STATELEDGER: DEPENDENCY-CLOSED ANALYTICAL STATE FOR EXECUTABLE DATA-ANALYSIS REPLAY

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models are increasingly used for multi-step data-analysis workflows in which analytical state accumulates across cleaned tables, repaired schemas, intermediate variables, and derived metrics. When such history is stored or retrieved as undifferentiated text, however, a later computation may omit prerequisite transformations or recover the wrong version of a variable, preventing faithful replay.

We present StateLedger, which selects executable history through dependency closure over versioned analytical state. StateLedger represents prior analysis as dependency-linked state events, resolves the state versions required by target code, recursively recovers their prerequisite producers, and projects the resulting closure back onto a chronological subsequence of whole historical turns. The result is a reduced history intended to preserve the state required for execution.

On 455 stable CoCoNote notebook targets, its source-only instantiation achieves 342 exact replays while selecting 38.4% fewer source characters on average than the full prefix. It substantially outperforms character-budget-matched BM25 and plain static slicing. Its gain over qualified reaching slicing is modest, with notebook-level uncertainty including zero, while prefix-instrumented dynamic recovery is more accurate at similar selected-history cost. With the recent-three guard fixed, adding transitive prerequisite expansion raises exact replay from 25/455 to 339/455, isolating the principal mechanism beyond direct producer recovery. An external notebook-trace pilot and target-code-hidden generation diagnostic provide complementary evidence. Together, these findings support dependency-closed analytical state as a practical unit for repair-free replay from reduced analytical history.

1 INTRODUCTION

Data-analysis workflows accumulate executable state: cleaned tables, repaired columns, filters, intermediate statistics, and revisions of earlier results. A later computation may depend on a chain of these transformations, even when its source mentions only the final table or metric. Executable data-science benchmarks expose such dependencies through code and files (Lai et al., 2023; Huang et al., 2024c; Zhang et al., 2025); LongDS records long analytical trajectories (Xu et al., 2026), CoCoNote mines contextualized data-wrangling code from notebooks (Huang et al., 2024a), and JuNE records development-time notebook executions (Titov et al., 2025). Recovering the history needed to rerun a later computation is a concrete state-management problem within these workflows.

Text-oriented memory retrieves passages that resemble the current question or retains a recent context window. Agent memories also organize observations and reflections for later use (Park et al., 2023; Shinn et al., 2023; Packer et al., 2023; Zhong et al., 2024). These interfaces support semantic relevance, but executable history imposes an additional requirement: selected code must reconstruct the prerequisite state. A table-cleaning cell can be necessary despite little lexical overlap with the target. Conversely, a relevant cell can refer to a column whose construction lies outside the retrieved context. Reusing a variable name after an update introduces a further ambiguity: recovery must follow the version read by that computation.

StateLedger addresses this problem through a versioned analytical-state representation. It records producers, state transitions, and declared dependencies; resolves each prerequisite at its consuming

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.

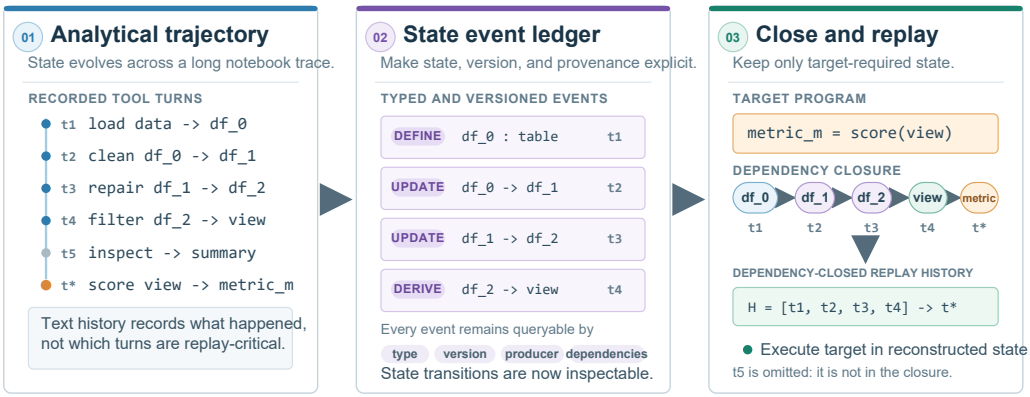


Figure 1: StateLedger recovers a chronological producer subsequence through typed, versioned analytical-state dependencies. The target seeds a backward closure; selected state versions map to executable prior turns. The grey turn remains in the recorded history but is not selected by the dependency closure.

event's version boundary; computes backward dependency closure; and maps the result to prior code segments (Figure 1). This separates the unit of reasoning, a state version, from the unit of execution, a whole historical turn. Program slicing and notebook provenance are the closest executable alternatives (Weiser, 1984; Horwitz et al., 1990; Agrawal & Horgan, 1990; Macke et al., 2021; Shankar et al., 2022). Qualified-symbol slicing provides a precise source-only comparison, while a prefix-instrumented producer control uses observed writes and mutations. Both are evaluated with text retrieval and recency under the same repair-free replay protocol.

Our central claim is that *on stable executable notebook histories, StateLedger's dependency closure over versioned analytical state improves repair-free replay over matched-budget text retrieval and plain static slicing while using substantially less source history than the full prefix. A fixed-guard ablation isolates the gain from recursively recovering prerequisite producers beyond direct producer selection.* The main experiment holds target code fixed and measures whether reduced history reproduces its full-prefix output, reporting exact replay and selected source cost together.

The evidence centers on repair-free CoCoNote replay and a controlled intervention on the recovery mechanism. The ablation holds event induction, target seeding, version resolution, executable projection, and recency fixed, and varies recursive prerequisite expansion. External and downstream checks then examine JuNE notebook traces, repair-assisted LongDS replay, and target-code-hidden generation. Our contributions are:

1. An executable analytical-state replay formulation that measures exact target recovery and source-history cost against a stable full-prefix reference.
2. A version-aware state-to-code selection method, with explicit producer resolution, a least-fixed-point dependency closure, and conditional replay-sufficiency properties.
3. Repair-free evidence that StateLedger recovers 342/455 CoCoNote targets with 38.4% less source history than full prefix, exceeding matched-budget text retrieval and plain static slicing, together with a fixed-guard ablation in which recursive prerequisite expansion increases exact replay from 25/455 to 339/455.

2 Related Work

Agent memory and data-analysis agents. ReAct and Toolformer integrate reasoning with tools, while Reflexion, MemGPT, MemoryBank, and A-Mem organize persistent information for subsequent decisions (Yao et al., 2023; Schick et al., 2023; Shinn et al., 2023; Packer et al., 2023; Zhong et al., 2024; Xu et al., 2025). Data Interpreter uses graph-structured data-science plans (Hong et al., 2025); concurrent MAGE treats memory as execution-state management (Chen et al., 2026). Con-

current Ledger records observed, modified, and attempted repository interactions and governs subsequent coding-agent actions (Wang et al., 2026b). StateLedger instead makes versioned analytical dependencies the interface for recovering a recorded computation's executable prerequisites. Its primary evaluation measures exact replay rather than an agent's complete interaction policy.

Executable data-science benchmarks. DS-1000, DSEval, DSBench, DataSciBench, DA-Code, ARCADE, and BLADE evaluate code generation and analytical tasks (Lai et al., 2023; Zhang et al., 2024; Jing et al., 2025; Zhang et al., 2025; Huang et al., 2024c; Yin et al., 2023; Gu et al., 2024). MLAgentBench, MLE-bench, and CodeFlowBench further examine longer executable workflows (Huang et al., 2024b; Chan et al., 2025; Wang et al., 2026a). We adapt CoCoNote, LongDS, and JuNE traces to measure history recovery under explicit executable boundaries; the resulting replay metrics differ from their original task metrics.

Slicing and notebook state. Static and dynamic slicing recover statements affecting a criterion (Weiser, 1984; Horwitz et al., 1990; Agrawal & Horgan, 1990). Notebook provenance, ProvBook, NBSafety, and nbslicer track dependencies, identify stale state, and gather executable code (Pimentel et al., 2015; Samuel & Konig-Ries, 2018; Head et al., 2019; Macke et al., 2021; Shankar et al., 2022). ElasticNotebook and Kishu reconstruct sessions, while FlowBook tracks dynamic reads and writes for clean-run consistency (Li et al., 2023; 2025; Freund et al., 2026). On code traces, StateLedger's deterministic instantiation is closely related to cell-granular, column-aware reaching-definition analysis. It organizes recovery around versioned analytical-state events: dependencies are resolved at event boundaries, closed recursively, and projected to whole executable turns. The event representation also accepts descriptions induced from public turn text, as in LongDS. We evaluate exact target replay, comparing with qualified slicing and prefix runtime provenance.

3 Executable Analytical-State Replay

A trace contains code turns $c_1, \ldots, c_T$ and public context. For target t, let $P_t = \{1, \ldots, t-1\}$ and let $S \subseteq P_t$ be selected prior turns. Write $\mathrm{Exec}(S; c_t, x_0)$ for execution of the selected code in increasing turn order, followed by c_t, from the same initial external inputs x_0. Let κ be the protocol's output canonicalization, and define

$$z_t(S) = \kappa(\mathrm{Exec}(S; c_t, x_0)), \qquad a_t(S) = \mathbf{1}\{z_t(S) = z_t(P_t)\}, \qquad h(S) = \sum_{i \in S} |c_i|. \tag{1}$$

On execution or canonicalization failure, define $z_t(S) = \perp_{\mathrm{exec}}$, a sentinel distinct from every valid output, so $a_t(S) = 0$. The reference $z_t(P_t)$ is admitted only after full-prefix stability checks. History cost h counts prior source characters, excluding target code and runtime object sizes. The objective is high exact replay with reduced history, rather than a text-compression ratio alone.

Source-only selectors receive preceding source, target source, and protocol-permitted public context. Reference answers, future turns, and reduced-history execution outcomes are excluded from selection. Full-prefix outputs serve evaluation, not retrieval. The separately labeled dynamic control additionally observes execution of the prefix, never of the target. Target-code-hidden diagnostics use a public-question seed and are reported under a separate generation protocol.

4 StateLedger

4.1 Analytical State and Event Induction

An event e carries a state identifier $s(e)$, kind $k(e)$, position $b(e) = (\tau(e), o(e))$, dependency identifiers, composition sources, an optional restoration target, and a nonpropagation indicator $\eta(e)$. Let $\mathcal{E}_{\leq t}$ contain events in turns 1 through t, and E_t those in turn t. Positions have strict lexicographic order: $b(e) \prec b(e')$ when $\tau(e) < \tau(e')$, or when the turns agree and $o(e) < o(e')$. Ties within a turn follow original event-list order. A *state version* is a producer event at this position; repeated state names can therefore denote different versions.

In the primary CoCoNote experiment, event induction is deterministic. An AST pass extracts variable reads and writes and recognized column accesses. State identifiers have the form `var:name`

or `col:base:key`; writing a column also updates its base variable. Each written state yields a definition on first occurrence and an update thereafter, with the cell's read states as dependencies after removing the written identifier. A cell without tracked writes yields a composition event for its read dependencies. This instantiation uses variable/column definitions, updates, and composition without learned extraction. LongDS instead uses a frozen prompt over public turn text to induce the common schema, including transition fields; learned-induction sensitivity is reported in `supplement_event_induction.pdf`.

CoCoNote induction is cell-level rather than statement-level. Events within a cell are emitted in sorted state-identifier order using cell-wide read sets: $o(e)$ defines deterministic positions for ledger resolution, not intra-cell statement provenance. Replay executes each selected cell intact in its original statement order. The effects of unrepresented intra-cell dependencies are measured by exact executable replay; the conditional sufficiency result below applies when the induced provenance satisfies its read-level assumptions.

4.2 Resolving Versions and Prerequisites

Let $\mathcal{C}$ contain the consumer kinds `STATE_REFERENCE` and `ANSWER_USE`, and $\mathcal{B}$ the barrier kinds `DEPRECATE_STATE`, `NONPROPAGATE`, and `BRANCH_CLOSE`. For identifier s and boundary b, first find the latest nonconsumer $\bar{e}_s(b) = \max_{\prec}\{e \in \mathcal{E}_{\le t} : s(e) = s,\ b(e) \prec b,\ k(e) \notin \mathcal{C}\}$, with $\bar{e}_s(b) = \perp$ for an empty set. Version resolution is

$$R(s,b) = \begin{cases} \perp, & \bar{e}_s(b) = \perp, \\ \perp, & k(\bar{e}_s(b)) \in \mathcal{B}, \\ \perp, & \eta(\bar{e}_s(b)) = 1 \text{ and } \tau(\bar{e}_s(b)) < \tau(b), \\ \bar{e}_s(b), & \text{otherwise.} \end{cases} \tag{2}$$

Here $\tau(b)$ is the turn component of b. A barrier terminates lookup rather than exposing an older producer. A flagged producer remains available earlier in the same turn, whereas a consumer never becomes a producer. Thus resolution enforces availability and version boundaries together.

Let $A(e)$ be the union of query identifiers obtained by normalizing the dependency, composition-source, and restoration-target fields under the parser contract in Appendix B. The implementation uses the following producer dependencies and target seed identifiers:

$$\begin{aligned} D(e) &= A(e) \cup \begin{cases} \{s(e)\}, & k(e) = \texttt{STATE_UPDATE}, \\ \varnothing, & \text{otherwise}, \end{cases} \\ U(e) &= D(e) \cup \begin{cases} \{s(e)\}, & k(e) \in \{\texttt{STATE_REFERENCE}, \texttt{ANSWER_USE}\}, \\ \varnothing, & \text{otherwise}. \end{cases} \end{aligned} \tag{3}$$

Empty identifiers are omitted. An update's self-dependency queries its preceding version and retains that producer when resolution succeeds, including for a complete overwrite. Other target events seed prerequisites without unconditionally seeding their own output identifier.

4.3 Closure and Executable Projection

The traversal unit is a *version query* $q = (s,b)$, not an unversioned state name. Let $\mathcal{Q}_t$ be the finite product of state and normalized dependency identifiers with event boundaries in $\mathcal{E}_{\le t}$, omitting empty identifiers but including those with no producer. Write $\rho(s,b) = R(s,b)$. A resolved query generates prerequisite queries at its producer's boundary:

$$\begin{aligned} \mathcal{Q}_t^{(0)} &= \{(s, b(e)) : e \in E_t,\ s \in U(e)\}, \\ \Psi(q) &= \begin{cases} \{(s', b(v)) : s' \in D(v)\}, & v = \rho(q) \neq \perp, \\ \varnothing, & \rho(q) = \perp. \end{cases} \end{aligned} \tag{4}$$

On the finite complete lattice $(2^{\mathcal{Q}_t}, \subseteq)$, define the seeded, monotone, inflationary operator

$$\begin{aligned} \mathcal{T}_t(W) &= \mathcal{Q}_t^{(0)} \cup W \cup \bigcup_{q \in W} \Psi(q), \\ W_t^\star &= \mathrm{lfp}(\mathcal{T}_t), \qquad C_t^\star = \{\rho(q) : q \in W_t^\star,\ \rho(q) \neq \perp\}. \end{aligned} \tag{5}$$

Iterating from $\mathcal{Q}_t^{(0)}$ reaches $W_t^\star$; Algorithm 1 implements this closure with a query-deduplicated stack. Let $Q_t = \{\rho(q) : q \in \mathcal{Q}_t^{(0)}, \rho(q) \neq \bot\}$ denote direct target producers. Target-internal producers can be traversed, but only prior turns enter the executable projection:

$$\begin{aligned} \pi_t(C) &= \{\tau(v) : v \in C,\ \tau(v) < t\}, \qquad S_t^{\text{base}} = \pi_t(C_t^\star), \\ S_t^{(r)} &= S_t^{\text{base}} \cup G_{t,r}, \qquad G_{t,r} = \{\max(1, t-r), \ldots, t-1\}. \end{aligned} \tag{6}$$

where r is a nonnegative integer and $G_{t,0} = \varnothing$. The primary configuration fixes $r = 3$ as a guard for local code not captured by events; the ablation evaluates $r = 0$ separately. Selected turns execute once in chronological order, followed by the target.

Algorithm 1: Version-aware dependency-closed selection.
Input: events $E_{\leq t}$, target t, guard r. *Output:* selected prior turns S.
1. Index. Sort events by turn and original order; index versions by state identifier. Initialize $S = \varnothing$, an empty visited-query set, and a stack containing $\mathcal{Q}_t^{(0)}$.
2. Resolve. Pop (s, b). If visited, skip it; otherwise mark it visited and resolve $v = R(s, b)$. If $v = \bot$, continue.
3. Expand. Add $\tau(v)$ to S when $\tau(v) < t$. Push $(s', b(v))$ for every $s' \in D(v)$. Repeat steps 2–3 until the stack is empty.
4. Project. Return $S \cup G_{t,r}$ in chronological order. Repair-free replay executes this fixed selection without error-triggered additions.

4.4 From Versions to Replayed Code

The following illustrative trace makes the role of the version boundary explicit. Each row is a whole executable turn; the final row is the target.

Turn	Source	Produced state
1	`raw = [2, 4, 8]`	`raw`
2	`clean = raw[:2]`	first version of `clean`
3	`summary = sum(clean)`	`summary`
4	`clean = [100]`	later version of `clean`
5	`audit = sum(clean)`	`audit`
6	`answer = summary`	target

Writing $v_{s,i}$ for the producer of state s in turn i, target seeding and closure give

$$Q_6 = \{v_{\text{summary},3}\}, \qquad C_6^\star = \{v_{\text{summary},3}, v_{\text{clean},2}, v_{\text{raw},1}\}, \qquad S_6^{\text{base}} = \{1, 2, 3\}. \tag{7}$$

The builtin `sum` is an initial input, not a historical producer. Resolving `clean` at turn 3 finds turn 2, not turn 4: the dependency boundary belongs to the consuming producer, rather than to the final target. Replaying turns 1–3 reconstructs `summary = 6`, and the target reproduces the full-prefix answer. Direct producer recovery selects turn 3 alone and lacks `clean`. The unrelated audit and later update remain outside the base closure. With the optional recent3 guard, turns 4 and 5 are added back; the ablation distinguishes this guard from the dependency mechanism.

4.5 Properties and Runtime Interface

Proposition 1 (Finite least closure and conditional replay sufficiency)**.** *Equation* (5) *converges finitely, and $C_t^\star$ is the unique least dependency-closed producer set containing Q_t. Under complete versioned provenance, deterministic transitions with fixed inputs, and faithful whole-turn realization of all executed reads, $z_t(S_t^{\text{base}}) = z_t(P_t)$ and $a_t(S_t^{\text{base}}) = 1$.*

Strictly decreasing resolution boundaries make the producer graph acyclic. Direct recovery satisfies $\pi_t(Q_t) \subseteq S_t^{\text{base}}$, making closure expansion a controlled structural intervention. Projection and source cost are monotone, so the least closure also minimizes projected cost among closed producer sets containing the same seeds. Appendix B proves these properties and the conditional execution guarantee.

History inclusion and replay. For $r_1 \leq r_2$, $S_t^{(r_1)} \subseteq S_t^{(r_2)}$ and source cost cannot decrease. Exact replay need not follow this ordering: $S \subseteq S' \not\Rightarrow a_t(S) \leq a_t(S')$. Added cells execute whole programs with their own prerequisites and effects. Appendix B gives an executable counterexample; the ablation measures closure and the guard separately.

CoCoNote and JuNE use fixed, repair-free selections. LongDS separately evaluates all support-enabled methods through the same bounded runtime: up to eight ordinary name/column repairs in total and at most one error-triggered static prerequisite expansion, using the remaining ordinary-repair budget afterward. Every addition precedes the target and is answer-blind. Appendix E specifies this shared interface and decomposes its contribution.

5 Experiments

5.1 Data and Execution Protocols

CoCoNote is the primary repair-free benchmark. From its official test split, candidates with at least five preceding code cells yield 1,050 targets from 776 notebooks. Two fresh-process full-prefix executions define the stable boundary: 455 targets are retained before reduced-history comparisons. Every method is evaluated on all 455 in the same environment, with target source visible and benchmark outputs hidden from selection. Retained histories average 12.84 prior cells, versus 26.30 among exclusions; the full accounting and retained–excluded distributions appear in Appendices A and D.

The external JuNE pilot uses the released task1 asset: 76 candidates from 19 sessions, 65 after source-safety screening, and 29 double-stable targets from 13 sessions. Replay respects kernel restarts and continues after historical cell exceptions. LongDS contributes 110 full-prefix-exact targets from a 23-task, 115-target executable extension; its full requested-50 lineage is available in the provenance supplement. LongDS replay is repair-assisted. A separate teacher-forced diagnostic hides target code while retaining the recorded prior trajectory. Appendix protocols distinguish these settings and their denominators.

5.2 Controls, Budgets, and Metrics

Full prefix supplies the executable reference; Recent3 selects the three preceding turns. BM25 (Robertson & Zaragoza, 2009) ranks whole prior turns under a per-target source-character budget $h(S_t^{(3)})$. Greedy inclusion skips a turn if it exceeds the remaining budget, so realized cost can be below the target. Plain static slicing traces variable definitions; the qualified reaching slice additionally tracks scope-qualified definitions. Both include the same recent-three guard. The prefix dynamic-producer control combines source-derived read sets with observed prefix writes and mutations, without executing the target. Its frozen fallback uses qualified slicing when collection fails or encounters unsupported observations. We report this runtime-aware control separately from source-only selection.

A matched budget refers to initial source selection, not identical final character counts. In support-enabled LongDS comparisons, repairs and the shared static expansion can increase cost; reported characters include these additions. No repair is available in the CoCoNote matrix or its direct-producer/closure ablation. Selected turns are deduplicated and executed in trace order.

Exact replay, target execution coverage, selected source characters, and prior cells or segments are the principal measures. Paired comparisons report differences and 95% confidence intervals, with notebook or task clustering where stated; small-sample JuNE tests cluster by session (Appendix I). Target-code-hidden generation uses method-blind dual-LLM semantic consensus, with nonexecution counted as failure. Its judge protocol and strict-exact metric are reported separately in Appendix F.

For CoCoNote, the target observation is its terminal expression or supported assignment value. Canonicalization records normalized type tags and nested values; for DataFrames it also records shape, column/index labels and names, and string representations of dtypes. Exact replay compares these canonical structures, so successful execution with a different canonicalized schema or value

is not an exact replay. Full-prefix failures and unsupported target outputs are accounted for at the shared boundary, independently of each reduced-history selector.

6 RESULTS

6.1 REPAIR-FREE NOTEBOOK REPLAY

Table 1: Repair-free CoCoNote replay on 455 stable targets. Source-only selectors use prior and target source; the runtime-provenance control additionally executes the prefix. Characters and cells measure selected prior history.

Method	Exact	Target exec.	Prior cells	Chars
Full prefix	1.0000	1.0000	12.84	1781.3
Source-only selection				
StateLedger closure + recent3	**0.7516**	**0.7692**	6.81	1097.7
Qualified reaching slice + recent3	0.7099	0.7626	6.70	1078.5
BM25, matched character target	0.3077	0.3407	8.36	1034.4
Plain static slicing + recent3	0.1011	0.1165	4.89	793.6
Recent3	0.0022	0.0022	3.00	423.6
Prefix-instrumented methods				
Prefix dynamic producer + recent3[†]	0.8176	0.8242	6.89	1121.8

[†]Dynamic selection completes on 424/455 targets; the other 31 use the frozen qualified-slicing fallback (Appendix D).

StateLedger reproduces 342/455 targets, and qualified slicing reaches 323/455 (Table 1). StateLedger's target-micro gain is +0.0418 with notebook-clustered 95% interval [0.0022, 0.0818], while the notebook-macro interval [-0.0036, 0.0661] includes zero.

Matched-budget BM25 recovers 140/455 targets and plain static slicing recovers 46/455. The paired gain over plain static slicing is +0.6505 (299 wins, 3 losses, 153 ties), with interval [0.5949, 0.7035]. StateLedger selects 1,097.7 source characters on average, 38.4% fewer than the full prefix's 1,781.3. Together, these results demonstrate recovery of executable prerequisites from reduced source history through the analytical-state representation.

Prefix-instrumented recovery. With observed prefix writes and mutations, the dynamic-producer control reaches 372/455 at 1,121.8 characters. The StateLedger-minus-dynamic difference is -0.0659 (9 wins, 39 losses; interval [-0.1028, -0.0307]). This control is more accurate at similar selected source cost and requires executing the prefix to collect provenance; StateLedger selects from source alone.

6.2 DEPENDENCY CLOSURE ABLATION

Table 2: Frozen CoCoNote ablation on the same 455 targets, in a separate repair-free run. The first two rows isolate closure with recent3 fixed; the third tests the guard.

Variant	Exact	Prior cells	Chars
Direct producer + recent3	0.0549	3.53	533.4
Closure + recent3	0.7451	6.81	1097.7
Closure without recent3	**0.8000**	5.18	899.0
Recent3	0.0022	3.00	423.6

The first two variants share event induction, target queries, version resolution, whole-turn projection, and the recent-three guard. The intervention is recursive expansion from direct producers to their prerequisites, corresponding to $\pi_t(Q_t) \subseteq \pi_t(C_t^\star)$ in Section 4. Adding this expansion increases exact replay from 25/455 to 339/455: +0.6901, notebook-clustered interval [0.6393, 0.7409]. Closure without recent3 reaches 364/455 with 899.0 characters, separating closure's benefit from the guard.

Closure + recent3 minus closure alone is -0.0549, with interval [-0.1116, 0.0000]. Appendix D provides run-level details.

6.3 Complementary Checks and Boundaries

Table 3: Complementary evidence with setting-specific protocols, metrics, and denominators. Full comparisons appear in Appendices C, E, and F, respectively.

Setting	Protocol / metric	StateLedger	Controls
JuNE pilot	Repair-free exact replay	28/29	Plain static 15/29; qualified 25/29
LongDS	Repair-assisted exact replay	106/110	Qualified 109/110; BM25 108/110
Teacher-forced live	Target code hidden; semantic consensus	44/110	Matched BM25 28/110; full prefix 54/110

JuNE provides a one-task pilot of recovery in external development-time notebook traces. StateLedger reaches 28/29 exact versus 15/29 for plain static slicing and 4/29 for Recent3; the static comparison has 13 wins, no losses, and session-cluster sign-flip $p = 0.0078$. Qualified slicing reaches 25/29; its comparison interval [-0.0800, 0.2821] includes zero (Appendix C).

Repair-assisted LongDS exposes a different accuracy–history ordering: StateLedger reaches 106/110 with 18,825.5 characters, while qualified slicing reaches 109/110 with 20,420.8 and BM25 reaches 108/110 with 19,790.9. The StateLedger-minus-qualified task-cluster interval is [-0.0784, 0.0174]. StateLedger's replay counts progress from 50/110 without repair to 102/110 with ordinary repair and 106/110 with shared static support. Appendix E gives the full matrix and support-trigger rates.

Teacher-forced live generation tests whether recovered history helps when target code is hidden, with the recorded history fixed and at most three execution-feedback actions. StateLedger's semantic-consensus gain over BM25 is +0.1455, task-cluster interval [0.0714, 0.2389] (Appendix F); Table 3 also reports full-prefix generation. A method-blind human audit of 32 pairs, stratified by the prior LLM-judged outcomes, yields 62/64 annotator agreement ($\kappa = 0.937$); sampling and descriptive outcomes appear in Appendix G.

7 Discussion and Limitations

Recovering an immediate producer is often insufficient to reconstruct the state a target needs. The fixed-guard ablation demonstrates the value of recursively recovering that producer's prerequisites, and the unguarded condition shows that successful closure-based replay persists without a recent-history buffer. CoCoNote's main comparison establishes this recovery benefit over text retrieval and plain static slicing while reducing source history relative to full prefix.

CoCoNote and LongDS expose different recovery settings. LongDS combines text-induced events with repair-assisted execution, while CoCoNote uses deterministic cell-level events and fixed, repair-free selections. Dataset composition, event induction, and repair therefore vary together with notebook versus script structure. Their individual contributions could be distinguished by applying matched extractors and execution protocols across both trace types.

Target-known replay isolates history recovery, and the teacher-forced diagnostic extends the interface to generation from a fixed recorded history. Fully online transfer showed no clear advantage over BM25 (Appendix H); it adds evolving agent decisions to the recovery problem. Broader execution environments and online state induction are natural directions for extending the interface. The current replay population is defined by full-prefix stability, with retention favoring shorter histories as documented in Section 5.

8 Conclusion

StateLedger selects executable history through dependency closure over versioned analytical state. On stable CoCoNote notebook targets, it improves repair-free replay over matched text retrieval and plain static slicing while selecting 38.4% less source history than full prefix on average. The fixed-guard ablation isolates the gain from transitive prerequisite expansion beyond direct producer recovery. Together, these results support analytical state as a practical unit for reduced-history replay.

AI Use Statement

Generative AI tools acted as simulated reviewers, providing feedback on manuscript organization, clarity, methodology, experiment design, and interpretation, and assisted manuscript revisions. They also assisted in formulating mathematical claims and developing and checking proofs, as well as implementation, debugging, data preparation, artifact checks, and literature triage. The authors approved the protocols and claims and checked reported experiments against logs, tests, and hash-bound records. Model-generated judgments are identified separately from human annotations. The authors take responsibility for the paper and its artifacts.

Reproducibility Statement

The anonymous package contains sanitized source and tests, frozen public protocols, and aggregate results for the reported analyses. Appendix B gives the conditional proof and implementation correspondence; Appendices A–D document replay boundaries; later appendices specify LongDS, generation, and human-audit protocols. The complete requested-50 audit is supplied as `supplement_provenance.pdf` and machine-readable `requested50.csv`. Learned-extractor results are supplied separately in `supplement_event_induction.pdf`, with development provenance in `event_induction_provenance.md`. Benchmark data and licensed environment assets must be obtained separately. Annotator identities, raw private scorer inputs, reference-bearing judge payloads, and runtime credentials are excluded from the anonymous package.

References

Hiralal Agrawal and Joseph R. Horgan. Dynamic program slicing. In *Proceedings of the ACM SIGPLAN 1990 Conference on Programming Language Design and Implementation*, pp. 246–256, 1990. doi: 10.1145/93542.93576. URL https://doi.org/10.1145/93542.93576.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Aleksander Madry, and Lilian Weng. MLE-bench: Evaluating machine learning agents on machine learning engineering. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=6s5uXNWGIh.

Yaoqi Chen, Haibin Lai, Yuru Feng, Chuyu Han, Qianxi Zhang, Baotong Lu, Menghao Li, Xinjiang Wang, Zhirui Wang, Shusen Xu, Zengzhong Li, Zewen Jin, Hao Wu, Cheng Li, and Qi Chen. Beyond semantic organization: Memory as execution state management for long-horizon agents, 2026. URL https://arxiv.org/abs/2606.06090.

Stephen N. Freund, Emery D. Berger, Cormac Flanagan, and Eunice Jun. FlowBook: Enforcing reproducibility in computational notebooks, 2026. URL https://arxiv.org/abs/2605.01560.

Ken Gu, Ruoxi Shang, Ruien Jiang, Keying Kuang, Richard-John Lin, Donghe Lyu, Yue Mao, Youran Pan, Teng Wu, Jiaqian Yu, Yikun Zhang, Tianmai M. Zhang, Lanyi Zhu, Mike A. Merrill, Jeffrey Heer, and Tim Althoff. BLADE: Benchmarking language model agents for data-driven science. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 13936–13971, Miami, Florida, USA, 2024. Association for Computational Linguistics. doi: 10.18653/v

1/2024.findings-emnlp.815. URL https://aclanthology.org/2024.findings-emnlp.815/.

Andrew Head, Fred Hohman, Titus Barik, Steven M. Drucker, and Robert DeLine. Managing messes in computational notebooks. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pp. 1–12, 2019. doi: 10.1145/3290605.3300500. URL https://doi.org/10.1145/3290605.3300500.

Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Li Zhang, Lingyao Zhang, Min Yang, Mingchen Zhuge, Taicheng Guo, Tuo Zhou, Wei Tao, Robert Tang, Xiangtao Lu, Xiawu Zheng, Xinbing Liang, Yaying Fei, Yuheng Cheng, Yongxin Ni, Zhibin Gou, Zongze Xu, Yuyu Luo, and Chenglin Wu. Data Interpreter: An LLM agent for data science. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 19796–19821. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-acl.1016. URL https://aclanthology.org/2025.findings-acl.1016/.

Susan Horwitz, Thomas Reps, and David Binkley. Interprocedural slicing using dependence graphs. *ACM Transactions on Programming Languages and Systems*, 12(1):26–60, 1990. doi: 10.1145/77606.77608. URL https://doi.org/10.1145/77606.77608.

Junjie Huang, Daya Guo, Chenglong Wang, Jiazhen Gu, Shuai Lu, Jeevana Priya Inala, Cong Yan, Jianfeng Gao, Nan Duan, and Michael R. Lyu. Contextualized data-wrangling code generation in computational notebooks. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2024a. doi: 10.1145/3691620.3695503. URL https://doi.org/10.1145/3691620.3695503.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. MLAgentBench: Evaluating language agents on machine learning experimentation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 20271–20309. PMLR, 2024b. URL https://proceedings.mlr.press/v235/huang24y.html.

Yiming Huang, Jianwen Luo, Yan Yu, Yitong Zhang, Fangyu Lei, Yifan Wei, Shizhu He, Lifu Huang, Xiao Liu, Jun Zhao, and Kang Liu. DA-Code: Agent data science code generation benchmark for large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024c. URL https://aclanthology.org/2024.emnlp-main.748/.

Liqiang Jing, Zhehui Huang, Xiaoyang Wang, Wenlin Yao, Wenhao Yu, Kaixin Ma, Hongming Zhang, Xinya Du, and Dong Yu. DSBench: How far are data science agents from becoming data science experts? In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=DSsSPr0RZJ.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-Tau Yih, Daniel Fried, Sida Wang, and Tao Yu. DS-1000: A natural and reliable benchmark for data science code generation. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 18319–18345. PMLR, 2023. URL https://proceedings.mlr.press/v202/lai23b.html.

Zhaoheng Li, Pranav Gor, Rahul Prabhu, Hui Yu, Yuzhou Mao, and Yongjoo Park. ElasticNotebook: Enabling live migration for computational notebooks. *Proceedings of the VLDB Endowment*, 17(2):119–133, 2023. doi: 10.14778/3626292.3626296. URL https://doi.org/10.14778/3626292.3626296.

Zhaoheng Li, Supawit Chockchowwat, Ribhav Sahu, Areet Sheth, and Yongjoo Park. Kishu: Time-traveling for computational notebooks. *Proceedings of the VLDB Endowment*, 18(4):970–985, 2025. doi: 10.14778/3717755.3717759. URL https://doi.org/10.14778/3717755.3717759.

Stephen Macke, Hongpu Gong, Doris Jung-Lin Lee, Andrew Head, Doris Xin, and Aditya Parameswaran. Fine-grained lineage for safer notebook interactions. *Proceedings of the VLDB*

Endowment, 14(6):1093–1101, 2021. doi: 10.14778/3447689.3447712. URL https://doi.org/10.14778/3447689.3447712.

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

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*, 2023. doi: 10.1145/3586183.3606763. URL https://dl.acm.org/doi/10.1145/3586183.3606763.

Joao Felipe Pimentel, Vanessa Braganholo, Leonardo Murta, and Juliana Freire. Collecting and analyzing provenance on interactive notebooks: When IPython meets noWorkflow. In *Proceedings of the 7th USENIX Workshop on the Theory and Practice of Provenance*, 2015. URL https://www.usenix.org/conference/tapp15/workshop-program/presentation/pimentel.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009. doi: 10.1561/1500000019. URL https://doi.org/10.1561/1500000019.

Sheeba Samuel and Birgitta Konig-Ries. ProvBook: Provenance-based semantic enrichment of interactive notebooks for reproducibility. In *Proceedings of the ISWC 2018 Posters and Demonstrations, Industry and Blue Sky Ideas Tracks*, volume 2180 of *CEUR Workshop Proceedings*, 2018. URL https://ceur-ws.org/Vol-2180/paper-57.pdf.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2302.04761.

Shreya Shankar, Stephen Macke, Sarah Chasins, Andrew Head, and Aditya Parameswaran. Bolt-on, compact, and rapid program slicing for notebooks. *Proceedings of the VLDB Endowment*, 15(13): 4038–4047, 2022. doi: 10.14778/3565838.3565855. URL https://doi.org/10.14778/3565838.3565855.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=vAElhFcKW6.

Sergey Titov, Konstantin Grotov, Cristina Sarasua, Yaroslav Golubev, Dhivyabharathi Ramasamy, Alberto Bacchelli, Abraham Bernstein, and Timofey Bryksin. Observing fine-grained changes in jupyter notebooks during development time, 2025. URL https://arxiv.org/abs/2507.15831.

Sizhe Wang, Zhengren Wang, Dongsheng Ma, Yongan Yu, Rui Ling, Zhiyu Li, Feiyu Xiong, and Wentao Zhang. CodeFlowBench: A multi-turn, iterative benchmark for complex code generation. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4369–4402, San Diego, California, United States, 2026a. Association for Computational Linguistics. doi: 10.18653/v1/2026.acl-long.201. URL https://aclanthology.org/2026.acl-long.201/.

Zehao Wang, Yisen Xu, Chenglin Li, Chao Peng, Bram Adams, Ahmed E. Hassan, and Tse-Hsun Chen. Turning interaction history into execution state: A runtime layer for long-horizon coding agents, 2026b. URL https://arxiv.org/abs/2608.00808.

Mark Weiser. Program slicing. *IEEE Transactions on Software Engineering*, SE-10(4):352–357, 1984. doi: 10.1109/TSE.1984.5010248. URL https://doi.org/10.1109/TSE.1984.5010248.

Kewei Xu, Xiaoben Lu, Shuofei Qiao, Zihan Ding, Haoming Xu, Lei Liang, and Ningyu Zhang. LongDS-Bench: On the failure of long-horizon agentic data analysis, 2026. URL https://arxiv.org/abs/2605.30434.

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, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/19909c36f51abc4856b4560aff3d36d6-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 *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Pengcheng Yin, Wen-Ding Li, Kefan Xiao, Abhishek Rao, Yeming Wen, Kensen Shi, Joshua Howland, Paige Bailey, Michele Catasta, Henryk Michalewski, Oleksandr Polozov, and Charles Sutton. Natural language to code generation in interactive data science notebooks. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 126–173, Toronto, Canada, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.9. URL https://aclanthology.org/2023.acl-long.9/.

Dan Zhang, Sining Zhoubian, Min Cai, Fengzu Li, Lekang Yang, Wei Wang, Tianjiao Dong, Ziniu Hu, Jie Tang, and Yisong Yue. DataSciBench: An LLM agent benchmark for data science, 2025. URL https://arxiv.org/abs/2502.13897.

Yuge Zhang, Qiyang Jiang, Xingyu Han, Nan Chen, Yuqing Yang, and Kan Ren. Benchmarking data science agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://aclanthology.org/2024.acl-long.308/.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024. URL https://ojs.aaai.org/index.php/AAAI/article/view/29946.

A Dataset Accounting and Visibility

Table 4: Method-independent benchmark accounting.

Benchmark stage	Tasks/notebooks	Candidates	Kept	Boundary
CoCoNote official-test adaptation	776 notebooks	1050	455	595 full-prefix failures
LongDS requested lineage	50 tasks	–	35	10 missing; 5 unstable-rule skips
LongDS executable extension	23 tasks	115	110	5 full-prefix mismatches
JuNE task1 external pilot	19 sessions	76	29	11 safety; 36 full-prefix failures
LongDS fully online confirmation	40 tasks	200/method	200/method	all frozen ITT rows scored

For both replay benchmarks, candidate rules and stable-boundary checks are method-independent. Selectors see only information preceding the target plus target source when required by the replay definition. Future turns, reference answers, competing selections, and method outcomes are forbidden. CoCoNote runs every method from a copied notebook directory in a fresh process without repair. Static LongDS replay uses the shared support protocol described in the main text. The fully online confirmation is a distinct 40-task protocol and reports all intent-to-treat rows rather than a full-prefix-exact replay subset.

The complete LongDS requested-50 table is provided as `requested50.csv` and `requested50.tex` in the supplement. It records rank, task id, local availability, original and recovered reference status, final inclusion decision, full-prefix-exact target count, and exclusion reason for every requested task. The five full-prefix boundary targets are also listed separately.

B Conditional Dependency-Closure Guarantee

We first establish properties of the finite representation, then connect it to executable replay. Let $\mathcal{V}_t$ contain the events in $\mathcal{E}_{\le t}$ with nonempty state identifiers and kinds outside $\mathcal{C} \cup \mathcal{B}$. These are potential producer nodes, including target-internal producers; Eq. (2) determines availability at each query boundary. Define

$$\mathrm{dep}(v) = \{R(s, b(v)) : s \in D(v)\} \setminus \{\bot\}, \qquad \Phi_t(X) = Q_t \cup X \cup \bigcup_{v \in X} \mathrm{dep}(v). \tag{8}$$

Version causality and update predecessors. For $u \in \mathrm{dep}(v)$, the strict candidate boundary in Eq. (2) gives $b(u) \prec b(v)$. A directed cycle would imply $b(v) \prec b(v)$ by transitivity, contradicting irreflexivity. Thus the graph with edges $v \to u$ is a DAG. If v is a `STATE_UPDATE` with nonempty identifier s, Eq. (3) adds s to $D(v)$. Whenever $R(s, b(v)) = u \neq \bot$, this places u in $\mathrm{dep}(v)$. The edge preserves the resolved preceding version, not a self-loop on an unversioned name. A barrier can make this predecessor unavailable; consumer events are skipped rather than counted as versions.

Query closure and event closure. The powerset of finite $\mathcal{Q}_t$, ordered by inclusion, is a complete lattice: unions and intersections give arbitrary joins and meets. If $W \subseteq W'$, every successor contributed by W is also contributed by W', so $\mathcal{T}_t(W) \subseteq \mathcal{T}_t(W')$. The explicit union with W makes the operator inflationary. Iterate $W^{(0)} = \mathcal{Q}_t^{(0)}$, $W^{(j+1)} = \mathcal{T}_t(W^{(j)})$. This increasing sequence stabilizes after at most $|\mathcal{Q}_t| - |\mathcal{Q}_t^{(0)}|$ strict expansions. Every fixed point contains the seeds and, inductively, every $W^{(j)}$; hence the stabilized set is $\mathrm{lfp}(\mathcal{T}_t)$.

For the event operator in Eq. (8), the same finite argument yields a least fixed point C_{evt}, reached from Q_t after at most $|\mathcal{V}_t| - |Q_t|$ strict expansions. We now verify that query resolution gives exactly this event closure. At initialization, every resolved query is in $Q_t \subseteq C_{\mathrm{evt}}$. If a query resolves to $v \in C_{\mathrm{evt}}$, each resolved successor belongs to $\mathrm{dep}(v) \subseteq C_{\mathrm{evt}}$. Induction over query expansions therefore gives $C_t^\star \subseteq C_{\mathrm{evt}}$. Conversely, $C_t^\star$ contains Q_t. For every $v \in C_t^\star$, choose a query $q \in W_t^\star$ resolving to v. Fixed-point closure includes every $(s, b(v))$ for $s \in D(v)$, so $\mathrm{dep}(v) \subseteq C_t^\star$. Leastness of C_{evt} then gives $C_{\mathrm{evt}} \subseteq C_t^\star$. Thus

$$C_t^\star = \mathrm{lfp}(\Phi_t) = \bigcap \{X \subseteq \mathcal{V}_t : Q_t \subseteq X,\ \forall v \in X,\ \mathrm{dep}(v) \subseteq X\}. \tag{9}$$

Both operators include their seeds; omitting seeds from an inflationary dependency operator would instead give the empty set as its least fixed point. Unresolved queries remain valid query nodes

but contribute no producer event. These statements establish the finite-closure part of Proposition 1 independently of execution assumptions.

Projection, direct recovery, and restricted cost optimality. Events in the same turn are equivalent under $u \sim_\tau v$ iff $\tau(u) = \tau(v)$. Projection π_t keeps the corresponding prior-turn classes once each. For $C \subseteq C'$, set-image monotonicity and nonnegative source lengths give

$$\pi_t(C) \subseteq \pi_t(C'), \qquad h(\pi_t(C)) \le h(\pi_t(C')). \tag{10}$$

In particular, direct recovery selects $S_t^{\mathrm{dir}} = \pi_t(Q_t) \subseteq S_t^{\mathrm{base}}$. Union with a common guard preserves this inclusion. The direct-versus-closure ablation therefore changes ancestor expansion while keeping its seeds, projection, and guard fixed.

Let $\mathcal{F}_t$ be the family inside the intersection in Eq. (9). Since $C_t^\star \in \mathcal{F}_t$ and $C_t^\star \subseteq X$ for every $X \in \mathcal{F}_t$,

$$h(\pi_t(C_t^\star)) = \min_{X \in \mathcal{F}_t} h(\pi_t(X)). \tag{11}$$

The same argument holds for turn count or a fixed guard added to every projection; ties are possible. This comparison fixes the event graph and seeds. Histories satisfying $a_t(S) = 1$ need not be projections of members of this family: for `x=0`, `x=1`, then target `answer=x`, the UPDATE rule retains both producers, although the second turn alone reproduces the target. Minimum cost in $\mathcal{F}_t$ and minimum cost over all semantically equivalent programs are different optimization problems.

Partial execution semantics. Let Σ include program state and relevant external state. A historical turn induces a partial transition $F_i : \Sigma \to \Sigma_\perp$, where $\Sigma_\perp = \Sigma \cup \{\perp\}$. Extend it by $F_i(\perp) = \perp$. The target induces a partial observation map $H_t : \Sigma_\perp \to \mathcal{O}_\perp$, incorporating target execution and its recorded output, with $H_t(\perp) = \perp$. For $S = \{i_1 < \cdots < i_m\}$, define

$$F_S = F_{i_m} \circ \cdots \circ F_{i_1}, \qquad z_t(S) = \kappa(H_t(F_S(\sigma_0))), \qquad F_\varnothing = \mathrm{id}. \tag{12}$$

Here σ_0 initializes the inputs x_0, $\kappa(\perp) = \perp_{\mathrm{exec}}$, and only the target observation is canonicalized, not the entire final store. This models abort-on-failure replay. A continuation protocol such as JuNE includes historical exception recording and continuation in its turn transition instead (Appendix C).

Execution assumptions. The sufficiency part of Proposition 1 uses these conditions on the admitted, successful full-prefix reference.

1. *Complete versioned provenance.* Every noninitial read needed by the target and its prerequisite producers is represented, and resolution identifies its actual preceding producer version. A required read with no historical producer is an explicitly available initial input. This covers mutable objects and target-relevant external state.
2. *Deterministic transitions and fixed inputs.* Each operation's definedness, control flow, writes, and observations are determined by its complete semantic inputs. Initial data, files, library behavior, and relevant random state agree with the reference. Matching these inputs to a successful reference operation makes the replay operation defined with the same outputs.
3. *Faithful whole-turn realization.* The selected whole turns and the target realize corresponding operations of the reference in their original order. Every executed read, including auxiliary operations and control/definedness inputs, reaches its original writer earlier in the expanded sequence or the same initial input. The correspondence covers all observable effects and introduces no unrepresented target-relevant writes.

The last condition concerns executable projection, not just closure of event identifiers. Cell-level synthetic order is used for resolution; the guarantee requires its resolved versions to correspond faithfully to actual operation occurrences, without assuming that event order reconstructs statement order.

Replay-sufficiency part of Proposition 1. Expand the selected prior turns and target into their operation occurrences. Provenance completeness and event closure include the target's prerequisite versions, while target-internal producers are realized in the target rather than as prior turns. Faithful realization also covers every auxiliary operation introduced by whole-turn expansion. Induct

through this expanded execution, maintaining equality of all writes and observations at corresponding operation occurrences, together with definedness.

For the first occurrence, all reads reach initial inputs: a noninitial read would require an earlier writer by faithful realization. These values match the reference by the fixed-input condition. The reference occurrence is defined, so determinism with matching complete inputs makes the replay occurrence defined and reproduces its outputs. For an arbitrary later occurrence, each read either comes from the same initial input or from the same earlier writer occurrence as in the reference. In the latter case its value agrees by the induction hypothesis, and faithful realization ensures that no intervening write changes the reaching writer. All control and definedness inputs consequently agree. The next operation follows the same control decision, is defined, and produces the same writes and observations. This closes the induction for the selected producers, auxiliary code, and target alike.

Writing $\mathrm{Val}_{S;t}(v)$ for a represented version's value at its corresponding production occurrence during history S followed by the target, the invariant implies

$$\forall v \in C_t^\star, \quad \mathrm{Val}_{S_t^{\mathrm{base}};t}(v) = \mathrm{Val}_{P_t;t}(v), \qquad z_t(S_t^{\mathrm{base}}) = z_t(P_t). \tag{13}$$

The final equality follows from equality of the target observations and the same canonicalization. Since the full-prefix output is valid, Eq. (1) gives $a_t(S_t^{\mathrm{base}}) = 1$. The invariant concerns production occurrences, not the contents of a variable after later overwrites. □

Implementation correspondence. Dependency-field normalization follows the frozen parser. A field is treated as a list of values (a scalar is wrapped in a singleton list); string or numeric entries are converted to text, split on commas, stripped of surrounding whitespace, and empty components are dropped. Other entry types are ignored. $A(e)$ in Eq. (3) is the union of these normalized components from the three fields. Producer keys $s(e)$ are stripped but not comma-split; the self-identifier added for an update or consumer is also unchanged. Consequently, a literal comma-bearing name is not an atomic dependency-field value under this parser. The equations describe these actual lookup keys, while the execution guarantee additionally requires that they recover the true read provenance.

The executable implementation indexes events by state identifier and resolves each query (s, τ, o) independently. The functions `_producer_dependency_ids` and `_target_dependency_ids` implement Eq. (3); `_latest_producer_before` implements R. The stack in `predict_stateledger_turns` visits the corresponding resolved ancestors and projects turns below t. Query-level deduplication preserves traversal when one identifier is needed at several different version boundaries. Skipping consumer events precedes barrier checks. An explicit nonpropagation flag blocks lookup across turns, while a same-turn lookup can still use an earlier flagged producer; the unconditional barrier kinds stop either lookup.

Implementation cost. Let N be the number of supplied events, including any filtered out after sorting, and n_s the length of the retained version list for identifier s. Let I_t include identifier normalization/index materialization and target-seed parsing and insertion. Let $\lambda(v)$ count the work of reparsing v's raw dependency fields, deduplicating them, and inserting its successor queries; it includes at least $1 + |D(v)|$ and the raw field lengths. With unit-cost indexed-key operations, the implemented selector has the bound

$$T_t = O\left(N \log(N+1) + I_t + \sum_{q=(s,b)\in W_t^\star} (1 + n_s) + \sum_{\substack{q \in W_t^\star \\ \rho(q) \neq \perp}} \lambda(\rho(q)) \right). \tag{14}$$

The scan bound counts each visited query once. Stack entries skipped as duplicate queries are covered by the seed and successor insertion costs. Distinct boundaries can resolve to the same producer, so its dependencies can be parsed and expanded repeatedly; the final sum is over queries rather than distinct producers. For example, n target queries at distinct event boundaries can all resolve to a shared producer with n dependencies, requiring n^2 successor insertions even with only $O(n)$ distinct events and dependency entries. A linear traversal bound on an already materialized event graph therefore describes a different implementation. Unit tests cover update predecessors, consumer seeds, future versions, same-turn chains, composition/restoration fields, and propagation barriers.

Guard inclusion does not order replay accuracy. Set inclusion gives $G_{t,r_1} \subseteq G_{t,r_2}$ for $r_1 \leq r_2$, so Eq. (10) also orders the guarded source costs. The following trace witnesses why the corresponding exact indicators need not be ordered, using the current deterministic extractor and selector:

Turn	Source	Turn	Source
1	`aux = 1`	4	`scratch = aux + 1`
2	`unused = 0`	5	`marker = 0`
3	`x = 0`	6	`answer = x` (target)

The base closure selects $\{3\}$ and reproduces the full-prefix answer 0. The recent3 union selects $\{3, 4, 5\}$; turn 4 reads `aux`, whose producer was not added, and abort-on-failure replay fails. Thus $a_6(S_6^{(0)}) = 1$ but $a_6(S_6^{(3)}) = 0$. The guard is a union of turns, not dependency closure of those added turns. This counterexample establishes nonmonotonicity of the implemented guard family; it does not attribute individual benchmark differences to this failure mode. Sufficiency for a guarded execution requires the same whole-turn realization conditions on its expanded sequence.

C JuNE External Trace Pilot

Table 5: External JuNE task1 pilot under fresh-process, Jupyter-continuation, reset-aware replay. It has 29 double-stable targets from 13 sessions; no condition uses repair, fallback, or error-triggered support.

Method	Exact	Prior turns	Chars
Full prefix	**1.0000**	57.21	12679.0
StateLedger closure + recent3	0.9655	32.52	11038.8
StateLedger closure	0.9310	30.79	10828.1
Qualified symbol-reaching slice + recent3	0.8621	29.97	10827.6
Static slicing + recent3	0.5172	28.10	10593.6
Recent3	0.1379	3.00	724.9

JuNE records fine-grained notebook-development events (Titov et al., 2025). Task1 has a released local reproduction asset. Candidate formation reads code execution sources and timestamps, without stored outputs, reference values, or method outcomes. The fixed set contains 76 source-only candidates from 19 sessions. A source-safety screen excludes 11 (five write calls, four unavailable `regexp` imports, and two misspelled `matplot` imports), leaving 65 cases for the executable boundary audit.

The executor uses Jupyter-continuation semantics: historical cell exceptions are recorded and execution continues, while code before the latest recorded kernel restart is excluded. Two independent full-prefix runs retain 29 stable targets across 13 sessions. The other 36 fail the boundary (19 target name errors, 11 timeouts, and six other target/canonicalization failures). Full prefix reuses its two gate runs; each reduced-history selector runs once per stable case in a fresh process with a 120-second timeout. No condition uses repair, fallback, or error-triggered history expansion. BM25 is not included in this pilot.

StateLedger with recent3 reaches 28/29 versus 15/29 for plain static slicing and 4/29 for Recent3. The static comparison has 13 wins, 0 losses, 16 ties and session-cluster sign-flip $p = 0.0078$. Against qualified slicing, the comparison is 28/29 versus 25/29 (4 wins, 1 loss, 24 ties; interval [-0.0800, 0.2821], $p = 0.5$). Closure without recent3 reaches 27/29 at 10,828.1 characters, close to qualified slicing's 10,827.6; that accuracy difference also has an interval containing zero. The anonymous package includes the final executor, frozen method contract, input hashes, and aggregate summary.

D CoCoNote Boundary and Survivorship

The 1,050 candidates contain code targets with at least five prior code cells. Full-prefix execution fails for 595: 70 missing external resources, 217 missing Python dependencies, 265 notebook runtime errors, 29 syntax or indentation errors, and 14 timeouts. Among the 595 excluded cases, 591

repeat runs agree in status and detail and four differ in status. Retained and excluded targets have similar target length, but retained histories are much shorter: 1,781 versus 4,261 prior source characters and 12.84 versus 26.30 prior code cells on average. These statistics are descriptive boundary accounting and are not used to infer method performance on excluded cases.

Table 6: Method-independent CoCoNote full-prefix stability coverage across all 1,050 candidates. Quartiles are frozen rank partitions with deterministic case-id tie breaking; rates are retained full-prefix-exact cases divided by candidates in that quartile.

Stratifier	Q1	Q2	Q3	Q4
Prior source characters	69.6%	50.0%	38.4%	15.3%
Target cell index	57.0%	57.3%	41.4%	17.6%

Thus full-prefix executability declines sharply for later and longer histories, while target code length and AST complexity have no comparably large retained–excluded shift in the frozen survivorship audit. This is an evaluation-coverage boundary, not a selector comparison: all 455 stable cases remain in every CoCoNote method denominator, and no reduced-history result is imputed for the 595 cases whose full prefix is undefined.

The prefix dynamic-producer collector completes dynamic selections on 424/455 targets; its frozen qualified-static fallback handles the remaining 31 (19 collector timeouts and 12 unsupported observations). Reads are derived from source ASTs, while bindings and detectable mutations are observed during prefix execution. The target is never executed by the collector.

Ablation run correspondence. The primary configuration remains frozen at recent3. Its 339/455 ablation result and 342/455 main result use identical selections; three canonical-state-to-timeout transitions account for the run difference.

E Repair-Assisted LongDS Results and Diagnostics

Table 7: Prefix-causal LongDS results on 110 stable targets. Characters and segments are measured after shared support.

Method	Exact	Segments	Chars
Full prefix	1.0000	9.99	22789.8
Qualified symbol-reaching slice + recent3 + support	**0.9909**	8.32	20420.8
BM25 + recent3, matched target + support	0.9818	7.90	19790.9
StateLedger + recent3 + support	0.9636	7.50	**18825.5**
Static slicing + recent3, matched target + support	0.9545	8.42	20553.6

The prefix-causal replay boundary contains 110 targets. StateLedger reaches 106/110, qualified slicing 109/110, and matched BM25 108/110. StateLedger-minus-qualified is -0.0273 (1 win, 4 losses, 105 ties), with task-cluster interval [-0.0784, 0.0174]; BM25-minus-StateLedger is +0.0182, interval [-0.0270, 0.0734]. StateLedger selects 1,595 fewer final source characters than qualified slicing on average.

Shared support and budget semantics. All support-enabled methods use the same runtime. The selected history and target execute; missing-name or missing-column errors can add a prior defining turn, with at most eight ordinary additions in total. If execution still fails, the runner may expand static prerequisites once. When that expansion adds turns, it reruns with the unused portion of the same ordinary-repair budget, not a fresh budget. Every added turn precedes the target. Reference answers and future turns are unavailable to this procedure.

A matched source-character target applies to initial selection before repair and static expansion. Whole turns are selected without requiring equality to the target cost; post-support characters in Table 7 count the actual expanded history. StateLedger triggers ordinary repair on 54/110 targets and static support on 4/110; qualified slicing triggers 10/110 and 2/110, and the matched plain-static control 13/110 and 4/110.

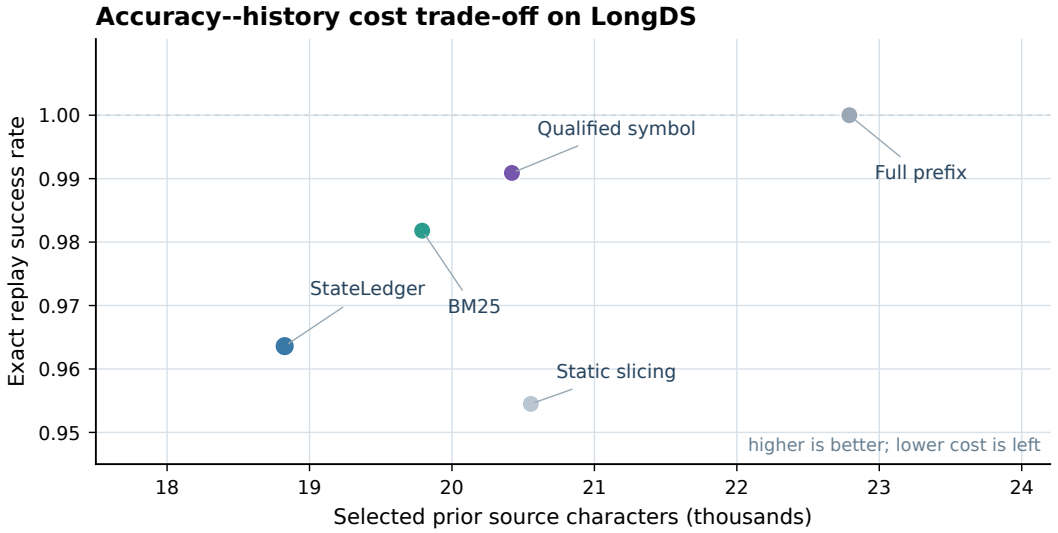


Figure 2: Accuracy–history cost trade-off on LongDS. Each point represents one method on the 110-target full-prefix-exact boundary after shared support. Qualified symbol slicing achieves the highest exact replay rate, whereas StateLedger selects fewer source characters, showing the observed accuracy–history trade-off under this shared runtime.

Table 8: LongDS repair decomposition on the fixed 110-target full-prefix-exact boundary. Ordinary repair is the deterministic name/column pass; static support is the one shared bounded pass after it. Characters are final source characters after the indicated stage.

Variant	Exact	Ordinary	Support	chars
StateLedger, no repair	50/110	0	0	16,980
+ ordinary name/column repair	102/110	54	0	18,731
+ shared static support	106/110	54	4	18,825
Matched static control, full protocol	105/110	13	4	20,554

Direct selector control. On all 110 targets, StateLedger is 106/110 and a direct same-model turn ranking is 101/110; the interval crosses zero. On the 102 targets with a complete direct ranking, direct ranking is 101/102 and StateLedger is 98/102. On the eight incomplete-ranking targets, StateLedger succeeds eight times and direct ranking contributes no successes. The net five-target intent-to-treat difference is therefore eight additional successes on incomplete-ranking cases offset by three fewer successes on complete-ranking cases.

Configuration and event perturbations. With shared support, recent-window sizes 0–5 span 0.9545–0.9636 exact; frozen recent3 is 106/110 and recent4/recent5 do not improve it. Dependency and transition perturbations change at most one target; masking `state_type` changes none. These diagnostics show limited sensitivity to the tested event perturbations and do not establish an independent contribution from every typed field.

Repair decomposition. Table 8 makes the runtime contribution explicit on the same 110-target boundary. StateLedger alone replays 50 targets; deterministic name/column repair reaches 102, and the one bounded shared-support pass reaches 106. The matched static control receives the same support protocol and reaches 105, so this table does not identify a repair-free selector advantage. It instead records the full analytical-state replay protocol and the much higher ordinary-repair trigger rate for StateLedger.

Exact metric sanity. All 990 recomputed LongDS trials agree with strict exact matching. Numeric tolerances through 10^{-2} add no successes.

Table 9: Frozen LongDS recent-window sweep on the 110-target full-prefix-exact boundary. Recent3 was fixed before execution and remains the paper configuration regardless of this descriptive sweep. Characters are final source characters after the indicated shared support pass.

	No static support		Shared static support	
Recent k	Exact	chars	Exact	chars
0	102/110	15,352	105/110	15,583
1	100/110	16,555	105/110	16,893
2	100/110	17,773	105/110	17,973
3	102/110	18,731	106/110	18,825
4	102/110	19,647	106/110	19,752
5	102/110	20,511	106/110	20,585

Runtime. Mean per-target execution time is 0.556 s for full prefix, 0.604 s for qualified slicing, 0.616 s for the matched static control, and 0.853 s for StateLedger. Recorded DeepSeek extraction latency averages 91.71 s over 45 timestamp-complete calls; provider token and dollar costs were not persisted.

F Teacher-Forced Target-Turn Diagnostic Protocols

The one-shot protocol generates one target program. The live protocol executes the selected recorded history once and permits at most three bounded `inspect` or `finish` actions. Target code is hidden in both. Public context and the target question seed history recovery; the target-source seed in the replay definition is unavailable here. Two independent method-blind judges score overall quality, numeric correctness, and coverage on a four-point scale. A semantic pass requires all three scores to be at least 3, and consensus requires both judges to pass. Nonexecution is a failure in the intent-to-treat denominator.

Live StateLedger reaches 44/110 consensus successes versus matched BM25's 28/110, with paired gain +0.1455 and task-cluster interval [0.0714, 0.2389]. Execution coverage is 81/110 versus 79/110; among executed trials, consensus is 44/81 versus 28/79. Full prefix reaches 54/110. Strict exact is 2/110 for StateLedger versus 3/110 for BM25. One-shot consensus is 22/110 versus 8/110.

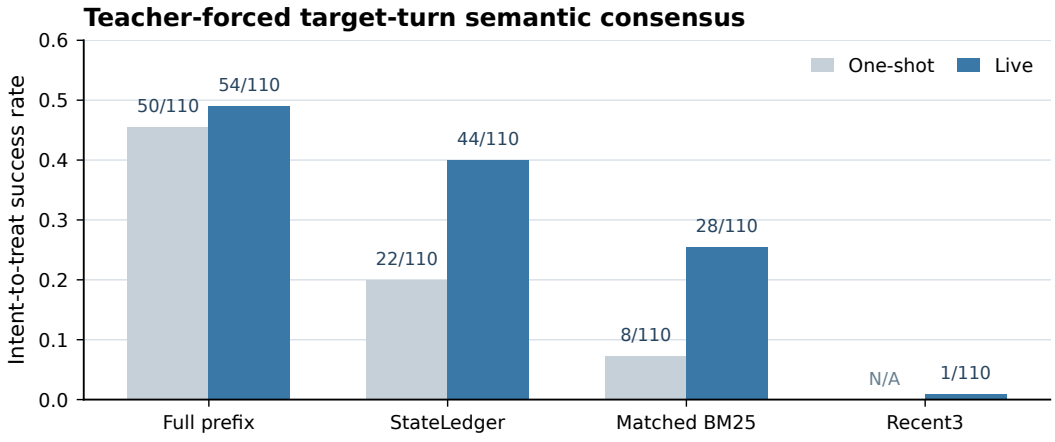


Figure 3: Teacher-forced target-turn semantic consensus. The figure compares one-shot and bounded live protocols on 110 LongDS targets. Semantic success is measured by conservative consensus between two independent judges, with nonexecuted trajectories counted as failures. In the live protocol, StateLedger achieves 44/110 consensus successes compared with 28/110 for matched BM25; one-shot Recent3 is not available under the fixed protocol.

All generation sources are frozen by SHA-256 before semantic evaluation. The one-shot primary manifest contains 220 method trials (110 StateLedger and 110 matched BM25), of which 54 execute and produce 50 unique method-blind candidate prompts. The full-prefix calibration expands it

Table 10: Teacher-forced target-turn diagnostic on 110 LongDS targets. Live permits up to three execution-feedback actions in persistent state. Semantic columns are intent-to-treat rates; nonexecution is a failure. The fully online protocol is evaluated separately.

Protocol	History	Exec.	Exact	DeepSeek	GLM	Consensus
One-shot	Full prefix	0.7818	0.0455	0.5000	0.5273	0.4545
	StateLedger	0.3818	0.0091	0.2273	0.2364	**0.2000**
	Matched BM25	0.1091	0.0091	0.0727	0.0818	0.0727
Live	Full prefix	0.7818	0.0182	0.5364	0.5364	0.4909
	StateLedger	0.7364	0.0182	0.4273	0.4818	**0.4000**
	Matched BM25	0.7182	0.0273	0.2727	0.3091	0.2545
	Recent3	0.6273	0.0000	0.0091	0.0091	0.0091

to 330 trials and 128 unique prompts. In the live transfer, all 440 answer-blind trajectories are first frozen; only then does a separate scorer load references. The four-method semantic manifest contains 440 rows and 295 unique prompts. Each prompt contains the public target context, question, reference answer, and one candidate output, but no history method, selected turns, source budget, strict-exact label, generation status, or competing candidate. Candidate tokens and prompt order are deterministic; call order is randomized once with seed 20260715.

The live worker runs in a chroot with an empty per-task workspace, no task JSON or source, no secrets, and blocked filesystem escape, network, subprocess, native-library, and stdin access. Frozen selected history executes once; up to three actions then share one Python state. Four StateLedger extractor failures remain intent-to-treat failures. The anonymous package includes the public protocol and aggregate analysis; raw trajectories, private runtime evidence, and reference-bearing score inputs are not redistributed.

Both judges use temperature zero and at most two attempts. Responses must contain exactly five fields: overall score, Boolean pass, numeric correctness, coverage, and a short reason. A single whole-response JSON code fence is accepted as transport formatting; surrounding prose or extra fields fail closed. For the live manifest, DeepSeek completes all 295 prompts; GLM completes 287 and fails closed on eight after the frozen retry. These failures touch two StateLedger, one BM25, two recent3, and three full-prefix rows. Only one affected StateLedger candidate and no affected BM25 candidate pass DeepSeek, so an optimistic agreement sensitivity gives 45/110 versus 28/110 rather than weakening the primary result.

For the live transfer, the individual-judge StateLedger-minus-BM25 task-cluster intervals are [0.0714, 0.2523] for DeepSeek and [0.0870, 0.2727] for GLM. Consensus gains are net positive in 11 of 23 tasks, negative in none, and tied in 12. These task-level results contrast with the replay comparison against qualified slicing, whose difference crosses zero, and with strict live exact, which is 2/110 for StateLedger and 3/110 for BM25.

G Blinded Human Semantic-Audit Protocol

The audit freezes 32 paired targets from the 110-pair historical teacher-forced frame before human labeling. Sampling is stratified by the prior method-blind semantic outcomes: two BM25-only pairs, 10 StateLedger-only pairs, 10 both-pass pairs, and 10 both-fail pairs. This design deliberately covers agreements and disagreements; it is not a prevalence-preserving sample and therefore does not estimate population pass rates. Each target contributes one StateLedger and one matched-BM25 answer, yielding 64 opaque items in randomized order. An item exposes the public target context and question, the reference answer, and one candidate answer, but no method identity, selected history, source budget, competing candidate, or internal audit key.

The coordinator recruited two workplace colleagues as independent annotators and did not annotate. Each annotator received only the recipient-specific 64-item packet, worked without discussing labels, and confirmed no access to the hidden method mapping or key and no use of an LLM to generate judgments. Reported completion time was approximately 20–30 minutes per annotator.

Table 11: Method-blind human semantic audit on a frozen stratified sample. Each method has 32 candidate answers; one answer per method is unresolved after two independent annotations. Rates use resolved answers only and are descriptive, not 110-target intent-to-treat estimates.

History	Pass	Resolved	Pass rate
StateLedger	22	31	**0.7097**
Matched BM25	11	31	0.3548

Table 12: Fully online LongDS semantic-consensus intent-to-treat rates on 40 frozen tasks (200 late-turn targets per method).

Method	Consensus	Execution	Turns	Chars
Full self-history	**0.2250**	**0.7600**	8.54	37559.4
StateLedger	0.1650	0.4950	3.40	**13086.3**
Direct selector + recent3	0.1650	0.6400	5.36	19638.3
BM25 + recent3	0.1600	0.5600	5.33	19393.2
Recent3	0.1500	0.4500	3.00	12670.9

Returned files were bound to the frozen packet tree and recipient identity by SHA-256 before scoring; annotator identities are retained privately and are not included in the anonymous artifacts.

Both annotators applied the same fail-closed rubric. A pass requires numeric correctness and coverage of at least 3/4. Wrong required entities, rankings, group membership, or material numbers cause a fail even when most requested fields are correct; for example, nine correct entries in a requested top ten do not pass when the tenth entry is wrong. Harmless key names, key order, equivalent numeric formatting, and list order when order is not requested are accepted. The frozen scorer resolves an item only when the two verdicts agree; both disagreements are retained as unresolved rather than adjudicated or replaced after outcomes are visible.

The resulting agreement is 62/64 (Wilson 95% interval [0.8930, 0.9914]) with Cohen's $\kappa = 0.9373$. Consensus resolves 31 answers per method: 22 StateLedger and 11 BM25 answers pass. At the pair level, StateLedger wins 12, BM25 wins one, both pass on 10, both fail on eight, and one pair remains unresolved. These are descriptive audit counts for the frozen stratified sample only.

H Fully Online LongDS Protocol

The fully online protocol fixes 40 tasks and five methods. Each task–method pair generates a 12-turn trajectory, yielding 200 trajectories and 2,400 generated turn records. Evaluation uses five late turns per trajectory (8–12), yielding 1,000 late-turn targets in total, or 200 per method. Each method generated its own prior trajectory under the same public-tool and bounded-action contract; no recorded selected history was injected. Before reference-bearing score inputs were loaded, the pre-reference trajectory population was frozen and independently checked against the frozen public task array. A public target-window validation separately verifies that every manifest-bound public task contains exactly ordered turn identifiers 1 through 12, preventing a positional/by-ID mismatch in the scorer.

StateLedger reaches 33/200 semantic-consensus successes versus 32/200 for BM25 + recent3 (task-cluster interval [-0.0650, 0.0700], $p = 1.0$). The scorer uses two method-blind judges and conservative consensus. Its aggregate analyses report every target, per-task macro estimates, task-cluster bootstrap intervals, leave-one-task-out ranges, and failure taxonomy. The semantic outcome is a dual-LLM consensus, not human ground truth.

A separate read-only cache audit replays the online StateLedger extractor updates in chronological order and validates their cache-bound prior state. It validates all 2,400 generated turn rows, 7,284 cache files, 2,400 history observations, and 2,400 turn records. Among 480 StateLedger turns, 272 extractor updates accept 798 events and 208 selector fallbacks leave the ledger unchanged; no accepted event has a non-current turn id and no earlier ledger prefix is mutated. This provides

implementation evidence for causal, write-once online ledger maintenance, not an online-success claim.

I CLUSTERED PAIRED INFERENCE

Estimands and resampling units. For paired target i and method m, write $X_{im} \in \{0,1\}$ for the protocol's success indicator: exact replay for replay experiments, or semantic consensus for the separately labeled generation diagnostics. For methods A, B, let $\Delta_i = X_{iA} - X_{iB}$. Partition paired targets into G nonempty clusters I_g (notebooks, tasks, or JuNE sessions), with $n_g = |I_g|$ and $d_g = \sum_{i \in I_g} \Delta_i$. The target-micro and cluster-macro differences are

$$\widehat{\Delta}_{\text{micro}} = \frac{\sum_{g=1}^{G} d_g}{\sum_{g=1}^{G} n_g}, \qquad \widehat{\Delta}_{\text{macro}} = \frac{1}{G} \sum_{g=1}^{G} \frac{d_g}{n_g}. \tag{15}$$

The former weights targets equally; the latter weights clusters equally. Their common pairing preserves the two methods' outcomes on each target. Clustering retains within-notebook or within-task dependence rather than treating targets as independent Bernoulli observations.

For bootstrap replicate b, draw $g_1^{*(b)}, \ldots, g_G^{*(b)}$ independently and uniformly from $\{1, \ldots, G\}$, retaining all targets and their pairing within each sampled cluster. Recompute

$$\widehat{\Delta}^{*(b)}_{\text{micro}} = \frac{\sum_{j=1}^{G} d_{g_j^{*(b)}}}{\sum_{j=1}^{G} n_{g_j^{*(b)}}}, \qquad \widehat{\Delta}^{*(b)}_{\text{macro}} = \frac{1}{G} \sum_{j=1}^{G} \frac{d_{g_j^{*(b)}}}{n_{g_j^{*(b)}}}. \tag{16}$$

The micro denominator changes with the sampled cluster multiplicities. For either estimand, let $\theta^*_{(0)} \le \cdots \le \theta^*_{(B-1)}$ be the sorted B bootstrap draws. The generic task-cluster implementation and primary CoCoNote analysis use the empirical percentile interval

$$\text{CI}_{0.95} = \left[\theta^*_{(\lfloor 0.025(B-1) \rfloor)},\ \theta^*_{(\lfloor 0.975(B-1) \rfloor)} \right]. \tag{17}$$

These are noninterpolated endpoints. The frozen JuNE implementation uses zero-based indices $\lfloor 0.025B \rfloor$ and $\lfloor 0.975B \rfloor - 1$ instead. Seeds and replicate counts are retained with each analysis artifact. These percentile intervals describe cluster-resampling uncertainty; the formulas do not assume independent targets or assert exact finite-sample coverage.

Exact cluster sign-flip test. The test statistic uses cluster net counts d_g, the numerator of the micro effect, rather than the macro-weighted differences. Discard zero net counts and relabel the remaining K clusters. Under the sign-symmetry null, conditional on their magnitudes, the joint signs are uniform over $\{-1, +1\}^K$. This is the assumption supporting the sign-flip reference distribution, not merely a zero-mean null. With independent Rademacher signs ε_g, the exact two-sided probability is

$$p = 2^{-K} \sum_{\varepsilon \in \{-1,+1\}^K} \mathbf{1}\left\{ \left| \sum_{g=1}^{K} \varepsilon_g |d_g| \right| \ge \left| \sum_{g=1}^{K} d_g \right| \right\}. \tag{18}$$

To obtain this distribution without enumerating all sign vectors, initialize counts $c_0(0) = 1$ and $c_0(u) = 0$ for $u \neq 0$, and recurse

$$c_j(u) = c_{j-1}(u - |d_j|) + c_{j-1}(u + |d_j|), \qquad p = \frac{\sum_{|u| \ge |\sum_g d_g|} c_K(u)}{2^K}. \tag{19}$$

Each old sign assignment has exactly two extensions, adding or subtracting the next magnitude; induction gives $c_j(u)$ as the number of assignments yielding total u, with total mass 2^j. This proves equivalence to Eq. (18) and matches `exact_sign_flip_pvalue`. If all counts vanish, $p = 1$. JuNE enumerates session signs directly (at most 20 sessions); including zero-net sessions multiplies both the numerator and denominator by the same power of two and leaves p unchanged.