STATETRACKBENCH: EXECUTABLE TRAJECTORIES FOR LOCATING WHERE LANGUAGE MODELS LOSE TRACK OF EVOLVING STATE

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-running agents must maintain the current state of a world that changes as messages arrive. A model may observe every update yet lose its cumulative effect, leaving its state incorrect even when the final answer is right. Final-answer scores alone cannot locate failures in message interpretation, state representation, or transition computation. We introduce **StateTrackBench**, a benchmark of $6{,}055$ executable trajectories that makes these failures observable. Each trajectory pairs messages with typed updates, deterministic transitions, and a query readout. Replay reconstructs the gold state after every message. Executing the same model-parsed updates through different state interfaces then separates computing and serializing state transitions from interpreting messages. The benchmark combines template-rendered platform records with generated task families. A four-model evaluation shows that three models approach ceiling on the platform strata, whereas generated histories remain challenging. GPT-4o diagnostics reveal corrupted state behind correct answers and transition errors on an auxiliary ledger evaluation. Using Parse–Then–Replay (PTR) as a parse-then-execute reference, we apply replay-time checks to locate messages for conflict-stated re-parsing. On GitHub label histories held out from repair development, the configured GPT-4o route improves with repair from 0.918 to 0.990 accuracy; matched controls support the contribution of conflict-specific feedback. Intermediate-state checks therefore reveal errors hidden by answer scores and provide a basis for targeted repair. Code and data will be made public upon acceptance.

1 INTRODUCTION

Long-running agents act in environments that change over time. Issues are reopened, labels are removed, and balances are revised. Reliable decisions require an accurate view of what is true after these updates, even when no single message states the current result. Reading every relevant event is therefore insufficient if the model loses their cumulative effect.

The GitLab history in Figure 1 illustrates the problem. GPT-4o's legacy model-state interface stores the first reopen as `true`, misses the second, and returns an incorrect count. A more fully specified interface and deterministic replay both recover the answer, but the final score does not explain why. An incorrect answer can arise from interpretation, representation, or transition computation, while a correct answer can conceal errors in state fields that the query never reads. Many memory and agent evaluations score answers or final states (Wu et al., 2025; Maharana et al., 2024; Hu et al., 2026; Hsieh et al., 2024; Yao et al., 2025). Checkpoint probes, per-step gold, and operation-level attribution provide finer views (Long et al., 2026; Kim & Schuster, 2023; Eric et al., 2020; Deng et al., 2026), and concurrent benchmarks compare designs on common executable streams (Fan et al., 2026; Cerruti et al., 2026; Pakhomov & Nijkamp, 2026).

We introduce StateTrackBench, which turns each history into an executable trajectory with typed intended updates, deterministic transitions, and a query readout. Replay reconstructs the gold state after every message, making the first incorrect state observable. Executing the same model-parsed updates with either a model or a program separates transition computation and serialization from interpretation (Table 1). The benchmark is independent of any particular solver. The $6{,}055$ trajectories

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.

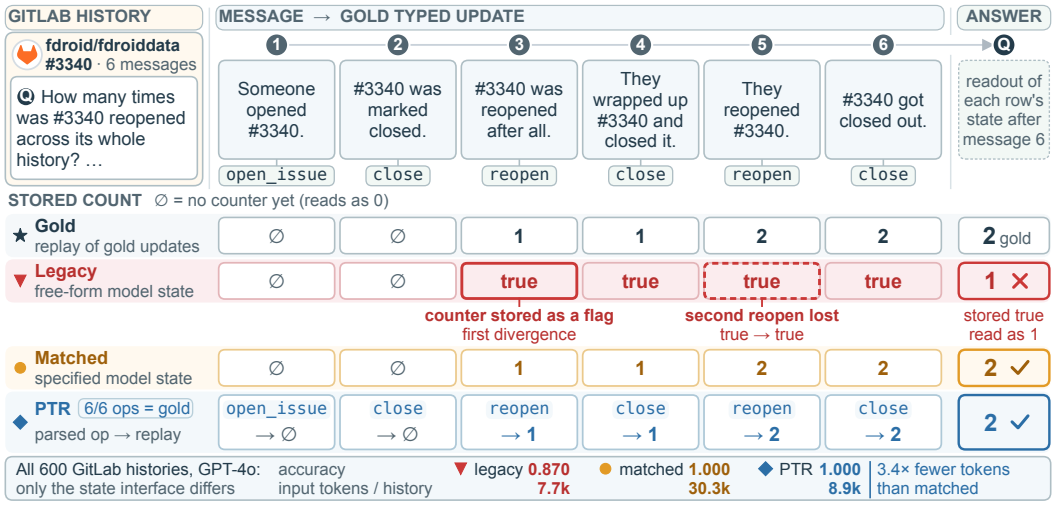

Figure 1: **Intermediate states reveal how a reopen count is lost.** In this GitLab example, GPT-4o's legacy model-state interface stores the first reopen as a Boolean and misses the second. Matched model state and PTR follow the gold count, with PTR executing GPT-4o's parsed updates in a program; ∅ reads as 0. The bottom panel reports accuracy and mean input tokens per history on all 600 GitLab reopen histories. The trajectories locate the first incorrect state and show how it affects the answer.

Table 1: **Executable trajectories combine per-message gold with fixed-parse comparisons.** Entries are coded from public documentation according to the definitions below and characterize design properties rather than benchmark quality.

Benchmark	Streams	Scale (unit)	Exec. gold	Per-step gold	Typed updates	Shortcut controls	Same stream	Fixed parse	Disjoint splits	Human-val. text
bAbI (Weston et al., 2016)	synth.	20 tasks×1k Q	✓	○	○	✗	✗	✗	○	✗
Kim & Schuster (2023)	synth.	2.2k×91 ex.	✓	✓	○	✓	✗	✗	✓	✗
RULER-VT (Hsieh et al., 2024)	synth.	500 ex./length	✓	✗	○	✗	✗	✗	✗	✗
LongMemEval (Wu et al., 2025)	human, LLM	500 Q	✗	✗	✗	○	○	✗	✗	✓
LoCoMo (Maharana et al., 2024)	human, LLM	7.5k Q	✗	✗	✗	○	○	✗	✗	✓
MemoryAgentBench (Hu et al., 2026)	mixed	2,071 Q	○	✗	✗	✗	○	✗	✗	○
MemTrace (Long et al., 2026)	LLM	15.4k Q	✗	○	✗	○	○	✗	✗	○
τ-bench (Yao et al., 2025)	human, LLM	165 tasks	✓	✗	○	✗	✗	✗	○	○
MultiWOZ 2.1 (Eric et al., 2020)	human	10k dialogues	✗	✓	✗	✗	✗	✗	✓	✓
StateMemBench (Fan et al., 2026)	LLM	322 probes	✓	○	✓	○	✓	✗	✗	✗
EAL-Bench (Cerruti et al., 2026)	LLM	3×32–64 pairs	✓	○	○	○	✓	✗	✗	✓
Parsing-the-Stream (Pakhomov & Nijkamp, 2026)	synth., agent	70 Q; 30 runs	✓	○	○	✗	✓	○	○	○
StateTrackBench (ours)	real, synth.	6,055 ex.	✓	✓‡	✓	✓	✓	✓	✓	✗

Exec. gold: computed by a program from typed updates or actions (○: some tasks). Per-step gold: the state after every update, turn, or message (○: probe points only). Typed updates: ≥ 3 families such as set, accumulate, conditional (○: one or two, or the agent's own actions). Shortcut controls: a no-tracking baseline or controls needing no accumulation (○: adversarial probes only). Same stream: designs compared on identical inputs (○: whole memory systems). Fixed parse: one model parse executed by a model and by a program (○: code-extracted facts with and without program aggregates). Disjoint splits: no source object, dialogue, or scenario in two splits (○: partial control; Parsing-the-Stream holds out development seeds, not templates). Human-val. text: the benchmark's rendered language is written, edited, or reviewed by people (○: part of it); human labels of observed failures do not count. Streams: synth. = templates; human = written or edited by people; agent = recorded agent sessions; real = platform records; mixed = recast plus new datasets. Scale: the scored unit each paper reports (Q: questions; ex.: examples).
‡Regenerated by replay.

combine histories derived from GitHub, GitLab, and npm records with 15 generated task families. All text is template-rendered. The collection covers replacement, accumulation, set updates, conditional transitions, relations, and time-sensitive state. Source-object splits keep related examples together, while shortcut readers test whether the answer requires accumulating updates.

We evaluate routes configured as GPT-5, GPT-4o, Gemini-2.5-Pro, and Doubao-Seed-1.8 on the test split and use controlled interfaces to examine the observed differences. The stronger models approach ceiling on platform histories but retain errors on generated state structures. Step-level GPT-4o diagnostics further reveal incorrect state fields behind correct answers. This evidence motivates a repair procedure built on Parse–Then–Replay (PTR), our parse-then-execute reference. PTR assigns

local update interpretation to the model and state accumulation to a deterministic program. Replay-time checks flag messages for re-parsing with explicit conflict feedback. On held-out GitHub label histories, this improves PTR with GPT-4o at an average cost of 0.89 additional calls per history relative to unrepaired PTR. Controls with matched messages and call budgets support the role of conflict-specific feedback (Section 5.4).

Our contributions are **(i)** a formulation of state-maintenance reliability that exposes failures left unresolved by final-answer scores; **(ii)** an executable benchmark spanning platform-derived and generated histories, with parse-then-replay and repair interfaces for studying those failures; and **(iii)** a four-model evaluation across state structures and data sources, together with a held-out demonstration of repair effectiveness on a GitHub label task with GPT-4o.

2 Related Work

Memory and long-context evaluation. LongMemEval, LoCoMo, and MemoryAgentBench assess retrieval, temporal updates, and selective forgetting through question answering over long or incremental histories (Wu et al., 2025; Maharana et al., 2024; Hu et al., 2026). RULER probes synthetic variable tracking, while τ-bench evaluates end-to-end agents on stateful tasks (Hsieh et al., 2024; Yao et al., 2025). These outcome scores expose long-history difficulty without locating whether an error begins in interpretation, transition, or storage.

State-tracking diagnostics. Synthetic probes vary update count or state complexity (Kim & Schuster, 2023; Hou et al., 2026), dialogue benchmarks annotate per-turn state (Eric et al., 2020), and formal analysis studies architectural expressivity (Merrill et al., 2024). StateMemBench and EAL-Bench pair executable gold with same-stream comparisons at probes or requests, without source-disjoint splits; EAL-Bench includes human review of its generated text (Fan et al., 2026; Cerruti et al., 2026). Parsing-the-Stream tests deterministic aggregation by removing aggregates from a view of the same code-extracted facts (Pakhomov & Nijkamp, 2026). MemTrace probes typed facts at checkpoints (Long et al., 2026), while Deng et al. (2026) localize faulty operations in executable memory-evolution graphs. StateTrackBench adds intended updates for every message and fixed-parse comparisons, enabling per-step first-divergence localization and executor isolation.

Parse-then-execute and repair. PTR follows methods that execute model-written programs, formalizations, or typed operations (Gao et al., 2023; Pan et al., 2023; Andreas et al., 2020; Hu et al., 2022; Wang et al., 2026). Other designs manage memory stores (Chhikara et al., 2025; Packer et al., 2023) or rewrite the running state (Yoneda et al., 2024). Logic-LM revises formalizations using solver errors (Pan et al., 2023); commit-time validators check proposed updates (Bryan & Nottingham, 2013; Zhang et al., 2026; Cui et al., 2026). Together, these methods establish the parse–execute split and validation. PTR uses this boundary as a controlled measurement interface, while replay-grounded repair returns execution conflicts to local interpretation, including suspected omissions for which no proposed update exists to validate.

3 Benchmark

3.1 Executable Trajectories

A task specification $\tau = (\mathcal{S}, \mathcal{E}, s_0, F, q, G_q)$ fixes a state space, a typed update schema that includes `no_update`, an initial state, a total transition function, and a readout for query q. Given messages $x_{1:T}$, the gold process and a solver A are

$$e_t^* = \mathcal{I}_\tau(x_t), \qquad s_t^* = F(s_{t-1}^*, e_t^*), \qquad y^* = G_q(s_T^*), \qquad \hat{y} = A(s_0, x_{1:T}, q; \tau). \tag{1}$$

In construction the direction is reversed: updates are sampled or recorded first and each message is rendered from its update, so the intended update is known for every message. An example stores $x_{1:T}$, $e_{1:T}^*$, q, s_T^*, and y^*, and replaying $e_{1:T}^*$ through F regenerates every intermediate state s_t^*. A solver that exposes $\hat{e}_t$ and $\hat{s}_t$ can therefore be checked step by step for interpretation errors ($\hat{e}_t \neq e_t^*$), representation errors ($\hat{s}_t \notin \mathcal{S}$), transition errors ($\hat{s}_t \neq F(\hat{s}_{t-1}, \hat{e}_t)$), and readout errors ($\hat{y} \neq G_q(\hat{s}_T)$). Interpretation error here denotes disagreement with the intended update; text–event fidelity is assessed separately. A query is *lookup* when every update e that can change its value

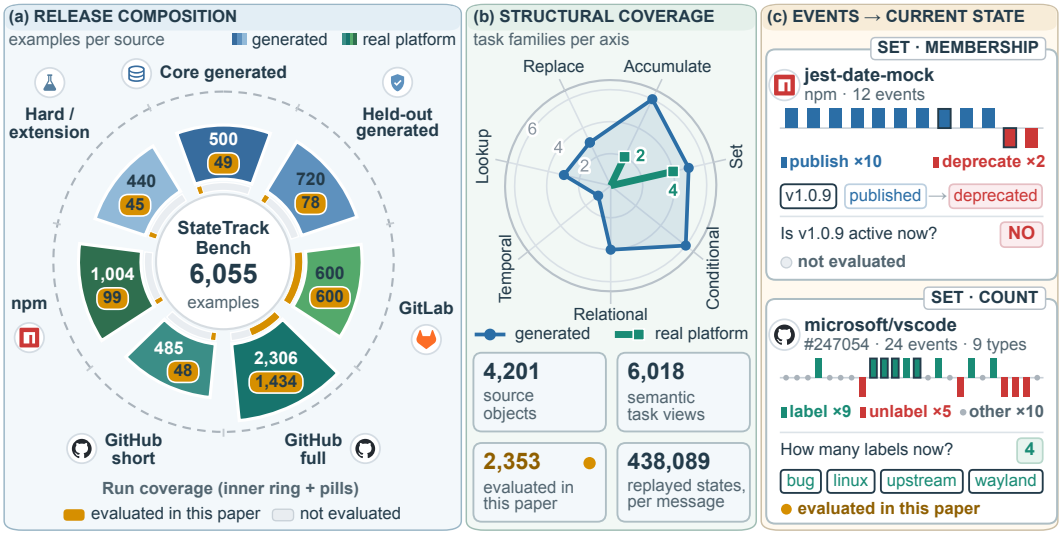

Figure 2: **Platform records and generated structure share one replay interpreter.** (a) The 6,055 examples in seven cohorts (with compressed petal lengths); markers identify the 2,353 examples evaluated across the test split and diagnostic cohorts. (b) Generated and platform task families per structural axis; counts of source objects, semantic task views (a snapshot with one query), scored examples, and replayed states, one per message. (c) Two platform histories, with the events that determine the answer outlined. Platform records ground the tasks in recorded events, while generated families extend their structural coverage.

fully determines that value, $G_q(F(s,e)) = G_q(F(s',e))$ for all $s, s' \in \mathcal{S}$, and *history-dependent* otherwise.

3.2 Construction

Sources and admission. Of the 6,055 examples, 1,660 come from 15 generated task families that sample legal update streams from specifications, and 4,395 from GitHub issues and pull requests, GitLab issues, and npm release histories, with each event mapped to the shared typed-update schema (Figure 2). Generated histories have a median of 121 executable operations and platform histories 14; only generated histories contain messages that change no state (`no_update`), drawn from 10 author-written sentences. Selection follows deterministic rules independent of model outputs, while assignee queries preferentially select users for whom the first-update or most-recent-update reader fails. For GitHub and GitLab, replaying $e^*_{1:T}$ from s_0 must reproduce the platform-reported state, so gold states are derived by replay rather than manual annotation; npm and generated histories use source-specific consistency checks, and histories that fail validation are excluded.

Surfaces and splits. Deterministic templates render each history as clean and noisy paraphrased surfaces with identical message counts, updates, queries, and gold answers. GitHub and GitLab preserve recorded event order; npm histories order releases by publication time and place deprecations after releases in a constructed order consistent with registry state. The 6,055 examples form 6,018 semantic task views (a snapshot with one query) from 4,201 source objects; this study evaluates 2,353 examples. A deterministic source-object assignment produces 4,750 train, 655 development, and 650 test examples.

3.3 Quality and Validity Controls

Replay and leakage. Replay regenerates 438,089 states, one after every message, and the final state and answer match the stored values on 6,055/6,055 examples. No source object occurs in two splits. Because platform objects are public, GPT-5 and GPT-4o are also scored closed book, on the query alone, to probe recall.

Table 2: **State interfaces separate interpretation from transition execution.** Upper rows denote the evaluated interfaces; fixed-operation rows reuse PTR's parse to isolate model and program execution. Parenthesized methods indicate shared mechanisms rather than replications; lower rows summarize mechanisms described in prior work. M: model; P: program; † single call; capped: truncated previous state. "Commit" checks an output before storing it; "replay" checks every replayed step. ‡Null parses of messages with a number. §Tested where every message carries an update. *Ledger grid only.

interface (related methods)	sees $\hat{s}_{t-1}$	model emits	computes / stores	check	on failure	omissions
A0 closed book	×	answer	—	none	—	—
A1 full context	×	answer	M†	none	—	—
A2 legacy model state	capped	full state	M / M	parse only	reject	×
A3 matched model state (Statler)	✓	full state	M / M	commit	retry, reject	×
A4 JSON patch (IC-DST)	✓	delta	M / P	commit	retry, reject	×
A4f* fixed-operation patch fold	✓	delta for $\hat{e}_t$	M / P	commit	retry, reject	—
A5 fixed-operation fold	✓	state for $\hat{e}_t$	M / M	commit	retry, reject	—
A6 PTR (Text2Mem, EAL)	×	typed update	P / P	commit	retry, drop	partly‡
A7 oracle updates	×	answer	M†	none	—	—
PTR + replay-grounded repair (ours)	at re-parse	typed update	P / P	commit, replay	re-interpret	key mention§
Logic-LM	×	formalization	solver / —	solver run	re-interpret	×
commit-time checks (PatchBoard, MemTxn)	partly	update	M / P	commit	reject	×
memory systems (Mem0, MemGPT)	retrieved	memory op	M / store	model-judged	overwrite	×

Shortcut readers and lookup controls. Three registered readers return the specification default, the first relevant update, or the most recent one, and $3{,}463$ of the $5{,}715$ history-dependent examples defeat all three (test-split reader scores: Table 3). The registry is finite, so the 369 reader-resistant test examples form a difficulty stratum, not a shortcut-free set. 340 lookup examples, solved by the most-recent reader by design, separate failures to read a message from failures to maintain state.

Surface fidelity. Replay does not establish that a message x_t faithfully expresses e_t^*. Two LLM judges from different vendors, blind to the typed events, flag 2 and 3 of 280 noisy-surface platform units as adverse, with no answer-leakage flags; on 220 auxiliary-grid units, they flag 14 and 23, respectively. No generated benchmark example was audited. No human audit was run; low judge agreement on a unique executable reading ($\kappa = 0.169$) limits this diagnostic.

Evaluation settings. Each example supports full-context answering from the history, bundled oracle-update answering from gold updates with their transition rules, and closed-book answering from the query alone. Per-message state interfaces additionally expose the interpretation and execution steps. Oracle-update prompts replace message-to-update inference, state F and any non-empty s_0, and drop no-update messages, but retain conflicting JSON-output instructions in the parser vocabulary. The full-versus-oracle gap is therefore a setting-level contrast rather than a pure decomposition of interpretation and accumulation.

4 METHOD

A state interface assigns message interpretation, transition computation, and state storage to the model or a program. StateTrackBench makes these choices comparable through per-message gold states and fixed-parse execution. Table 2 places the evaluated interfaces alongside related designs (Yoneda et al., 2024; Hu et al., 2022; Cerruti et al., 2026; Chhikara et al., 2025; Packer et al., 2023); these are mechanism comparisons, not replications of the named systems.

Fixed-input counterfactuals. Within each evaluation cohort, the compared interfaces receive the same examples and therefore share $x_{1:T}$, $e^*_{1:T}$, F, and G_q. The transition contract is compiled from the specification that produces the gold and states the effect of each update type, such as "$reopens[issue]$ increases by 1". Matched model state, JSON patch, both fixed-operation folds, and oracle updates receive this contract; all but oracle updates also receive demonstrations obtained by replaying PTR's few-shot updates. No interface receives manually written task descriptions beyond PTR's operation vocabulary and few-shot pairs. Both fixed-operation folds reuse PTR's final updates

verbatim: the full-state fold isolates who computes and serializes each transition, while the patch fold isolates who computes new values with program-owned storage. Disagreement in either pair therefore occurs after interpretation. Offline comparison with gold states locates the first divergence without exposing gold to an interface. Legacy model state rewrites a capped JSON state without schema, contract, or demonstrations.

Parse, then replay. Parse–Then–Replay (PTR) asks the model only for the local effect of each message,

$$\tilde{e}_t = P_\theta(x_t; \mathcal{E}), \qquad \hat{e}_{1:T} = R_{\mathcal{E}}(\tilde{e}_{1:T}, x_{1:T}), \qquad \hat{s}_t = F(\hat{s}_{t-1}, \hat{e}_t), \qquad \hat{y}^{\mathrm{PTR}} = G_q(\hat{s}_T), \quad (2)$$

where the parser P_θ sees the message, the update schema, and few-shot pairs but not the running state. The post-processor $R_{\mathcal{E}}$ normalizes updates with specification-derived corrections; its parser fallback re-parses flagged messages once and drops invalid updates. It uses no gold but reads sibling updates in the same stream, so it is stream-level post-processing rather than an online filter. PTR serves as a reference whose parse–execute boundary enables fixed-parse comparisons.

Replay-grounded, conflict-stated repair. Replay applies specified transitions but cannot correct a misread update; replay-grounded repair turns a replay-time alarm into local parser feedback. A precondition alarm, derived from the specification, fires when an effect of $\hat{e}_t$ does not apply to $\hat{s}_{t-1}$, such as removing an absent label. An omission alarm, a key-mention heuristic, fires when x_t mentions a tracked key, such as the queried issue, but its update is `no_update` after parsing or dropping. Either alarm triggers one re-parse of x_t with the detected conflict; the prompt also includes the affected state slice, whose contribution is evaluated in Section 5.4. For an omission alarm, a well-formed candidate replaces `no_update` only if it raises no precondition alarm. For a precondition alarm, a well-formed candidate replaces the original if it differs, after which replay continues from the revised operation. Alarms request reconsideration without automatically rejecting an update because valid platform events can violate simple preconditions. A generic re-read repeats the task; conflict-stated feedback names the violated condition or suspected omission, narrowing what to reconsider. Commit-time validation checks proposed updates (Bryan & Nottingham, 2013; Zhang et al., 2026; Cui et al., 2026), whereas an omitted update supplies no proposal to check. The repair uses no gold and re-parses each flagged message at most once; its held-out evaluation covers GitHub labels using the configured GPT-4o route.

5 Experiments

5.1 Experimental Setup

Leaderboard. We evaluate GPT-5, Gemini-2.5-Pro, Doubao-Seed-1.8, and GPT-4o on the 650 noisy test histories with full context and oracle updates. GPT-5 and GPT-4o are also evaluated closed book. Each setting uses one call per example, with provider-default reasoning and unequal token budgets. All rows complete after retries; budget-truncated answers count as errors. Gemini-2.5-Pro is a configured route label; GPT-4o repair runs did not retain returned model identities. Request details are in the appendix. The non-model baselines are the best registered reader on gold updates and a train-split answer prior. Intervals use a 95% source-object cluster bootstrap. The leaderboard is descriptive: 253 test examples share source objects with diagnostic cohorts (208 exact matches); no between-model significance tests are used.

Diagnostic cohorts. The interface study covers GitHub (468) and GitLab (600) reopen-count histories, a dense ledger grid, and GitHub label-count histories in development (443) and held-out (400) cohorts. The repair evaluation holds source objects out from repair development using a partition separate from the official benchmark split. GPT-4o is evaluated at temperature 0 unless otherwise specified. The reopen comparisons and held-out repair evaluation were prespecified; all other hypothesis tests are exploratory and report unadjusted p-values.

5.2 Main Results

Generated histories remain challenging when platform strata approach ceiling. GPT-5, Gemini-2.5-Pro, and Doubao-Seed-1.8 score highly on each platform stratum, with lower accuracy

Table 3: **Models near ceiling on platform records still err on generated histories.** Normalized answer-match accuracy on the official split (numeric tolerance 0.011; higher is better) with full context and oracle updates; parentheses give example counts. Resistant: all registered readers fail; long: each source's long band. Bold: highest observed accuracy per setting and column, ties included. Reader: best registered reader on gold updates; prior: most frequent train answer to the question template. Budget-truncated responses count as errors.

model	setting	all (650)	GitHub (315)	GitLab (64)	npm (99)	generated (172)	resistant (369)	long (196)
GPT-5	full context	**.965**	**.990**	**1.00**	**1.00**	**.884**	**.951**	**.954**
	oracle updates	**.988**	**1.00**	**1.00**	**1.00**	**.953**	**.978**	**.974**
Gemini-2.5-Pro	full context	.938	.978	**1.00**	.939	.843	.911	.913
	oracle updates	.966	**1.00**	**1.00**	.879	.942	.943	.908
Doubao-Seed-1.8	full context	.917	.946	**1.00**	**1.00**	.785	.867	.908
	oracle updates	.974	**1.00**	**1.00**	.990	.907	.957	.954
GPT-4o	full context	.577	.679	**1.00**	.394	.337	.333	.388
	oracle updates	.689	.857	**1.00**	.455	.401	.523	.500
best shortcut reader	gold updates	.418	.451	.875	.172	.349	.000	.311
answer prior (train)	none	.362	.381	.875	.172	.244	.201	.260

on generated histories (Table 3). GPT-4o reaches 0.577 overall, but its generated-history accuracy of 0.337 is below the best reader's 0.349. GitLab offers little separation because most histories have one reopen, and the answer prior already reaches 0.875.

Oracle updates leave different errors across models. Oracle updates improve every model's overall accuracy, yet also turn previously correct answers into errors for every model. Gemini-2.5-Pro's npm accuracy declines under this setting (Table 3). Most full-context errors of GPT-5, Gemini-2.5-Pro, and Doubao-Seed-1.8 disappear, whereas 185 of GPT-4o's 275 errors persist. The contrast is especially clear against GPT-5, whose persistent errors number 2 of 23. Oracle updates also supply the transition rules and initial state and remove no-update messages. These bundled changes prevent attributing the accuracy difference solely to interpretation versus accumulation.

Closed-book answers fall below the answer prior overall. In the query-only condition, GPT-4o scores 0.132 and GPT-5 0.251, below the train-split answer prior of 0.362. This provides no positive evidence of record recall across the split, but cannot establish that records were absent from pretraining. The gold states postdate both model snapshots, limiting the probe to recall of earlier-settled states. On npm, both models exceed the prior, mostly on yes/no questions, and recall remains possible.

PTR reaches perfect subset accuracy at greater cost. PTR with GPT-4o answers all 44 generated test histories, sampled across task families and length bands, while GPT-5 with full context reaches 0.841. PTR uses about 198 calls per history and 9.6× the tokens of GPT-5's single call, including reasoning. Inference budgets differ across systems, and some GPT-5 errors are truncated responses. PTR uses the benchmark's deterministic transition function and is evaluated with GPT-4o parses. The subset is too small to support a performance distinction between PTR and GPT-5 or Gemini-2.5-Pro with oracle updates.

5.3 State Tracking Diagnostics

5.3.1 State Representation under Legacy and Specified Interfaces

Legacy model state falls below a full-context read. On the GitHub and GitLab reopen-count histories, legacy model state reaches 0.885 and 0.870, below a single full-context read at 0.951 and 0.975 (constant answer 1: 0.833 and 0.905). In 95 of its 132 errors the final counter is a word such as "reopened" or the issue identifier, and the other 37 undercount, as when a stored `true` reads as 1 (Figure 1). Matched model state removes all 132 errors without introducing one. This improvement combines representation, prompting, and decoding changes, examined in the ablation study.

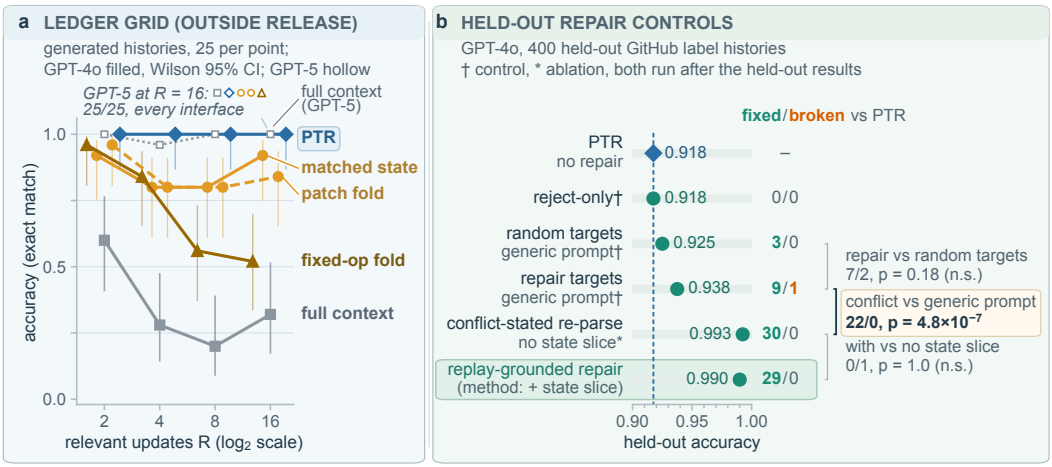

Figure 3: **Fixed parses expose GPT-4o execution errors; conflict feedback improves label repair.** (a) GPT-4o (filled) and GPT-5 (hollow; full context dotted); only full context is not told the sign convention of balances, and 32 of the 100 gold answers are 0. (b) Each rung changes one component; annotations report histories corrected or invalidated relative to PTR, and brackets report discordant predictions. The ledger grid is an auxiliary evaluation set; both panels report a single realization.

Specified interfaces tie at ceiling. Matched model state, JSON patch, fixed-operation fold, and PTR answer all 1,068 histories correctly. The prespecified comparison excludes three pilot histories and finds 0 disagreements across the remaining 1,065 histories (95% upper bound 0.35%). This ceiling result establishes no advantage for PTR on these cohorts.

5.3.2 Transition: Fixed-Parse Isolation of the Executor

With the parse fixed, GPT-4o miscomputes transitions that replay executes. The short platform histories leave both executors at ceiling, motivating evaluation on a denser auxiliary ledger grid. Its 100 histories vary the number R of transfers touching the queried person while keeping message count fixed. Irrelevant transfers and answer magnitude co-vary with R, and the fold comparisons were introduced after the initial grid analysis. When GPT-4o executes PTR's parsed updates, it answers 0.720 of the histories correctly; program replay of the same updates answers all of them (Figure 3a). Fold accuracy falls from 0.96 at $R = 2$ to 0.52 at $R = 16$. Because the parsed sequence is identical, the errors arise in computing or serializing state, although the trend with R is observational. In the fixed-operation patch fold, the program stores the state and all model-written patches are well formed, yet accuracy is 0.850. Thus valid update syntax alone does not ensure correct state transitions.

The transition failure depends on the model. In a subsequent evaluation, GPT-5 answers 99/100 grid histories with full context and all 25 histories in the $R = 16$ slice under every other interface. With 25 examples, the 95% Clopper–Pearson upper bound on its per-interface error rate is 14%. These results distinguish GPT-5 from GPT-4o's fixed-operation fold, but not from GPT-4o's matched model-state or patch-fold conditions.

5.3.3 Interpretation: Silent Corruption and Targeted Re-parsing

Final answers hide corrupted states. PTR's update-level accuracy on the reopen histories is 0.961 (GitHub) and 0.932 (GitLab): it answers all 1,068 correctly, yet 84 of 468 and 162 of 600 end in an incorrect state. All but 5 of the 802 divergent updates are omissions, mostly of open, assign, and label events the reopen readout ignores, so final-answer scoring cannot see them.

Held-out errors begin at omitted or inverted updates. On open-vocabulary label counting, omissions reach the answer: on the development cohort, PTR (0.962) trails matched model state

(0.998; discordance 1/17, PTR-only / model-state-only), and 16 of its 17 errors are label additions parsed as no update. Because replay is exact, every PTR answer error has a first update at which the parse departs from gold; for the 33 errors on the 400 held-out histories these are 29 omissions and 4 label additions that PTR's normalization turns into removals.

Conflict-stated re-parsing improves held-out label accuracy. The held-out histories share no source object with the development cohort. Repair raises PTR from 0.918 to 0.990, correcting 29 errors while introducing 0 new errors (Figure 3b). A prespecified variant with an additional retrospective pass produces identical predictions and satisfies the adjusted test criterion. The gain persists on repositories absent in development; over the full held-out cohort, the 95% repository-cluster interval is $[2.2, 14.2]$ percentage points. The repair corrects 25 of the 29 omission errors and all 4 normalization errors at 0.89 extra calls per history. The normalization errors are detected exclusively by the precondition alarm.

Alarm coverage does not imply selectivity. On development histories the omission alarm flags every answer error, but also many correct answers. Every held-out observation carries a gold update, so the omission alarm there detects null parses and does not test discrimination against genuine no-update messages. Offline replay exposes a different limit on generated histories: the precondition alarm has 0.75% precision and 4.72% recall, and the omission alarm never fires, including on the 2,063 gold no-update observations. Precondition checks also flag correct updates in every platform block. False alarms induce additional re-parsing and may alter updates; consequently, the absence of newly introduced errors on the held-out cohort does not establish general selectivity or safety.

5.4 Ablation Study

State-interface components. Matched model state adds JSON decoding, a numeric schema, validation with one retry, a transition contract with demonstrations, an uncapped state, and a larger output budget. Validation never fires on the 14,947 matched-state outputs, and legacy model state never reaches its cap or output limit. The gain therefore cannot be attributed to retries or avoiding truncation. On GitLab, JSON mode with a minimal numeric schema but without the contract or demonstrations also answers all 600 histories. Because the complementary legacy interface with the contract was not evaluated, the effects of schema and serialization remain confounded.

Stating the conflict contributes more than supplying the state slice. Matched controls separate rejection, targeting, conflict feedback, and state context (Figure 3b). The rejection-only control makes no model calls and yields accuracy 0.918. The re-parse controls each use the repair's 356 calls. A generic prompt reaches 0.925 at random observations and 0.938 at the alarmed ones; the targeting difference is not significant ($p = 0.180$). Stating the conflict at those same targets reaches 0.993, compared with 0.938 for the generic prompt. Adding the affected state slice yields 0.990 and does not improve accuracy. This comparison supports the conflict feedback on one task and one model; the held-out cohort has no matched model-state baseline.

6 Conclusion

We introduced StateTrackBench to study how language models maintain state under successive updates. Its executable trajectories connect each message to an intended update and a checkable intermediate state, exposing how an answer was reached. Platform records ground the evaluation in recorded changes, while generated task families extend the range of state structures and update patterns. Across GPT, Gemini, and SEED models, these settings reveal different levels of difficulty; GPT-4o diagnostics show that correct answers can conceal incorrect state and that fixed-parse comparisons can locate execution failures.

The diagnostic framework also supports targeted repair: when replay raises an alarm, the parser receives the specific conflict and reinterprets the message. On held-out label histories, this feedback improves accuracy with few additional calls. These results support evaluating maintained state throughout the trajectory, including information not read by the final query. Future work will extend this evaluation to human-validated language, additional stateful systems, and repair across more task structures.

AI USE STATEMENT

Large language models serve as evaluated systems in this study. Routes configured as GPT-5, Gemini-2.5-Pro, Doubao-Seed-1.8, and GPT-4o are queried through provider APIs under the evaluation settings defined in the paper; their outputs are recorded and scored by deterministic evaluation code. Two language models also support the surface-fidelity diagnostic, which is reported separately from human validation. LLM-based writing assistance was used only for language polishing and grammatical revision.

ETHICS STATEMENT

StateTrackBench contains no raw user-written text. Platform histories are rendered from public GitHub, GitLab, and npm event metadata. The GitHub login and assignee values that appear in them are replaced by salted pseudonyms whose salt is not released, and a privacy scan of all platform-derived records and raw source snapshots found no plaintext e-mail addresses, handles, or credential tokens; public repository, project, and package names remain for provenance. Code is licensed under Apache-2.0 and derived data under CC BY 4.0. The supplementary material provides selected dataset examples and recorded model outcomes. Code and data will be made public upon acceptance. The public release will include instructions for maintainers to request corrections or removal of their source records. The benchmark measures a reliability property of agents; we do not foresee direct harmful uses beyond those of general agent capability evaluation.

Dataset documentation. StateTrackBench is intended for three uses: evaluating how well systems maintain evolving state, diagnosing where a system loses it with the settings and step diagnostics of Sections 5.2 and 5.3, and developing state-keeping methods on the train and development splits, which share no source object with the 650-example test split. The diagnostic cohorts used in this study instead follow the separate selections documented in Appendix Table 11, including their overlap with the official test split. Out-of-scope uses include profiling maintainers, recovering identities, and drawing conclusions about agent memory beyond observable, rule-governed state changes. The paper evaluates version 1.0.0; the ledger grid of Section 5.3.2 is evaluated separately as an extension set. Because gold comes from replay rather than annotation, the benchmark can be renewed with histories built from newer platform events that pass the same replay admission, under new version numbers. The core appendix and Additional Supplementary Material document dataset composition, collection, preprocessing, privacy handling, intended uses, and known limitations.

REPRODUCIBILITY STATEMENT

The supplementary material contains additional protocol documentation, the complete prompt reference, selected dataset examples and recorded model outcomes, and an offline website for inspecting these cases. Code and the complete dataset will be made public upon acceptance. Appendix B documents the interfaces, call and token accounting, and statistical procedures; Appendix E summarizes protocol deviations. The Additional Supplementary Material provides the complete prompt reference and deviation record. The ledger grid is an extension set outside version 1.0.0 and will be released with the benchmark upon acceptance. Intermediate gold states are reconstructed by deterministic replay of each example's gold updates, one after every message. The final state and answer were verified for all $6{,}055$ examples ($438{,}089$ states, $358{,}181$ of them after an update other than `no_update`). The historical acquisition and evaluation figures in the Additional Supplementary Material come from the earlier study and are not regenerated. Repeated API calls can produce different outputs, and provider-default settings can change as providers update their models; the reported results describe the recorded realization of each run. The closed-book setting of Section 5.2 probes contamination by measuring how much of the test split a model answers from the query alone.

REFERENCES

Jacob Andreas, John Bufe, David Burkett, Charles Chen, Josh Clausman, Jean Crawford, Kate Crim, Jordan DeLoach, Leah Dorner, Jason Eisner, Hao Fang, Alan Guo, David Hall, Kristin Hayes,

Kellie Hill, Diana Ho, Wendy Iwaszuk, Smriti Jha, Dan Klein, Jayant Krishnamurthy, Theo Lanman, Percy Liang, Christopher H. Lin, Ilya Lintsbakh, Andy McGovern, Aleksandr Nisnevich, Adam Pauls, Dmitrij Petters, Brent Read, Dan Roth, Subhro Roy, Jesse Rusak, Beth Short, Div Slomin, Ben Snyder, Stephon Striplin, Yu Su, Zachary Tellman, Sam Thomson, Andrei Vorobev, Izabela Witoszko, Jason Wolfe, Abby Wray, Yuchen Zhang, and Alexander Zotov. Task-oriented dialogue as dataflow synthesis. *Transactions of the Association for Computational Linguistics*, 8: 556–571, 2020. doi: 10.1162/tacl_a_00333.

Paul C. Bryan and Mark Nottingham. JavaScript Object Notation (JSON) Patch. RFC 6902, Internet Engineering Task Force, April 2013.

Tommaso Cerruti, Mika Okamoto, and Ansel Kaplan Erol. Agent memory is a surface for endogenous authorization laundering. *arXiv preprint arXiv:2609.01836*, 2026.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, Frontiers in Artificial Intelligence and Applications, 2025. doi: 10.3233/FAIA251160.

Hanshuai Cui, Zhiqing Tang, Zhi Yao, Fanshuai Meng, Qianli Ma, and Weijia Jia. MemTxn: A transaction boundary for source-supported updates and complete-state recovery in agent memory. *arXiv preprint arXiv:2607.27834*, 2026.

Xinle Deng, Ruobin Zhong, Hujin Peng, Xiaoben Lu, Yanzhe Wu, Guang Li, Buqiang Xu, Yunzhi Yao, Jizhan Fang, Haoliang Cao, Junjie Guo, Yuan Yuan, Ziqing Ma, Yuanqiang Yu, Rui Hu, Baohua Dong, Hangcheng Zhu, and Ningyu Zhang. MemTrace: Tracing and attributing errors in large language model memory systems. *arXiv preprint arXiv:2605.28732*, 2026.

Mihail Eric, Rahul Goel, Shachi Paul, Abhishek Sethi, Sanchit Agarwal, Shuyang Gao, Adarsh Kumar, Anuj Goyal, Peter Ku, and Dilek Hakkani-Tur. MultiWOZ 2.1: A consolidated multi-domain dialogue dataset with state corrections and state tracking baselines. In *Proceedings of the Language Resources and Evaluation Conference (LREC)*, pp. 422–428, 2020.

Xinyi Fan, Miri Liu, Ruozhen Yang, Siru Ouyang, and Jiawei Han. Can agent memory systems track evolving state? *arXiv preprint arXiv:2608.19652*, 2026.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 10764–10799, 2023.

Dengzhe Hou, Lingyu Jiang, Deng Li, Zirui Li, Fangzhou Lin, and Kazunori D. Yamada. WMF-AM: Probing LLM working memory via depth-parameterized cumulative state tracking. *arXiv preprint arXiv:2603.27343*, 2026.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *Proceedings of the Conference on Language Modeling (COLM)*, 2024.

Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating memory in LLM agents via incremental multi-turn interactions. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2026.

Yushi Hu, Chia-Hsuan Lee, Tianbao Xie, Tao Yu, Noah A. Smith, and Mari Ostendorf. In-context learning for few-shot dialogue state tracking. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 2627–2643, 2022. doi: 10.18653/v1/2022.findings-emnlp.193.

Najoung Kim and Sebastian Schuster. Entity tracking in language models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 3835–3855, 2023. doi: 10.18653/v1/2023.acl-long.213.

Xianxuan Long, Zhikai Chen, Shenglai Zeng, Shouren Wang, Kai Guo, and Jiliang Tang. MemTrace: Probing what final accuracy misses in long-term memory. *arXiv preprint arXiv:2606.17328*, 2026.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of LLM agents. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 13851–13870, 2024. doi: 10.18653/v1/2024.acl-long.747.

William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 35492–35506, 2024.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Egor Pakhomov and Erik Nijkamp. Parsing the stream: A live trace model for long-horizon agents and their observers. *arXiv preprint arXiv:2609.01466*, 2026.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. Logic-LM: Empowering large language models with symbolic solvers for faithful logical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 3806–3824, 2023. doi: 10.18653/v1/2023.findings-emnlp.248.

Leo Wang, Lihai Yang, Boyu Chen, Kerun Xu, Gongyi Zou, Bo Tang, Feiyu Xiong, Siheng Chen, and Zhiyu Li. Text2Mem: A unified memory operation language for memory operating system. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 2105–2119, 2026. doi: 10.18653/v1/2026.findings-acl.100.

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M. Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards AI-complete question answering: A set of prerequisite toy tasks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking chat assistants on long-term interactive memory. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025.

Takuma Yoneda, Jiading Fang, Peng Li, Huanyu Zhang, Tianchong Jiang, Shengjie Lin, Ben Picker, David Yunis, Hongyuan Mei, and Matthew R. Walter. Statler: State-maintaining language models for embodied reasoning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 15083–15091, 2024. doi: 10.1109/ICRA57147.2024.10610634.

Shuyu Zhang, Yaqi Shi, Jiarui Zhang, Yanxiao Zhao, and Lu Wang. PatchBoard: Schema-grounded state mutation for reliable and auditable LLM multi-agent collaboration. *arXiv preprint arXiv:2605.29313*, 2026.

A Benchmark Construction and Validity

This appendix documents the construction and validity checks behind the benchmark; Tables 4 and 5 define its counting units and source composition. Historical evaluation coverage, acquisition reports, extended specification audits, worked examples, full judge diagnostics, and the unrun human-audit protocol are in the *Additional Supplementary Material*.

Composition by task family and cohort. Table 5 expands the source partition of Table 4 into 15 generated task families and four platform cohorts. Generated families provide long histories with controlled state structures, while the platform cohorts add recorded event streams with shorter and more variable length profiles. The question column gives the task-level label from Section 3.1; instance-level reader resistance is reported separately in Table 8. The table documents coverage and replay validity; it does not imply that the task families are equally difficult or directly comparable.

Table 4: StateTrackBench counting units with operational definitions. 42 GitHub objects occur in both the short and the full pull: 37 share snapshot and query and form one semantic task view with two example rows, and 5 were re-pulled with a different history, which adds a second snapshot. Splits are assigned at the source-object level; the last row counts the examples scored in this paper.

unit	operational definition	count
source object	issue, pull request, package, or generated episode; the unit of the split	4,201
source snapshot	source object fixed at one event history	4,206
semantic task view	snapshot paired with one query	6,018
example	clean and noisy rendering of one view (one record)	6,055
example scored here	example scored in this paper (test split and the Section 5.3 blocks)	2,353

Table 5: Benchmark composition by task family and source cohort. L is the number of executable operations in a history; "hist." denotes a history-dependent question. Example and object counts are cohort-local: 42 GitHub objects occur in both the short and full collections, so the object column sums to 4,243, while the benchmark contains 4,201 unique source objects, 6,018 semantic task views, and 6,055 examples. Every example reproduces its stored answer and final state under deterministic replay.

task family / cohort	source	objects	examples	L med.	L max.	question
ledger	generated	100	100	119.5	256	hist.
graph	generated	100	100	122	256	hist.
location	generated	100	100	118.5	256	lookup
folder	generated	100	100	120.5	256	hist.
inventory	generated	100	100	119.5	256	hist.
access	generated	150	150	120	256	hist.
counter	generated	150	150	120	255	hist.
workflow	generated	120	120	120	256	lookup
customer support	generated	100	100	126.5	278	hist.
calendar	generated	100	100	132.5	294	hist.
incident	generated	100	100	165.5	373	hist.
location hard	generated	100	100	119	256	hist.
workflow hard	generated	100	100	150.5	356	hist.
subscription lifecycle	generated	120	120	119.5	256	lookup
key–value TTL	generated	120	120	121	256	hist.
GitHub short	platform	471	485	7	77	hist.
GitHub full	platform	1,177	2,306	11	112	hist.
GitLab full	platform	600	600	9	70	hist.
npm lifecycle	platform	335	1,004	56	149	hist.

A.1 Selection, Replay Admission, and Splits

Selection and admission. Selection uses no model outputs: GitHub histories need at least three lifecycle events, GitLab issues a reopen, and npm packages at least eight releases and one deprecation. Never-reopened GitHub issues are subsampled for reopen-count queries; assignee queries prefer users for whom the first- or most-recent-update reader fails. GitHub and GitLab histories must replay to the platform-reported state. Histories that fail mapping or replay are dropped: 86 of 1,286 fetched GitHub issues and pull requests (85 replay mismatches), and 91 of 691 reopened GitLab issues. The short GitHub pull and npm kept no rejection report.

Source order and coverage. GitHub and GitLab preserve recorded event order. npm records release times but not deprecation times, so its histories order releases by publication time and append deprecations after all releases, consistently with the registry snapshot. npm and generated gold share their construction source with replay; their admission checks establish consistency, rather than an independent final-state check. The full GitHub collection contains 1,177 reopen and 1,129 label views; its diagnostic subset has 468 and 443 views across 8 repositories, with 17.56% from the largest. The GitHub and GitLab reopen blocks have 78 and 57 histories with at least two reopens; the 600 GitLab histories span 26 projects. npm covers 335 packages and 1,004 examples.

Table 6: Rendering and source-object split construction. "Both views" counts rows containing clean and noisy histories; equal message count is checked row by row. No source object appears in more than one split.

(a) Surface views and distractors

row source	rows	both views	equal message count	injected distractor messages	rows with distractors
generated	1,660	1,660	1,660	79,908	1,660
platform	4,395	4,395	4,395	0	0

(b) Source-object split isolation

split	source objects	benchmark rows	cross-split source objects
train	3,312	4,750	0 across all three splits
development	442	655	—
test	447	650	—

Table 7: Validity controls and their scope. "Pass" denotes the stated finite or deterministic check, not a universal guarantee.

control	audited scope	status
gold replay	all 6,055 examples	pass
step-wise replay consistency	4,653 rows of the precondition audit (Appendix D)	pass
specification and reader audit	31 registered + 10,000 sampled specifications, $k \leq 5$	bounded pass
reader-resistant examples	all examples	3,463/6,055
reader-resistant, history-dependent	5,715 non-lookup examples	3,463/5,715
solved by at least one reader	5,715 non-lookup examples	2,252/5,715
privacy scan	5,595 canonical and raw records	pass
surface diagnostic	500 scored units, two model judges	diagnostic (Appendix A.3)
human surface fidelity	—	not run

Surfaces and splits. Each row's clean and noisy surfaces have equal message counts and identical operations, query, and gold. Only generated histories contain injected distractors, drawn from author-written sentences. Source-object assignment keeps every related view in one official split (Table 6); the diagnostic cohorts' separate selection and overlap with the official test split are reported in Table 11.

Governance. Platform text is rendered from event metadata, without raw issue descriptions, comments, or package source text. GitHub login and assignee values use salted pseudonyms; the salt is not distributed. The scan of 5,595 canonical and raw records found no plaintext email, handle, credential-token, or unapproved-assignee hits. Public repository, project, and package names remain for provenance.

A.2 Validity Controls

Replay and finite reader audit. Replay reproduces the stored state and answer on every released example (Table 7). The three registered readers return the specification default, first relevant update, or most recent relevant update; relevance follows the typed schema, not the gold answer. A conservative syntactic lookup classifier agrees with bounded enumeration on 31 registered specifications and 10,000 legal specifications sampled from the stated template family, with event sequences of length at most five. The check is limited to the registry, sampled template family, and enumeration bound, with degenerate specifications excluded; it is not a universal equivalence result or a guarantee against every single-message shortcut.

Instance-level resistance. Of the 5,715 history-dependent examples, 3,463 defeat all three readers and 2,252 are solved by at least one (Table 8). The 340 lookup examples are reader-solvable by design. History dependence therefore does not imply resistance for every example: many reopen histories have answer one, while other structures mix resistant and reader-solvable instances.

Table 8: Registered-reader resistance by benchmark structure. A resistant row defeats the specification-default, first-relevant-update, and most-recent-relevant-update readers. The dagger marks the three lookup controls, which are reader-solvable as intended. These finite readers do not establish that a row is free of every shortcut.

structure	n	resistant	%	structure	n	resistant	%
access	150	69	46.0	kv_store_ttl	120	96	80.0
calendar	100	88	88.0	ledger	100	100	100.0
counter	150	99	66.0	location_hard	100	100	100.0
customer_support	100	100	100.0	package_releases	335	330	98.5
folder	100	100	100.0	package_releases_active	334	167	50.0
graph	100	79	79.0	package_releases_count	335	335	100.0
incident	100	92	92.0	workflow_hard	100	63	63.0
inventory	100	87	87.0	location†	100	0	0.0
issue_lifecycle	1,600	1,357	84.8	subscription_lifecycle†	120	0	0.0
issue_lifecycle_assign	14	7	50.0	workflow†	120	0	0.0
issue_lifecycle_reopen	1,777	194	10.9				

Table 9: Dual-model surface diagnostic on 500 scored units. "Both" counts units that both judges mark adverse. Platform units show the noisy rendering of benchmark source objects; controlled-grid units come from a generator grid outside the benchmark examples. This is a model-judge diagnostic, not a human fidelity audit.

unit group	units	msg./hist.	A adverse	B adverse	both
GitHub	100	80/20	1	1	1
GitLab	100	80/20	1	2	1
npm	80	60/20	0	0	0
platform total	280	220/60	2	3	2
controlled grid, clean	70	60/10	13	14	9
controlled grid, natural	150	120/30	1	9	0
controlled grid total	220	180/40	14	23	9
all scored units	500	400/100	16	26	11

A.3 Surface Diagnostics

Sampling and blinding. Replay checks typed transitions, not whether a message faithfully expresses its event. The dual-model diagnostic scores 500 units (400 messages and 100 histories), excluding 30 calibration units. Its 280 platform units show noisy surfaces: 269 are text-identical to benchmark rows and 11 share benchmark objects but differ in text. The other 220 units come from an auxiliary generator grid outside version 1.0.0, with 70 clean and 150 natural units. No generated benchmark example was audited. Judge A (Claude Fable 5) and judge B (GPT-5.6-sol at maximum reasoning) assess the same inputs independently in fresh contexts or processes, seeing only text and queries, without typed or source events, thresholds, or model predictions. They assess ambiguity, leakage, template cues, naturalness, and whether a history admits one executable reading, under the pre-specified adverse-unit rule; naturalness at 2 or below is adverse.

Findings and limits. The judges flag 2 and 3 of the 280 platform units, and 14 and 23 of the 220 auxiliary-grid units; neither flags answer leakage. Their agreement on whether a history admits one executable reading is low ($\kappa = 0.169$), so the history-level counts do not estimate the benchmark's error rate. The 3 and 16 history flags also exceed the planned human protocol's reference limit of one; that limit was not designed for model judges, and no pass/fail decision is made. Because the judges never see the source events, this diagnostic does not validate text–event fidelity. No human audit has been run, and the paper reports no human labels.

B Evaluation Protocol

This appendix specifies the interfaces of the matched study, the evaluation blocks, the accounting, and the analysis protocol. The row manifests, the primary comparisons on the reopen cohorts, and the held-out repair protocol were fixed before their reported calls; the GitLab schema variant, the

grid fold arms, and the state-slice ablation were added after earlier outcomes were inspected and are exploratory. Every scored row records its interface implementation version except 149 PTR rows and 44 legacy GitLab rows; the full version record is in the Additional Supplementary Material. The protocol is prospectively specified for new API realizations; it is not a registered report.

B.1 Shared Conditions

Rows, surfaces, and readout. Every arm sees the same rows in the same message order. Platform blocks use the noisy rendering of the current benchmark records, and the controlled grid uses its natural rendering. All arms that produce a state (A2–A6) receive the same task-specific operation vocabulary and end with the same deterministic readout G_q on their final state. A1 answers the question directly and is a reference, not a state interface. Answers are scored by normalized match (strict), with a numeric tolerance of 0.011 after rounding to four decimals, and by a fixed semantic normalizer that accepts pre-declared aliases symmetrically across arms. Here strict excludes semantic alias mapping, not the numeric tolerance; both scores are reported. No arm inspects gold operations, states, or answers; gold is used only by the oracle and in post-run audits.

Model and decoding. All arms of the main study use GPT-4o. The repair runs (the repair variants and the state-slice ablation on the development and held-out label histories, 2,308 calls, 1,170 of them held-out) did not serialize the returned identity, so their model is known only by the configured GPT-4o label; the post-gate repair controls returned GPT-4o on all 712 calls. Decoding uses temperature 0. The replication on a second model family uses Doubao-Seed-1.8. Across the prediction files of all runs, stopped runs included and smoke tests excluded, the provider returned GPT-4o on 131,153 calls, Doubao-Seed-1.8 on 10,575 calls, GPT-5 on 6,452 calls, and Gemini-2.5-Pro on 122 calls, with no other identity; the post-gate controls add 712 GPT-4o calls.

Transport and failed requests. A semantic call is one request for one model output, and an HTTP attempt is one transport attempt; both are logged per row. A semantic call is retried for up to six HTTP attempts on rate-limit, server, and timeout errors, with exponential backoff. A row in which any semantic call still fails is not scored as model behaviour; it is removed and re-run. A run stops scheduling new rows when more than 20% of the last 200 calls fail. Rows are processed concurrently across rows and sequentially within a row, so each state-producing arm conditions every call on the outcome of the previous one.

Pilot rows. The first three rows of the GitHub reopen manifest served as a pipeline check. They are reported descriptively and excluded from inferential tests.

B.2 State Interfaces

The Additional Supplementary Material gives the complete interface procedures and prompt templates.

Prompt parity. A3, A4, and A5 receive two additions that A6 has implicitly. The first is the exact transition contract, generated from the same specification that A6's replay executes, which states for each operation type its effect on each container, including conditional status transitions and unconditional counters. The second is a set of demonstrations derived mechanically from the few-shot pairs that the A6 parser sees (10 pairs for the issue-lifecycle tasks), rendered in each arm's own format: message to state for A3, message to patch for A4, and operation to state for A5. Demonstration states are computed by replaying the demonstration operations, and demonstration patches are computed as the RFC 6902 difference between consecutive states. A2 keeps its original prompt so that it measures the earlier comparator unchanged. The prompt reference in the Additional Supplementary Material shows the contract and one demonstration in all three formats.

B.3 Evaluation Blocks

Table 12 lists the fixed row manifests. Each manifest records the row identifiers, source-object identifiers, history field, data version, evaluation and scoring code versions, model configuration, and sampling and bootstrap seed (20260925). The gold-difficulty stratum of the reopen blocks

Table 10: Per-step protocol of the six core arms. T is the number of messages in a row. "Keep" means the previous state is retained. A5 executes the final operations of A6 on the same row; steps where A6 accepted no operation are identity steps without a call.

arm	model input	model output	validation and retry	on failure	calls per row
A1 full-context read	whole history, question	answer ($\leq$ 32 tokens)	none	—	1
A2 model state (legacy)	previous state capped at 6,000 characters, message	full JSON state ($\leq$ 512 tokens)	JSON parse only; no retry	keep	T
A3 model state (matched)	uncapped previous state, message; contract and demonstrations	full JSON state, JSON mode, dynamic budget	schema check; one retry with the error	keep	T + retries
A4 JSON-patch delta	program-held state, message; contract and demonstrations	RFC 6902 patch, JSON mode ($\leq$ 512 tokens)	atomic apply and schema check; one retry with the error	reject, keep	T + retries
A5 fixed-operation fold	own previous state, A6 operation; contract and demonstrations	full JSON state, JSON mode, dynamic budget	schema check; one retry with the error	keep	accepted operations + retries
A6 PTR	message; schema and few-shot pairs	one typed operation ($\leq$ 64 tokens)	canonicalization, deterministic repairs, validity check; one re-parse of flagged messages	drop operation	T + flagged

Table 11: Diagnostic cohorts by official split. Counts are example identifiers, not source objects. The three pipeline-check rows excluded from the pre-specified reopen test comprise two train and one development example, so the test overlap is unchanged. The ledger extension lies outside v1.0.0.

cohort	train	dev	test	use
GitHub reopen (P1)	362	51	55	interface study; repair development
GitLab reopen (P2)	474	62	64	interface study; repair development
GitHub labels (P3)	343	50	50	repair development
Held-out labels (HL)	320	41	39	repair gate; post-gate controls
Total	1,499	204	208	distinct examples

consists of rows with at least two reopens; within these cohorts it coincides with the rows that defeat all registered readers, and it is not called shortcut-free.

Cohort selection and the official split. The diagnostic cohorts are separate selections, not the official train/development/test partition (Table 11). Here "held-out" means source-object separation from repair development; it does not mean the official test split. The two reopen cohorts and the label-development cohort contain 169 official-test examples, and the held-out label cohort contains 39. Thus the official test split supports the descriptive leaderboard but is not an untouched test set for every development activity in this study. Membership is determined by joining example identifiers to the released data rows. Including other task views of the same source objects, 253 official-test examples share a source object with a diagnostic block; 208 are exact example matches. The held-out label objects remain disjoint from all three repair-development blocks, whereas the label-development objects are also present in the GitHub reopen block.

Controlled ledger grid. The grid uses only the ledger family because its initial state and readout are fully supported by the benchmark's specification language; replay of gold reproduces every answer. Two other grid families were excluded before any model call because the benchmark interpreter does not support their initial-state and counterfactual readouts (oracle accuracy 0.75 and 0.00). The generator varies R while irrelevant operations and answer magnitude co-vary with it, so R is a composite generator-difficulty axis and not a causal manipulation of update count alone. The grid is outside version 1.0.0; its 100 histories will be released as an extension set upon acceptance.

Table 12: Evaluation blocks of the matched study. Messages count all rendered messages in the block. The held-out label block shares no source object with the development blocks and was not inspected before the repair code was fixed. Asterisks mark exploratory arms added after earlier outcomes were inspected; A4f executes A6's fixed operations through model-generated state patches.

block	rows	messages	content	arms
GitHub reopen	468	7,285	reopen count; 78 rows with at least two reopens	A1–A6
GitLab reopen	600	7,662	reopen count; 57 rows with at least two reopens	A1–A6
GitHub labels	443	7,156	current-label count (open vocabulary); repair development	A3, A6, repair variants
controlled ledger grid	100	4,800	48 messages per row, 8 entities, $R \in \{2, 4, 8, 16\}$ relevant updates, 25 rows per R	A1, A3, A4f*, A5*, A6
held-out GitHub labels	400	5,763	label rows outside the evaluated 443, sampled by source object	A6, repair variants, state-slice ablation*
schema variant*	600	7,662	GitLab reopen, JSON mode and schema without transition contract	A3n
second family	600	7,662	GitLab reopen with Doubao-Seed	A6 (A5 stopped after 39 rows)

Table 13: Call and token accounting on the two reopen blocks (GPT-4o). Semantic calls count model outputs; HTTP attempts include transport retries. Invalid counts first outputs that fail parsing or the schema check; fixed counts retries that produced a valid output; the last column counts steps that kept the previous state after a failed retry (A3, A5), rejected patches (A4), in-prompt cap hits (A2), or A6 messages sent to re-parse.

block	arm	calls	attempts	in tokens	out tokens	trunc.	invalid	fixed	fallback / cap / flagged
GitHub reopen	A1	468	556	0.16M	0.00M	0	—	—	—
	A2	7,285	8,730	4.27M	0.71M	0	—	—	cap 0
	A3	7,285	9,193	17.18M	0.68M	0	0	0	fallback 0
	A4	7,289	9,197	15.71M	0.28M	0	4	4	rejected 0
	A5	7,004	8,905	16.77M	0.66M	0	0	0	fallback 0
	A6	7,291	9,048	5.04M	0.14M	0	—	—	flagged 6
GitLab reopen	A1	600	717	0.19M	0.00M	0	—	—	—
	A2	7,662	9,371	4.60M	0.88M	0	—	—	cap 0
	A3	7,662	9,743	18.18M	0.84M	0	0	0	fallback 0
	A4	7,667	9,281	16.62M	0.32M	0	5	4	rejected 1
	A5	7,146	8,555	17.19M	0.78M	0	0	0	fallback 0
	A6	7,670	9,451	5.32M	0.16M	0	—	—	flagged 8

B.4 Analysis Protocol

Tests. The primary comparison is A5 against A6 on the union of the two reopen blocks without the three pipeline-check rows (1,065 histories), with net discordance defined as A6-only-correct minus A5-only-correct. It uses an exact two-sided McNemar test at $\alpha = 0.05$ and a source-object cluster bootstrap 95% interval for the accuracy difference (100,000 resamples, seed 20260925). The secondary comparisons, A3 against A6 and A4 against A6 on the same union, are Holm-corrected over two tests. A2 against A3, the label block, the grid comparisons and slope, and the offline ladder are exploratory; their p-values are nominal and unadjusted. Exploratory grid analyses report per-arm slopes; the planned interaction and answer-magnitude covariate fits were not reported.

Reporting rules. Strict and semantic scores are both reported. Accuracies of 1.000 receive Clopper–Pearson 95% intervals. All results are scoped to one captured API realization per arm; the planned repeated block was replaced by the exploratory schema variant.

B.5 Call and Token Accounting

On the two reopen cohorts, no output was truncated, the legacy arm never reached its in-prompt cap, and the matched arms never fell back to a previous state.

B.6 LEADERBOARD PROTOCOL

This section specifies the test split, model routes, requests, scoring, and baselines behind Table 3. All leaderboard analyses were added after the pre-specified interface study and are exploratory (Appendix E).

Test split. The official test split holds 650 examples from 447 source objects and shares no source object with the train and development splits (Section 3.2). Of these, 478 are platform examples: 315 from GitHub (267 from the long-history pull and 48 from the short pull), 64 from GitLab, and 99 from npm. The other 172 come from the 15 generated task families, with 7 to 17 examples per family. The platform examples fall into 230 short, 105 medium, and 143 long histories, and the generated ones into 67 short, 52 mid, and 53 long histories. Every example is rendered on the noisy surface. The evaluation records specify the example identifiers, rendering surface, and dataset version used. The LB and LBG manifests retain an outdated, redundant source-object list from the P1 template; their example identifiers and data rows are correct. Membership, split counts, and source-object clustering use the corresponding dataset rows, not that redundant list.

Generated subset. The subset used for PTR holds one generated test example per task family and length band present in the split, drawn with a fixed seed. It has 44 examples (15 short, 15 mid, and 14 long, since `inventory` has no long test example) with 49 to 447 messages. PTR with GPT-4o made 8,725 calls on it, with no retried or failed HTTP attempt.

Models and routes. Each model is scored through one provider route. The Gemini-2.5-Pro route returns no identity, so its model name is the configured label and is not verified by the provider. All four models are scored with full context and oracle updates, and GPT-5 and GPT-4o also closed book; each of these cells has 650 completed rows and zero terminal row failures after retries. A second route, configured for Gemini-3-Pro, returned Gemini-2.5-Pro before it stalled on rate limits after 93 full-context rows; it is not used, and the agreement of its answers with the first route does not verify the first route's identity, since GPT-5's answers agree with them about as often. This observation bears on the archival labels reported in the Additional Supplementary Material. Four further routes, for GPT-5.1, GPT-5.4, GPT-5.5, and DeepSeek, failed a single test call with timeouts or HTTP errors and were not used.

Requests. Each setting makes one call per example with the complete prompt templates in the Additional Supplementary Material and a 32-token answer budget. Requests set no reasoning parameter, so each model runs at its provider's default reasoning. The GPT-5 and Gemini-2.5-Pro routes are configured as reasoning deployments: their completion limit is the answer budget plus a 12,000-token reasoning allowance (12,032 tokens), and no temperature is sent. Doubao-Seed-1.8 and GPT-4o requests send temperature 0 and a 32-token limit. The Doubao route does not apply this limit to reasoning: its completions reach 25,212 tokens and no answer is cut. The Gemini route reports at most 23 completion tokens per call, so any reasoning it performs is not visible in the token accounting. A GPT-5 response that spends the whole allowance returns no answer and is scored as an error; this happens on 6 full-context and 3 oracle-update examples, and on no example for the other models.

Retries, row failures, and scoring. A call is retried for up to six HTTP attempts on rate-limit, server, and timeout errors, as in the interface study (Appendix B). A row in which a call still fails is removed and re-run, a model's cell enters Tables 3 and 14 only when all 650 rows have completed, and a family or cohort cell of Table 15 only when all of its rows have, so no row whose call failed terminally is scored as an empty answer. Across the leaderboard, generated-subset, and grid-slice runs, the 19,627 semantic calls took 1,737 extra HTTP attempts, 1,625 of them retries after rate-limit (HTTP 429) responses, all on the GPT-5, Gemini-2.5-Pro, and Doubao-Seed-1.8 routes; 0 rows failed terminally, and 9 answers, all from GPT-5, were cut at the completion budget (`request_accounting.py`). Answers are scored by normalized answer match after lowercasing, trimming, and collapsing whitespace; numeric answers are rounded to four decimals and match when they differ by at most 0.011. No alias normalizer is applied, and an empty answer is an error.

Strata. GitHub pools the two GitHub pulls, and generated pools the 15 families. Resistant holds the 369 examples on which all three registered readers of Section 3.3 are wrong: the specification default, the first relevant update, and the most recent relevant update, each run on the gold updates. Long holds the 196 examples of the long band of each source (143 platform and 53 generated), with 20 to 139 updates on the platforms and 244 to 352 in generated families. Resistant and long cut across sources, so the strata overlap.

Intervals. Intervals are 95% percentile intervals of a cluster bootstrap over the 447 source objects. Each of 20,000 resamples (seed 20260925) draws objects with replacement and computes accuracy over all examples of the drawn objects. Intervals are reported for the whole split only, and no significance test is run. With full context the intervals are $[0.949, 0.978]$ (GPT-5), $[0.919, 0.956]$ (Gemini-2.5-Pro), $[0.894, 0.938]$ (Doubao-Seed-1.8), and $[0.539, 0.613]$ (GPT-4o), and with oracle updates $[0.978, 0.995]$, $[0.951, 0.980]$, $[0.961, 0.985]$, and $[0.653, 0.725]$. Adjacent full-context intervals of GPT-5, Gemini-2.5-Pro, and Doubao-Seed-1.8 overlap.

Answer prior. The answer prior is the no-history baseline for the closed-book probe. For a test example, it returns the most frequent lower-cased gold answer among the train-split examples of the same task and question template. The template is the question text with the query's string arguments replaced by a placeholder and every digit run replaced by one symbol, truncated to 80 characters. When no train example shares the template, the prior falls back to the most frequent train answer of the task. It sees neither the history nor the object, and it is scored with the same normalized match; it reaches 0.362 on the whole split. Table 14 gives the closed-book scores by stratum next to the prior. Gold reflects each object's state at the June–July 2026 pull, which postdates both model snapshots, and some test objects were opened after them. On npm both models exceed the prior (0.253 and 0.212 against 0.172), almost entirely on yes/no questions about whether a version is still active, so recall is not excluded there, although both stay below their full-context accuracy.

B.7 Information Differences Between Settings

The leaderboard scores each example in three settings that differ only in what stands in the prompt where the history would be. Closed book (A0) gives the query alone, full context (A1) the rendered messages, and oracle updates (A7) the gold typed updates with the task's operation vocabulary, compiled transition contract, and initial state. All three share the system prompt and answer instruction of the full-context read (see the Additional Supplementary Material), make one call, and take the model's answer as the prediction, with no program readout.

Full context (A1). The model interprets every message, accumulates the updates, and reads out the answer in one call. It receives no transition contract, so it must infer the task's semantics from the question and the messages.

Oracle updates (A7). The gold updates replace message-to-update inference, and the model still has to accumulate them and read out the answer in one call. The contract is generated from the same specification as the contracts of the matched interfaces (Appendix B), and the vocabulary block is the one in PTR's parser prompt, so it still describes messages and asks for a JSON object per message, while the system prompt and the question ask for the answer only. A7 is therefore treated as a bundled setting comparison, not as a clean ablation of interpretation alone. A7 differs from A1 in more than interpretation: it states the transition function, supplies the initial state, drops the messages without an update, and presents typed updates instead of text. The gap between A1 and A7 is therefore the net effect of the bundled oracle-update intervention, not the cost of interpretation alone (Section 3.3). A7 lists the updates as v1.0.0 stores them, before the key derivation that the program applies to `inventory` and npm updates.

Closed book (A0). Without the history, a model can answer correctly only by recalling the public record or by guessing well. A0 is a contamination probe, not a capability measure, and it is read against the answer prior, which uses train answers but no object (Appendix B.6). A model that recalls a record but does not express it without the history is not detected.

Oracle initial state and ledger conventions. The program replays each history from an initial state s_0, which is empty except where a task seeds it. In the test split only the `ledger` family seeds a state: the group of people among whom a shared expense is split. The contract of the `paid` update divides the amount among the declared participants and, when none are declared, charges only the payer. The first A7 prompt listed the updates and the contract but not this initial state, so a model that followed the contract literally would charge every shared expense to its payer alone. The messages do not supply the group either: they refer to "the whole group", and in none of the 8 test `ledger` histories does a message name more than two people, so a full-context reader must collect the members from the names across the history. The fixed prompt adds the initial state whenever a task seeds one. The 8 `ledger` rows were re-run with it for GPT-5, Doubao-Seed-1.8, and GPT-4o, and every Gemini-2.5-Pro oracle-update row ran after the fix. The pre-fix predictions are kept. The fix raises GPT-5 on these rows from $0/8$ to $7/8$ and on the whole split from 0.977 in the pre-fix file, over its 647 completed rows, to 0.988, and Doubao-Seed-1.8 from 0 of its 5 completed pre-fix rows to $4/8$ and from 0.972 over 647 completed rows to 0.974. GPT-4o answers none of the 8 rows before or after the fix.

C Detailed Results and State Diagnostics

C.1 Leaderboard Results by Stratum and Family

Reading the family table. GPT-5 makes 23 errors with full context: 3 on GitHub and 20 on generated histories, 18 of them in `customer_support` (6), `incident` (6), `ledger` (3), and `workflow_hard` (3). Six of the 23 are empty answers that exhausted the reasoning allowance (2 in `incident`, 2 in `ledger`, and 1 each in `workflow_hard` and `folder`), and so are 3 of GPT-5's 8 oracle-update errors (1 each in `folder`, `graph`, and `ledger`). With oracle updates, GPT-5 answers every `customer_support` and `incident` example, so these errors vanish under the bundled oracle-update setting. Oracle updates are not easier in every family: GPT-5 falls on `folder` (10 to 6 of 11, one of the five errors an empty answer) and on `graph`, where its single error is an empty answer, and GPT-4o falls on four families, including `customer_support`. Gemini-2.5-Pro and Doubao-Seed-1.8 show the same pattern in `customer_support` and `incident`, where oracle updates cut their errors from 12 and 11 to 1 each. Gemini-2.5-Pro falls on npm with oracle updates (0.939 to 0.879), `calendar`, and `graph`, Doubao-Seed-1.8 falls on `calendar` and npm, and it answers no `ledger` example with full context and 4 of 8 with oracle updates. Across the split, GPT-5 is correct only with oracle updates on 21 examples and only with full context on 6, and GPT-4o on 90 and 17; for GPT-5, 4 of the 21 are full-context empty answers and 1 of the 6 an oracle-update empty answer. Full-context errors that persist under oracle updates number 2 of 23 for GPT-5, 10 of 40 for Gemini-2.5-Pro, 13 of 54 for Doubao-Seed-1.8, and 185 of 275 for GPT-4o. Oracle updates also state the transition function, supply the initial state, and drop messages without an update (Appendix B.7), so these counts measure the net effect of the setting, not of interpretation alone. GPT-4o answers no `ledger` and no `location_hard` example in either setting. On this split, one registered reader answers every example of `location`, `subscription_lifecycle`, and `workflow`, so these three families contribute no example to the resistant stratum. On the generated subset, PTR with GPT-4o is correct where GPT-5 with full context is wrong on 7 histories and never the reverse; against GPT-5 with oracle updates the counts are 2 and 0, against Gemini-2.5-Pro 9 and 0 with full context and 1 and 0 with oracle updates, against Doubao-Seed-1.8 14 and 0 and 5 and 0, and against GPT-4o 30 and 0 and 24 and 0. Two of GPT-5's 7 full-context errors and 1 of its 2 oracle-update errors on this subset are empty answers.

Table 14: **Closed-book accuracy by stratum.** Normalized answer-match accuracy with the query alone (A0), and the answer prior, which returns the most frequent train-split answer to the same question template; parentheses give the number of examples. Resistant examples defeat the specification-default, first-relevant-update, and most-recent-relevant-update readers on gold updates; long examples belong to each source's long-history band. The probe detects only recall that a model can express without the history, so a low score does not show that a record is absent from pretraining data.

	all (650)	GitHub (315)	GitLab (64)	npm (99)	generated (172)	resistant (369)	long (196)
answer prior (train)	0.362	0.381	0.875	0.172	0.244	0.201	0.260
GPT-5, closed book	0.251	0.289	0.391	0.212	0.151	0.190	0.189
GPT-4o, closed book	0.132	0.152	0.047	0.253	0.058	0.081	0.143

Table 15: **Leaderboard accuracy by generated family and platform cohort.** Normalized answer-match accuracy (higher is better) with full context (A1) and oracle updates (A7); n is the number of test examples. Reader: the best of the three registered shortcut readers within the row, run on the gold updates. Leading zeros are omitted, and 1.00 marks a cell with every example correct. GitHub (full) and GitHub (short) are the long-history and short pulls. Families hold 7 to 17 examples, so one example moves a family cell by 0.06 to 0.14; the table locates errors and does not rank models within a family.

family / cohort	n	GPT-5		Gemini-2.5-Pro		Doubao-Seed-1.8		GPT-4o		reader
		full	oracle	full	oracle	full	oracle	full	oracle	
GitHub (full)	267	.993	1.00	.981	1.00	.951	1.00	.715	.854	.521
GitHub (short)	48	.979	1.00	.958	1.00	.917	1.00	.479	.875	.062
GitLab	64	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	.875
npm	99	1.00	1.00	.939	.879	1.00	.990	.394	.455	.172
access	12	1.00	1.00	.833	1.00	.750	.750	.500	.417	.417
calendar	14	1.00	1.00	.929	.857	.929	.857	.286	.286	.143
counter	16	1.00	1.00	1.00	1.00	.938	1.00	.438	.375	.312
customer_support	11	.455	1.00	.455	.909	.455	1.00	.636	.364	.000
folder	11	.909	.545	.636	.636	.455	.455	.091	.182	.000
graph	7	1.00	.857	1.00	.857	1.00	1.00	.286	.857	.000
incident	8	.250	1.00	.250	1.00	.375	.875	.000	.500	.125
inventory	11	1.00	1.00	1.00	1.00	.818	1.00	.364	.636	.182
kv_store_ttl	12	1.00	1.00	1.00	1.00	1.00	1.00	.250	.250	.250
ledger	8	.625	.875	.625	.750	.000	.500	.000	.000	.000
location	12	1.00	1.00	1.00	1.00	.917	1.00	.583	.583	1.00
location_hard	10	1.00	1.00	1.00	1.00	1.00	1.00	.000	.000	.000
subscription_lifecycle	12	1.00	1.00	1.00	1.00	1.00	1.00	.250	.083	1.00
workflow	17	.941	1.00	.941	1.00	.941	1.00	.647	.824	1.00
workflow_hard	11	.727	.909	.636	1.00	.727	1.00	.273	.545	.273

C.2 Matched Interface Comparisons

This section gives the GPT-4o interface results that Section 5.3 states in prose: the interface ladder on the platform reopen histories (Table 16), the controlled ledger grid (Table 17), and GitHub label counting (Table 18). All cells come from one API realization at temperature 0.

Table 16: **Interface ladder on platform reopen histories (GPT-4o).** Normalized answer-match accuracy; higher is better. "Hard": at least two reopens, which here coincides with defeating every registered reader. A5 executes A6's parsed updates, and the 1.000 cells are a ceiling, not a ranking. Input tokens are averaged per history.

		GitHub reopen		GitLab reopen		input tokens
	interface	full (468)	hard (78)	full (600)	hard (57)	per history
A1	full-context read	0.951	0.718	0.975	0.825	0.3k
A2	model state (legacy)	0.885	0.756	0.870	0.632	8.3k
A3	model state (matched)	**1.000**	**1.000**	**1.000**	**1.000**	33.1k
A4	JSON-patch delta	**1.000**	**1.000**	**1.000**	**1.000**	30.3k
A5	fixed-operation fold	**1.000**	**1.000**	**1.000**	**1.000**	31.8k
A6	PTR	**1.000**	**1.000**	**1.000**	**1.000**	9.7k

Table 17: **GPT-4o on the controlled ledger grid, outside v1.0.0 (an extension set).** Each history has 48 messages and 32 transfers among eight people; R transfers touch the queried person, and irrelevant transfers and answer magnitude co-vary with R. The full-context reader does not see the ledger's sign convention, which the other interfaces receive in their operation schema. The fold arms execute PTR's parsed operations and were added after the other grid results were seen; the asterisked patch fold was added last. Each R cell has $n = 25$, and 32 of the 100 answers are 0.

interface	$R = 2$	4	8	16	all (100)
full-context read	0.60	0.28	0.20	0.32	0.350
model state (matched)	0.92	0.80	0.80	0.92	0.860
fixed-operation fold	0.96	0.84	0.56	0.52	0.720
fixed-operation patch fold*	0.96	0.80	0.80	0.84	0.850
PTR	1.00	1.00	1.00	1.00	1.000

Table 18: **GPT-4o on GitHub current-label counting.** The repair was developed on the development cohort, fixed, and evaluated once on the held-out histories; matched model state was not run on the held-out cohort. The daggered control re-parses exactly the repair's alarmed observations with PTR's generic instruction, and the asterisked ablation uses the repair's prompt minus the state slice; both use the repair's calls and were run after the held-out results. Accuracy uses normalized answer match; higher is better.

interface	development (443)	held-out (400)
model state (matched, A3)	0.998	–
PTR (A6)	0.962	0.918
+ generic re-parse at the alarms†	–	0.938
+ replay-grounded repair	0.998	0.990
without the state slice*	**1.000**	**0.993**

C.3 STRONG-MODEL CHECK ON THE LEDGER SLICE

The ledger grid holds 100 histories of 48 messages and 32 transfers among eight people, with 25 histories for each number $R \in \{2, 4, 8, 16\}$ of transfers that touch the queried person (Section 5.3.2, Figure 3a); it is generated outside version 1.0.0 and will be released as an extension set upon acceptance. To test whether GPT-4o's transition errors (Section 5.3.2) depend on the model, GPT-5 was run on the 25 histories with $R = 16$ under matched model state (A3), the fixed-operation patch fold (A4f), the fixed-operation fold (A5), and PTR (A6), and with full context (A1) on all 100 grid histories. Requests set no reasoning parameter. GPT-5's PTR parses equal GPT-4o's on all 25 histories, so the fold arms of the two models execute identical operations. GPT-5 made 1,200, 802, 800, and 1,600 calls under A3, A4f, A5, and A6, with no terminal row failure and no truncated answer; the calls took 921 extra HTTP attempts, 920 of them after rate-limit (HTTP 429) responses (Appendix B.6).

Table 19: **GPT-5 and GPT-4o on the $R = 16$ slice of the ledger grid.** Correct histories out of 25 (higher is better). GPT-4o values come from the grid runs of the interface study on the same histories; GPT-5 values come from runs on the slice, except full context, which is taken from GPT-5's run on all 100 grid histories. The fold arms execute PTR's parsed operations, which are identical for the two models on these histories. Each cell is one realization; the grid lies outside v1.0.0, and 25 histories at one value of R do not bound GPT-5's transition errors on longer or denser histories.

interface	GPT-4o	GPT-5
full-context read	8/25	25/25
model state (matched)	23/25	25/25
fixed-operation patch fold	21/25	25/25
fixed-operation fold	13/25	25/25
PTR	25/25	25/25

Results. GPT-5 answers all 25 histories under every interface, while GPT-4o answers 8 with full context, 23 with matched model state, 21 with the patch fold, 13 with the fold, and 25 with PTR (Table 19). With full context, GPT-5 answers 99 of the 100 grid histories; its one error, at $R = 4$, has the wrong sign (16 for -16), and the full-context prompt does not state the sign convention that the other interfaces receive. On this slice GPT-5 makes no transition error where GPT-4o's fold makes 12, but 25 histories bound GPT-5's per-interface error rate only below 14% (95% Clopper–Pearson upper bound), which does not separate it from GPT-4o's matched model state or patch fold; replay of the same operations is exact for both.

C.4 STEP-LEVEL DIAGNOSTICS

This appendix defines the step-level diagnostics of Section 5.3.3 and reports them for the four platform blocks of the interface study and the generated subset; the ledger grid is not included. The diagnostics make no model call; they compare PTR's stored parses with the gold updates and replay both.

Replay tool. The function `replay_trajectory` in `step_diagnostics.py` regenerates the gold states $s_0^*, \ldots, s_T^*$ of one example. It creates the task's containers, adds the seeded initial state, and applies each gold update in turn with the guards and effects of the specification, recording the state after every observation, `no_update` included. It is the incremental form of the interpreter's fold, and the two agree on every example of version 1.0.0: over the 6,055 examples the tool regenerates 438,089 states, one per observation, 358,181 of them after an update other than `no_update`, and on 6,055/6,055 examples its last state equals the fold's final state and yields the gold answer. Version 1.0.0 stores the final state; intermediate states are regenerated on demand. Extended worked examples appear in the Additional Supplementary Material.

Definitions. PTR emits one typed update $\hat{e}_t$, possibly `no_update`, for each observation x_t, and both $\hat{e}_t$ and e_t^* are put in the canonical form defined by the task's specification before comparison. *Update-level accuracy* is the fraction of observations with $\hat{e}_t = e_t^*$; agreement on `no_update` counts as correct. A divergent update is an *omission* when $\hat{e}_t$ is `no_update` and e_t^* is not, a

Table 20: **Step-level diagnostics of PTR (GPT-4o) against replayed gold states.** Observations: one gold operation is compared per observation, including `no_update`; none of the platform observations is `no_update`, while 2,063 of the 8,561 generated observations are. Update acc.: fraction of observations whose canonical parsed update equals gold (higher is better). Omitted: gold updates parsed as `no_update`; other wrong: gold updates parsed as a different update. Answer errors (located): wrong final answers, with the number whose history has a first divergent update in parentheses. Correct but corrupted: correct answers whose replayed final state differs from gold. The platform blocks comprise GitHub and GitLab reopen histories and GitHub label histories; the generated subset contains one test example per available task-family and length-band combination. The diagnostics cover PTR with one model and one realization per block.

block	histories	observations	update acc.	omitted	other wrong	answer errors (located)	correct but corrupted
GitHub reopen	468	7,285	0.961	281	2	0 (0)	84
GitLab reopen	600	7,662	0.932	516	3	0 (0)	162
GitHub labels (dev.)	443	7,156	0.962	272	2	17 (17)	56
GitHub labels (held-out)	400	5,763	0.949	291	4	33 (33)	97
generated subset	44	8,561	0.988	0	106	0 (0)	1

wrong update when both are updates that differ, and a commission when only $\hat{e}_t$ is an update; no commission occurs in these blocks. The *first divergent update* of a history is its earliest divergent update, and an answer error is *located* when its history has one. The first divergent update is the earliest departure from gold, which need not be the update that changes the answer. A history is *correct but corrupted* when PTR answers it correctly, it has a divergent update, and replaying PTR's updates gives a final state that differs from the replayed gold final state, with numbers compared by value.

Reopen histories. PTR answers all 1,068 reopen histories correctly, with update-level accuracy 0.961 on GitHub and 0.932 on GitLab, yet 84 of 468 and 162 of 600 histories end in a corrupted state (Table 20). Of the 802 divergent updates, 797 are omissions. On GitHub the omitted updates are 152 openings, 98 assignments, 16 label additions, and 15 unassignments; on GitLab they are 351 label additions, 153 openings, and 12 label removals. Every omitted type is in PTR's operation vocabulary, and the reopen readout reads none of them, so the final answer cannot reveal the omissions. The omissions concentrate in a few noisy-surface paraphrases. Every omitted opening in the two blocks is rendered as "{issue} just landed in the tracker.", every omitted label addition as "{issue} picked up the '{label}' tag.", and 82 of the 98 omitted GitHub assignments as "{user} took ownership of {issue}."

Wrong updates. All 11 wrong updates on the four platform blocks are label additions that PTR records as removals. In each, the label name contains a word that suggests removal, as in `PullApprove: disable`, `featureflag::disabled`, and `automation:quick-win-removed`.

Label histories. On label counting the omissions reach the answer. All 17 development errors and all 33 held-out errors are located. On the held-out histories the first divergent update is an omission in 29 errors and a wrong update in 4, and these 4 are the only wrong updates in the block. On the development histories the first divergent update is an omission in all 17 errors. In one of them, an omitted opening precedes the inverted label addition that changes the count, so the error-type count of Section 5.3.3 (16 omissions and one inversion) differs from this first-divergence count. A further 56 development and 97 held-out histories are correct but corrupted.

Generated subset. On the 44 generated histories, PTR answers every history correctly with update-level accuracy 0.988. Its 106 divergent updates fall in 13 histories and are all wrong updates; none is an omission. In 12 of these 13 histories the replayed final state still equals gold, so a wrong update need not change the state, and one history is correct but corrupted.

Algorithm 1 PTR with replay-grounded repair

Require: observations $x_{1:T}$, final PTR operations $\hat{e}_{1:T}$, transition F, initial state s_0, query keys K_q

$\hat{s}_0 \leftarrow s_0$
for $t = 1$ to T **do**
  $K \leftarrow K_q \cup \{\text{key}_\tau(\hat{e}_j) : j < t,\ \hat{e}_j \neq \texttt{no_update}\}$
  if $\hat{e}_t = \texttt{no_update}$ and x_t mentions an eligible $k \in K$ **then**
    $e' \leftarrow P_\theta(x_t;\ \hat{s}_{t-1}[k], \text{“mentions } k \text{ but read as no update”})$
    if e' is well formed and raises no precondition alarm at $\hat{s}_{t-1}$ **then**
      $\hat{e}_t \leftarrow e'$
    end if
  else if a precondition of $\hat{e}_t$ fails at $\hat{s}_{t-1}$ on key k **then**
    $e' \leftarrow P_\theta(x_t;\ \hat{s}_{t-1}[k], \text{conflict})$
    if e' is well formed and $e' \neq \hat{e}_t$ **then**
      $\hat{e}_t \leftarrow e'$
    end if
  end if
  $\hat{s}_t \leftarrow F(\hat{s}_{t-1}, \hat{e}_t)$
end for
return $G_q(\hat{s}_T)$

D Replay-Grounded Repair and Controls

PTR replays every accepted operation exactly, but replay cannot correct a misinterpreted update on its own. This appendix asks whether the program state can flag such errors, audits candidate preconditions against gold, and specifies the replay-grounded repair of Section 4. The repair was developed on the label development cohort and evaluated once on held-out histories under the gate in Section D.2.

D.1 Replay-Grounded Repair

The repair runs during replay of PTR's final operations and uses only the observations and operations up to the current step. Two alarms are derived from the specification. A *precondition alarm* fires when an effect of the operation does not apply to the replayed state: a status transition whose current status is not an allowed predecessor, adding a member that is already present, or removing a member that is absent. For a non-null operation e, let $\text{key}_\tau(e)$ be the value of its first declared field in the operation schema; `no_update` contributes no key. Only string-valued keys of at least three characters are eligible, and a mention is a literal substring match. An *omission alarm* fires when an observation is parsed as `no_update` although it mentions an eligible tracked key: a query argument or $\text{key}_\tau(e)$ for an earlier parsed operation. If several keys match, the implementation takes the first in set iteration order. In the held-out label block, every recorded omission alarm selected the queried issue key. Each alarm triggers one re-parse of the alarmed observation with the PTR parser prompt, extended by the affected slice of the replayed state and a description of the conflict. For an omission alarm, the re-parsed operation is accepted only if it is well formed and raises no precondition alarm. For a precondition alarm, a well-formed re-parsed operation that differs from the original replaces it. The replacement need not clear every naive precondition, because these checks request reconsideration rather than reject an otherwise valid platform event. Alarms never reject an operation by themselves, because Table 23 shows that correct platform events violate naive preconditions. The gate also evaluated a retrospective step: when a precondition alarm on a value persists after the re-parse, the most recent earlier observation that contains the value and whose operation is absent or of another type is re-parsed once, and the revision is kept only if both alarms clear after replay. This step changed no answer and is not part of the method.

Positioning. Commit-time validation rejects an update that fails a check, such as the RFC 6902 `test` operation (Bryan & Nottingham, 2013) or a kernel invariant (Zhang et al., 2026; Cui et al., 2026), and solver feedback drives re-formalization in Logic-LM (Pan et al., 2023). Here the checks run during replay: the precondition alarm reads the executor's replayed state, the omission alarm compares the parse with the keys the executor tracks, and either alarm triggers a conflict-aware re-

interpretation instead of rejecting the update. The matched ablation shows no observed benefit from also showing the model its state slice. The post-gate controls locate the active ingredient in the stated conflict ($22/0$ against a generic re-parse of the same observations at equal calls), while targeting by the alarms alone is not significant against random targets ($7/2$, $p = 0.18$). The repair is therefore a replay-targeted, conflict-stating re-parse, close to verifier-feedback retries such as Logic-LM's, and not a new state-conditioning mechanism.

D.2 Evaluation Protocol and Gate

Development and held-out data. The repair was developed only on traces from the two reopen blocks and the GitHub label block. Its code and prompts were then fixed, and it is evaluated once on the 400 held-out GitHub label rows, whose source objects are disjoint from all development blocks.

Controls. The pre-specified controls were intended to use the same flagged messages and the same number of calls as the repair; the message-only control did not meet this requirement, as detailed below and in the full deviation record in the Additional Supplementary Material. They are a message-only re-parse, which uses PTR's generic re-parse instruction without the state slice or the conflict, the evaluated repair, and the repair with the retrospective step; a matched ablation that removes only the state slice from the repair's prompt was added afterwards. The pre-specified reject-only and random-target controls, and a generic re-parse at exactly the repair's alarm targets, were run after the gate with the repair's per-history call budget. Reject-only drops every update that raises a precondition alarm on the current state, as commit-time validation would, and needs no model call. The random-target control re-parses, on each history, as many observations as the repair did, drawn uniformly with a seed derived from the row identifier, using PTR's generic re-parse instruction; the generic control uses the same instruction at the repair's targets. With the state-slice ablation, these controls form a ladder in which each step changes one component: target selection (random to alarmed), the prompt (generic to conflict-stating), and the state slice.

Gate. Three tests on the held-out rows are Holm-corrected: the repair against PTR, the repair against the message-only re-parse, and the retrospective step on against off. A test passes when the net discordance is positive, the exact McNemar p-value is below 0.05 after correction, and the paired cluster-bootstrap 95% interval excludes zero. Repaired and newly broken rows are reported separately. On the reopen blocks, which served as development data, a lower paired-interval bound of at least -1.0 percentage points serves as a non-inferiority diagnostic only. The claim scope is held-out GitHub label trajectories. If the held-out rows contain too few PTR errors, the gate fails by rule.

D.3 Results

G1 passes; G2 passes nominally against a control that deviated from its specification, and the matched ablation does not support the state-slice hypothesis it was meant to test; G3 fails. The gate tested the variant with the retrospective step, which changed no answer, so the repair is reported with identical outcomes. Replay-grounded repair raises held-out accuracy from 0.918 to 0.990 with 29 histories fixed and none broken, and it is 5.3 points above the message-only re-parse policy, whose prompt, calls, and later alarms differ. The matched ablation that removes only the state slice reaches 0.993 on the held-out rows and 1.000 on the development rows, so we observe no benefit from the state slice in this realization. Of the 15 held-out repositories, 10 do not occur in development; on their 302 histories the repair raises correct answers from 270 to 298 (0.894 to 0.987). Over all 400 held-out histories, a repository-cluster bootstrap gives a 95% interval of $[2.24, 14.17]$ points for the gain (20,000 resamples, seed 20260925), and leaving out one repository at a time gives gains from 3.95 to 8.24 points (`repo_cluster_ci.py`). The post-gate controls reach 0.918 (reject-only), 0.925 (random targets) and 0.938 (generic re-parse at the repair's targets); against the last, which shares the repair's targets and calls, stating the conflict gives $22/0$ discordant histories ($p = 4.8 \times 10^{-7}$), while alarm targeting alone gives $7/2$ against random targets ($p = 0.180$). The retrospective step changes no answer. All four remaining errors are alarmed but not repaired, and no error is introduced by a repair (Table 21).

Table 21: Replay-grounded repair on the held-out GitHub label rows ($n = 400$). Accuracy uses normalized answer match; higher is better. Discordance is variant-only-correct / PTR-only-correct, the interval is the paired cluster-bootstrap 95% interval of the difference from PTR, and extra calls are per history. "Dev. Δ" is the difference from PTR on the development label cohort, where the method was designed. The asterisked ablation and the daggered controls were run after the gate; the daggered controls use the repair's per-history call budget.

variant	acc.	disc.	p (Holm)	CI (pp)	extra calls	dev. Δ (pp)
PTR	0.918	—	—	—	0	—
message-only re-parse	0.938	8/0	—	[0.8, 3.5]	1.16	+1.8
replay-grounded repair (method)	0.990	29/0	1.1×10^{-8}; 1.9×10^{-6} (vs. message-only)	[4.8, 10.0]	0.89	+3.6
+ retrospective re-parse	0.990	29/0	1.00 (vs. repair)	[4.8, 10.0]	0.90	+3.6
repair without state slice*	0.993	30/0	—	—	0.89	+3.8
reject-only (commit-time validation)†	0.918	0/0	—	—	0.00	+0.0
random targets, generic prompt†	0.925	3/0	—	—	0.89	—
repair targets, generic prompt†	0.938	9/1	—	—	0.89	—

remaining errors of the method on held-out rows: 0 without any alarm, 4 alarmed but not repaired, 0 introduced by a repair

D.4 Alarm Selectivity

Every held-out observation carries a gold update, so this cohort cannot test whether the omission alarm distinguishes omitted updates from correct null parses. We therefore replay PTR's stored final updates (GPT-4o) on five blocks with the frozen alarm rules of the repair, without any re-parse or model call, and compare each alarmed observation with gold (Table 22). An alarm is a hit when PTR's update differs from gold at that observation and a false alarm otherwise. No re-parse is simulated, so the counts describe the trigger on PTR's updates, not the repair's calls; once a re-parse restores an omitted update, later alarms that it caused no longer fire.

On the generated subset, PTR omits no update, and the omission alarm never fires, including on the $2{,}063$ observations without a gold update. The precondition alarm fires 665 times there, 5 times on parse errors, and 659 of its 660 false alarms fall on gold updates that themselves violate the naive preconditions, as legitimate events can (Appendix D). On the four platform blocks, the two alarms together flag every parse error, and the omission alarm fires only on null parses, which are all errors there. The precondition alarm also fires on 196 to 250 correct updates per block; 35 to 63 of these are gold updates that violate the naive preconditions, and the rest follow an earlier parse error that changed the replayed state. Because no alarm rejects an update, a false alarm costs one re-parse call and changes the update only if the re-parse differs; on the held-out histories the repair broke no answer (0 broken against PTR). These counts cover one model and one realization per block.

Table 22: **Offline selectivity of the replay-time alarms on PTR's updates (GPT-4o).** Obs.: observations; no-upd.: gold `no_update` observations; errors: observations whose parsed update differs from gold. Recall: share of errors with an alarm. False alarms are precondition alarms on correct updates; parentheses give those at which the gold updates also violate the naive preconditions. No model call is made; the counts describe the trigger, not the repair's calls.

block	histories	obs.	no-upd.	errors	omission	precondition	recall	false alarms
generated subset	44	8,561	2,063	106	0	665	0.047	660 (659)
held-out labels	400	5,763	0	295	291	200	1.000	196 (55)
development labels	443	7,156	0	274	272	233	1.000	231 (56)
GitHub reopen	468	7,285	0	283	281	243	1.000	241 (63)
GitLab reopen	600	7,662	0	519	516	253	1.000	250 (35)

D.5 Gold Precondition Audit

Setup. A precondition is a predicate $\mathrm{pre}(e, s)$ on an operation and the state before it, for example "`close` requires the issue to be open". Gold operations are correct by construction, so any gold operation that violates a precondition is a false alarm of that precondition as a parse-error detector. The audit replays gold operations step by step and evaluates every candidate precondition on every operation. The audit manifest contains $4{,}653$ rows, and step-wise replay reproduces the stored final state and answer on all of them. It is a selected audit scope rather than an exhaustive partition of version 1.0.0. Separately, $1{,}379$ rows are excluded to preserve the held-out seal because their source snapshots belong to the held-out label pool or share gold operations with it: 686 label rows, the 686 reopen views of the same objects, and 7 short-collection rows. The remaining 23 benchmark rows are unselected GitHub reopen examples outside the audit subsets. For each precondition, we report the rows with at least one violating gold operation and the rows whose answer would change if violating operations were rejected online ("harm").

Naive preconditions have gold violations. Table 23 shows that intuitive lifecycle preconditions are violated by gold operations in 61 of 468 GitHub reopen rows, 54 of 443 GitHub label rows, and 26 of 600 GitLab rows. Most violations are redundant platform events. On the GitHub reopen rows, 50 violating operations close an already merged pull request and 8 close an already closed issue; neither changes the state. On GitLab, 19 operations close an already closed issue, and 2 reopen an already open issue. The two reopen events still increment the reopen counter under the benchmark semantics, so rejecting them would change the answer on 2 rows. A precondition that forbids any change after a merge is violated by 33 state-changing gold operations on the label rows, and rejecting them would change 7 of 443 answers. Rejection by naive preconditions would therefore discard correct platform events.

The remaining invariant audit and parse-error decomposition are provided in the Additional Supplementary Material.

E Limitations and Protocol Deviations

E.1 Protocol Deviations

Only the reopen comparisons and the held-out repair gate were pre-specified; the leaderboard, ledger folds, strong-model slice, schema variant, matched repair controls, and state-slice ablation were added after earlier results were inspected. A planned repeat of the GitHub comparison was replaced by the schema variant, and the second-family fold stopped at 39 histories, all correct; its completed PTR results do not establish matched executor equivalence. The three pilot rows are excluded from the reopen tests.

The gate's message-only control used different prompts, targets, and calls (462 versus 356), so its result does not isolate the state slice. Matched controls were run after the gate; they support conflict feedback but show no added benefit from the state slice or a significant targeting advantage over random observations. The retrospective pass changed no answer, the planned repair non-inferiority

Table 23: Gold violations of naive lifecycle preconditions. Each cell gives rows with at least one violating gold operation, and in parentheses the rows whose answer changes if violating operations are rejected. "Status" requires `close` on an open item, `reopen` on a closed item, and `merge` on an open item; "label" requires `label` on an absent label and `unlabel` on a present one; "all candidates" adds assignee, milestone, and post-merge preconditions.

precondition bundle	GitHub reopen (468)	GitHub labels (443)	GitLab reopen (600)
status	57 (0)	50 (0)	21 (2)
label	4 (0)	4 (0)	6 (0)
status + label	61 (0)	54 (0)	26 (2)
all candidates	68 (0)	61 (7)	27 (2)
any operation without effect	68 (0)	61 (0)	26 (0)

diagnostic on reopen histories was not run, and no matched model-state baseline was evaluated on the held-out label cohort.

The interface study includes 149 PTR rows and 44 legacy GitLab rows without a recorded implementation version, and one GitHub fold row was re-run under a later version. The archived versions retain identical A1–A6 functions, but the planned mixed-cell rerun was not performed. The oracle initial-state correction, model-route evidence, unequal budgets, terminal-retry policy, and diagnostic/test overlap are documented alongside the protocol. A semantic alias normalizer was added after the original strict summaries and applied symmetrically; both scoring variants are reported. The full deviation record, including earlier evaluations and the surface-judge execution details, is in the Additional Supplementary Material.

E.2 Limitations

Studied problem. We study observable, rule-governed state changes for which the relevant observations are available and a current-state answer is well defined. Open-ended memory, hidden world state, planning, and questions that require unavailable evidence are outside this scope.

Coverage. The generated suite covers several state structures, but the platform histories come from software issues and package registries. GitHub and GitLab are independent platforms, not independent application domains. Most matched-interface comparisons and step diagnostics use GPT-4o; a second model family replicates only PTR, and GPT-5 is tested on a ledger slice. Specified interfaces saturate the reopen cohorts, and three leaderboard models nearly solve the platform strata; transition failures were observed only with GPT-4o on the ledger extension outside v1.0.0. Repair was tested on one GPT-4o label task, where every held-out observation carries an update. No third-party memory system was evaluated under the matched protocol. Each arm has one captured API realization at provider-default reasoning, and the Gemini-2.5-Pro leaderboard route returns no model identity (Appendix B.6). The controlled grid varies a composite difficulty axis, so it does not estimate how reliability changes causally with the number of relevant updates. Ledger histories also leave the expense group implicit (Appendix B.7).

Analysis scope. Only the reopen comparisons against PTR and the held-out repair gate were prespecified; the remaining analyses are exploratory. The diagnostic cohorts use separate selections from the official split, with the overlap detailed in Table 11.

Rendered language. Replay verifies typed transitions and answers, not whether each rendered message is natural, unambiguous, and faithful to its source event. GitHub and GitLab histories follow recorded event order, npm histories follow recorded release times with deprecations appended in a constructed order, and all platform text is author-rendered. The surface evidence consists of a dual-model judge diagnostic whose history-level agreement is low, and no human audit has been carried out.

PTR parsing boundary. Replay is deterministic only after an update has been parsed. Open-vocabulary labels, ambiguous verbs, and numeric identifiers can produce a wrong operation that

replay cannot correct on its own; on the earlier GitHub label rows PTR is 4.1 points below model-maintained state. Targeted repairs help in specific cases, a generic verifier does not, and the pipeline uses stream-level post-processing, so it is not a strictly online filter.

Earlier evaluations. The earlier results use different interface implementations, reported model labels without endpoint records for most models, and in part summary-level records. Their model-state error counts vary across API realizations, so they characterize recurrence of the failure and do not estimate effect sizes for current interfaces. The complete earlier evaluations, including their negative results and provenance limits, are retained in the Additional Supplementary Material.

Shared implementation risk. Deterministic generation makes intended updates explicit, but gold and PTR can share an implementation mistake. Oracle gates, the finite specification audit, step-wise replay checks, independently recorded platform events, and the deviation log reduce this risk without removing it. Independent final-state checks cover GitHub and GitLab; npm and generated histories are checked for consistency with the shared specification.

Scope relative to the closest work. In-context-learning dialogue state tracking already asks a model for turn-level state changes that code merges into the state (Hu et al., 2022), and Text2Mem executes typed memory operations after validation (Wang et al., 2026); we do not claim the delta-and-merge mechanism. Parsing-the-Stream folds an agent's append-only event ledger into typed run state (Pakhomov & Nijkamp, 2026). MemoryAgentBench probes overwrite semantics through question answering (Hu et al., 2026), and the concurrent StateMemBench grades whether answers reflect current or superseded state (Fan et al., 2026). Two different papers are named MemTrace: the one of Long et al. (2026) probes typed knowledge points at checkpoints, and the one of Deng et al. (2026) converts memory-system pipelines into executable memory-evolution graphs, attributes 160 human-annotated failures of long-context, retrieval, Mem0, and EverMemOS pipelines to faulty extraction, update, retrieval, or response operations, and uses the attribution to improve the systems. It thus localizes errors inside particular memory systems, and we do not claim to be the first to localize memory errors. Our distinction is narrower: system-independent gold input trajectories with a typed intended update per message, deterministic replay, and fixed-parse counterfactual interfaces, on recorded platform event streams (GitHub and GitLab order; npm release times) and generated tasks, with a matched comparison that changes only the executor of each transition. Table 1 compares the design properties of the benchmarks, and Table 2 those of the state interfaces.