table 2.

Factor	Setting	Mean (%)	95% CI (%)
Overall	All	57.26	[54.09, 60.39]
Stage	30%	49.38	[45.70, 52.95]
	50%	54.60	[51.05, 58.06]
	70%	67.77	[64.56, 70.86]
Compressor	2B	54.22	[51.00, 57.48]
	4B	57.96	[54.80, 61.12]
	9B	59.59	[56.40, 62.82]
Method	Generic	57.94	[54.66, 61.19]
	Causal	57.75	[54.54, 60.86]
	Evidence	57.61	[54.37, 60.83]
	Structured	55.72	[52.40, 58.93]
Budget	10%	56.81	[53.56, 60.03]
	30%	57.69	[54.56, 60.79]
	50%	57.26	[54.08, 60.44]

E Memory Method by Budget

Table 6 provides the interaction behind the aggregate budget result. The table reinforces two observations. First, actual memory length differs substantially across methods under the same nominal label: at 50%, generic summary averages 546 tokens while structured state averages 911. Second, additional tokens have method-dependent value. Causal and evidence-preserving memories improve moderately with budget, generic summaries saturate early, and structured state degrades at its largest realized memory.

Table 6: Memory method by nominal budget. Tokens are realized memory length; success and score are means over verifier-valid rows.

Method	Budget	Memory tokens	Success (%)	Score ratio
Causal recovery	10%	364	56.78	0.825
	30%	641	58.10	0.837
	50%	726	58.43	0.842
Evidence preserving	10%	376	55.88	0.823
	30%	713	58.24	0.844
	50%	817	58.78	0.848
Generic summary	10%	326	58.26	0.832
	30%	507	57.80	0.838
	50%	546	57.84	0.839
Structured state	10%	389	56.43	0.821
	30%	786	56.69	0.833
	50%	911	54.10	0.828

F Task Weighting and Replay Sensitivity

The 300 source trajectories correspond to 151 unique task identifiers. Fifty-five tasks occur in one source stratum, 43 in two strata, and 53 in all three. The pooled estimator in the main text weights a task according to the number of valid continuation conditions derived from it. Equal-weighting the 151 task identifiers gives a task-macro success of 52.52% (task-bootstrap 95% CI [48.27, 56.63]), while resampling whole task clusters for the pooled estimator gives [52.64, 61.62]. We retain the pooled estimator as the primary measure and report task-macro success as a weighting sensitivity.

Table 7: Replay sensitivity and task weighting.

Analysis	Valid runs / tasks	Success (%)
All verifier-valid runs	32,382	57.28
Task-macro (151 tasks)	151	52.52
Replay matched	25,987	61.83
Replay mismatch	6,395	38.76

Within the no-mismatch subset, success is 59.26%, 62.35%, and 63.88% for 2B, 4B, and 9B compressors. The corresponding 10%, 30%, and 50% budget values are 61.824%, 61.826%, and 61.842%. Thus, the compressor trend remains visible after conditioning on replay consistency, while nominal budget remains nearly flat.

G Compression Token Ledger

The compression logs contain one completed record for each of the 32,400 factorial conditions. We stream these JSONL records and select the same rank-preferred record used by the run analysis (completed status, API prompt-token availability, then timestamp). No conversation, compressed-memory, or continuation-history body is needed for this audit. Table 8 reports the principal operating points. "Compression" is the sum of API prompt and completion tokens; "E2E" adds the continuation-token mean from the corresponding factorial cell. These are observed token counts, not prices, and the E2E values are condition-level means.

Table 8: Observed token ledger by compressor scale, nominal budget, and interruption stage. Compression is API prompt plus completion tokens; E2E adds continuation tokens. The final column averages the compression/E2E ratio within factorial cells, which differs from the ratio of the overall means (4.84%).

Axis	Level	Compression	Continuation	E2E	Compression share (%)
Compressor	2B	5,078	108,632	113,710	4.78
	4B	5,175	98,279	103,453	5.43
	9B	5,204	97,057	102,261	5.50
Budget	10%	4,923	103,554	108,477	4.88
	30%	5,222	99,934	105,157	5.39
	50%	5,311	100,480	105,791	5.44
Stage	30%	3,018	114,877	117,895	2.63
	50%	5,166	100,603	105,769	5.10
	70%	7,272	88,489	95,761	7.99
Overall	all	5,152	101,323	106,475	5.24

The median compression cost is 4,243 tokens and the 90th percentile is 9,391 tokens. Mean construction cost rises from 4,923 tokens at the 10% allowance to 5,222 at 30% and 5,311 at 50%. Total handoff cost is non-monotonic: the corresponding cell-level means are 108,477, 105,157, and 105,791 tokens. Thus a smaller memory allowance does not necessarily produce a cheaper continuation. Post-hoc truncation occurs in 0.14% of records, while 32.87% report the API finish reason `length`. These fields record different events: reaching the generation limit does not necessarily require further truncation of the returned memory.

H Failure Profile

The failure profile is computed over 13,835 verifier-valid non-passing runs. A normal no-tool-call stop accounts for 81.98% of failures, compared with 14.46% that exhaust the step limit and 3.56% that exhaust the completion-token limit. The score bands are 22.95% with score ratio at least 0.9,

49.87% between 0.5 and 0.9, and 27.18% below 0.5. These score bands overlap with the termination categories. Exactly one failed verifier criterion occurs in 52.17% of failures, two in 19.16%, and three in 9.97%; the remaining failures miss four or more criteria.

Table 9: Verifier-valid failure profile. Score bands and termination categories are separate marginal summaries and should not be added across rows.

Failure attribute	Count	Fraction of failures
Normal no-tool-call stop	11,342	81.98%
Step-limit exhaustion	2,000	14.46%
Completion-token exhaustion	493	3.56%
Score ratio ≥ 0.9	3,175	22.95%
Score ratio 0.5–0.9	6,900	49.87%
Score ratio < 0.5	3,760	27.18%
Exactly one failed criterion	7,218	52.17%
Exactly two failed criteria	2,651	19.16%
Exactly three failed criteria	1,379	9.97%

I Artifact and Reproduction Map

The analysis provides run-level records, summaries for each combination of experimental factors, and marginal summaries for individual factors. Run-level records contain validity, verifier outcomes, and execution measurements. Factor summaries group runs by source/agent stratum, compressor, method, budget, and stage. Marginal summaries provide the values used in the main paper. Separate tables store bootstrap intervals and paired comparisons between conditions derived from the same source.

To reproduce the results, first check that the planned conditions are present, then select the appropriate validity population, compute the factor summaries, and resample whole source or task clusters for uncertainty estimates. Keep validity counts alongside success rates so that missing evaluations remain visible. The accompanying scripts also export exploratory paired comparisons for further analysis.