WHEN SQL CORRECTNESS IS NOT ENOUGH: VERIFYING CONTRACT-TO-DELIVERY CONFORMANCE IN DATA AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Data agents can produce executable and seemingly reasonable artifacts that nevertheless violate the requirements ultimately established with the user. We formulate this distinct failure mode, *silent delivery non-conformance*, as a weak-oracle verification problem in which both the binding contract and the relevant evidence must be reconstructed from evolving proposals, revisions, approvals, cancellations and tool observations. We propose CER-VERIFY (Contract–Evidence Reconciliation for Verification), a framework that decouples revision-aware contract reconstruction from terminal-state evidence collection and performs requirement-level reconciliation using semantic judgment plus deterministic checks for schema consistency, artifact completeness, and namespace correctness. Across 147 data-agent trajectories spanning nine failure families, CER-VERIFY achieves 121/147 correct verdicts, retains 55/57 conformant deliveries, and detects 66/90 non-conformant ones, compared with 93/147 correct verdicts for the strongest standard baseline and 101/147 for the strongest component-matched control. Structural experiments show that, when the verification target evolves during interaction, errors arise primarily before the final correctness decision: the verifier must first infer what remains binding and resolve what was actually delivered. Deterministic checks become reliable only after this inference. Our findings therefore suggest that verification for persistent-state data agents is fundamentally a state-reconstruction problem before it is a correctness-checking problem.

1 INTRODUCTION

Data agents are expanding beyond writing DQL to answer questions over existing data (Yu et al., 2018; Li et al., 2023) toward modifying persistent data state (Jin et al., 2025; Lei et al., 2025b). A data agent may negotiate a schema, produce or revise a DDL design after user feedback, materialize ETL and pipeline artifacts, and finally hand those artifacts to downstream systems. This changes the consequence of an agent error: a wrong answer may end with the interaction, whereas a wrong delivery can be successfully executed, persisted, and propagated. Yet the delivered state can silently diverge from what was agreed: an accepted schema is altered, a committed field type has changed, an artifact is written to the wrong namespace, or a superseded revision is restored (See Appendix A for an illustration). Crucially, none of these failures necessarily causes SQL execution to fail. The system can remain fully operational while persisting the wrong delivery.

We call this failure mode *silent delivery non-conformance*: a mismatch between the latest accepted contract and the terminal delivered state despite apparently successful execution. It is distinct from *query- or execution-level correctness*, which asks whether an artifact satisfies a current request or is executable, and from *solution quality*, which asks whether the design is desirable. The central question is instead whether the agent delivered the state it committed to deliver. For data agents that modify persistent systems, successful execution can therefore show that a state was materialized without showing that the right state was materialized.

This creates a verification problem that is easy to miss but consequential in agentic data workflows. Yet existing data-agent verification largely inherits the verification target of Text-to-SQL systems, particularly in DQL settings. Recent methods move well beyond syntax checking: they test semantic

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.

alignment between a current question and generated SQL, inspect query-generation plans, or make intermediate reasoning and execution traces verifiable (Tian & Zhang, 2026; Wang et al., 2026; Li et al., 2026; Arif & Singh, 2026). Data-agent benchmarks similarly evaluate against prepared questions, reference results, executable tests, or fixed goal states (Li et al., 2023; Zhang et al., 2024; Lei et al., 2025a;b; Ye et al., 2026). These approaches provide useful oracles for asking whether a query is semantically correct for a given request or whether execution reaches a predefined target. A key distinction in our setting is that the verification target is itself produced and evolves during the interaction. The verifier must determine which requirements among proposals, corrections, approvals, and cancellations are still binding, and then determine whether a potentially multi-artifact terminal state satisfies them. Thus, the object being verified is not a single SQL statement or the latest user message, but an interaction-derived *accepted contract*.

This shift creates a more challenging verification setting. First, the oracle is *weak and distributed*: there is no reference artifact or hand-written checker that directly specifies the expected delivery. Evidence about what *should* be delivered is scattered across instructions, agent proposals, user acceptances, and revisions, while evidence about what *was* delivered resides in artifacts, namespaces, and tool observations. A completion message is only a claim about state, not independent evidence of that state. Second, both the specification and its evidence are time dependent. A later accepted revision can supersede an earlier requirement, a cancellation can remove an obligation, and a previously valid observation can become stale after an artifact is overwritten, moved, or deleted. Verification therefore requires reasoning not only about semantics, but also about authority, revision, identity, and freshness.

Together, these properties make seemingly reasonable verification strategies unreliable. Providing a judge with better-structured context can reduce distraction, but does not by itself determine which revision became authoritative. Giving an agent access to schemas and SQL tools does not guarantee that it can reliably resolve the artifacts and evidence those tools should inspect. Conversely, deterministic checks are only as sound as the specification they receive: an exact checker applied to an incorrect or obsolete contract can convert an upstream interpretation error into a confident false rejection. Yet once the accepted contract and terminal evidence are correctly structured and aligned, many delivery requirements reduce to exact questions of set membership, schema completeness, or namespace identity. The challenge is therefore not simply to use stronger judges or more tools, but to establish a reliable contract–evidence interface before applying semantic or deterministic verification.

We address this challenge with CER-VERIFY (**Contract–Evidence Reconciliation for Verification**), a weak-oracle verification framework. CER-VERIFY explicitly separates two sources of authority. A revision-aware contract branch reconstructs the active requirements that remain binding after proposal, acceptance, supersession, cancellation, and reopening events. Independently, an evidence branch inventories terminal artifacts and authoritative tool observations while preserving artifact identity, location, and freshness. The two branches meet only at requirement-level reconciliation: semantic judgment handles requirements that require interpretation, while deterministic Schema, Artifact Completeness, and Namespace checks decide properties that admit exact evidence. When trusted evidence is insufficient, the verifier returns INCONCLUSIVE rather than manufacturing a success.

We evaluate CER-VERIFY on 147 data-agent trajectories spanning nine families of silent delivery non-conformance, against holistic, agentic, rubric-based, and deterministic verification baselines. We further use component-matched controls, cumulative ablations, multiple judge configurations, and case-level failure analysis to examine where verification errors arise and which parts of the framework address them. Across these evaluations, CER-VERIFY provides a significantly stronger balance between retaining conformant deliveries and detecting non-conformant ones.

We summarize our main contributions as follows:

1. We formulate *silent delivery non-conformance* as a distinct weak-oracle verification problem for data agents that modify persistent systems: whether the delivered state satisfies the latest binding requirements, rather than merely being executable or semantically plausible.
2. We introduce CER-VERIFY, which separates revision-aware contract reconstruction from terminal evidence acquisition and reconciles them at requirement granularity using semantic judgment together with deterministic checks.
3. We construct a 147-case evaluation corpus spanning nine failure families and use component-matched controls, cumulative ablations, multi-judge evaluation, and case-level failure-locus auditing to show why structured context, tool access, or exact checking alone are insufficient.

2 PROBLEM FORMULATION

2.1 CONTRACT-TO-DELIVERY VERIFICATION

Let $\tau = (r_1, \ldots, r_T)$ denote an interaction trajectory, where each event may contain a user instruction, agent proposal, acceptance, revision, cancellation, or tool observation. Let S_T denote the terminal environment state after the interaction.

The verification target is the set of requirements that remain binding at the end of the trajectory. We define the *active contract* as

$$\mathcal{C}^* = \mathrm{Active}(\tau) = \{r_i = (q_i, s_i, v_i)\}_{i=1}^{n}, \tag{1}$$

where q_i is a normalized requirement, s_i denotes its scope, and v_i specifies its latest accepted revision. $\mathrm{Active}(\cdot)$ resolves acceptance, supersession, cancellation, and reopening in interaction order. Thus, $\mathcal{C}^*$ is neither the complete transcript nor simply the latest user message.

The verifier does not directly observe all properties of S_T. Instead, it constructs terminal evidence

$$\mathcal{E}_T = \mathrm{Observe}(\tau, S_T; \mathcal{C}^*), \tag{2}$$

from delivered artifacts and authoritative tool observations that can be resolved to the terminal state. Because neither a reference artifact nor a pre-authored checker is assumed, both $\mathcal{C}^*$ and $\mathcal{E}_T$ must be recovered from the interaction and environment; We refer to this as a weak-oracle verification setting.

2.2 VERIFICATION OBJECTIVE

For each active requirement r_i, the verifier predicts

$$\hat{y}_i = g(r_i, \mathcal{E}_T) \in \{\textsc{Pass}, \textsc{Non-Conformant}, \textsc{Inconclusive}\}. \tag{3}$$

Here, PASS means that available evidence supports the requirement, NON-CONFORMANT means that grounded evidence contradicts it or establishes a required delivery as missing, and INCONCLUSIVE means that the available evidence is insufficient for either conclusion.

Let $\mathcal{M} \subseteq \mathcal{C}^*$ denote the mandatory requirements. The trajectory-level verdict is

$$\hat{Y} = \begin{cases} \textsc{Non-Conformant}, & \exists r_i \in \mathcal{M} : \hat{y}_i = \textsc{Non-Conformant}, \\ \textsc{Pass}, & \forall r_i \in \mathcal{M} : \hat{y}_i = \textsc{Pass}, \\ \textsc{Inconclusive}, & \text{otherwise}. \end{cases} \tag{4}$$

A *silent delivery non-conformance* occurs when the terminal delivery violates at least one active requirement while remaining valid under local execution checks:

$$\mathcal{E}_T \not\models \mathcal{C}^* \quad \text{while} \quad \mathrm{LocalExec}(\mathcal{E}_T) = \textsc{Pass}. \tag{5}$$

The verification problem is therefore to determine whether the terminal state conforms to the interaction-derived contract, rather than merely whether its artifacts execute successfully.

This formulation exposes three possible loci of verification error:

$$\tau \xrightarrow{\text{contract reconstruction}} \mathcal{C}^*, \qquad S_T \xrightarrow{\text{state observation}} \mathcal{E}_T, \qquad (\mathcal{C}^*, \mathcal{E}_T) \xrightarrow{\text{reconciliation}} \hat{Y}. \tag{6}$$

3 METHOD

3.1 FRAMEWORK OVERVIEW

Section 2 formulates contract-to-delivery verification as three operations: recovering the active contract, observing the delivered state, and reconciling the two at requirement granularity. Our framework instantiates these operations through two logically separated branches (Fig. 1). The *contract branch* determines what remains binding from the interaction history, whereas the *evidence branch* determines what has actually been delivered. An agent's completion claim may help identify candidate artifacts, but it cannot serve as evidence that its own claim is correct.

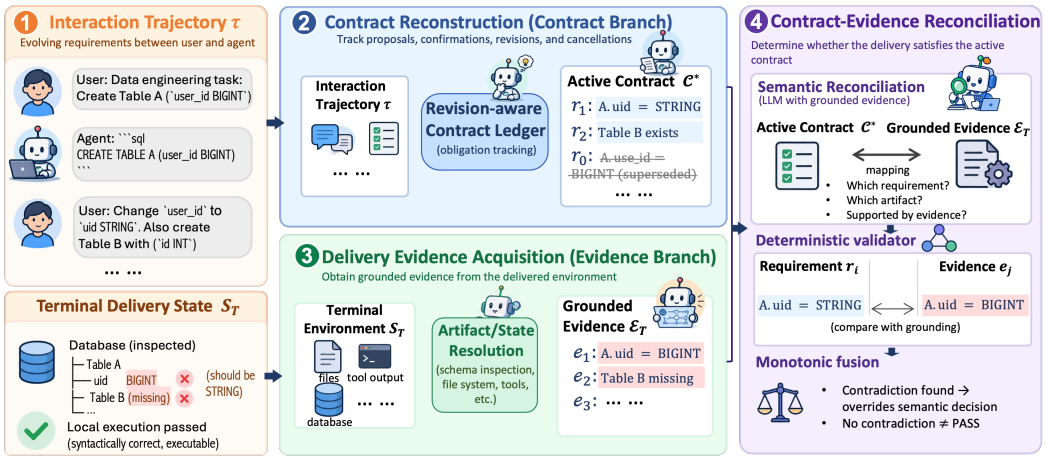


Figure 1: Overview of CER-VERIFY. (1) Given an interaction trajectory τ and the terminal delivery state S_T, CER-VERIFY (2) reconstructs the active contract C^* via a revision-aware ledger, (3) acquires grounded evidence $\mathcal{E}_T$ from the delivered environment, and (4) reconciles requirements with evidence using grounded semantic reasoning and deterministic validators. Monotonic fusion produces requirement-level decisions, which are aggregated into a final task-level verdict.

The framework is task-scoped and can operate repeatedly as an interaction evolves. At a verification boundary t, the verifier reconstructs or updates the active contract, resolves evidence against the current delivery state, and produces a task-level verdict, i.e., $(\tau_{\leq t}, S_t) \rightarrow \mathcal{C}_t \rightarrow \mathcal{E}_t \rightarrow \hat{Y}_t$. This decomposition is logical rather than implementation-specific: verification can proceed incrementally during interaction or be replayed from a complete task trajectory.

3.2 REVISION-AWARE CONTRACT LEDGER

The contract cannot be recovered by simply selecting the latest instruction, as requirements may be confirmed, revised, partially replaced, or withdrawn over time. We therefore maintain a revision-aware ledger, with $\mathcal{C}_t = \text{Update}(\mathcal{C}_{t-1}, r_t)$, that tracks decision provenance and status. At each verification boundary t, $\mathcal{C}_t$ retains only the requirements that remain binding.

Contract reconstruction follows three principles. First, an explicit user requirement may directly introduce a binding obligation. Second, an agent-proposed requirement becomes binding only when supported by subsequent user confirmation. Third, later explicit revisions supersede conflicting earlier decisions while leaving unaffected obligations active. Revision resolution therefore depends on interaction semantics rather than recency alone. Agent execution or completion statements do not modify the contract unless the interaction establishes a corresponding revision.

Each active decision retains a normalized requirement, its provenance in the interaction, and, where identifiable, the target artifact or property to which the requirement applies. This representation separates accepted obligations from transient proposals and provides structured anchors for subsequent evidence resolution. Further implementation details, including evidence reconstruction, deterministic validators, reconciliation, and representative verification cases are provided in Appendix C.

3.3 DELIVERY EVIDENCE ACQUISITION

The evidence branch independently reconstructs the state against which the active contract is evaluated. At verification boundary t, we construct $\mathcal{E}_t = \text{Observe}(\tau_{\leq t}, S_t, \mathcal{C}_t)$, where S_t denotes the current delivered state. This stage exposes observable properties of the delivery without yet deciding whether they satisfy the contract. Conceptually, an evidence item is represented as $e_j = (\text{source}, \text{locator}, \text{value}, \text{provenance}, \text{currency})$. The *locator* identifies the concrete delivery object and property being observed, such as an artifact, table, field, or delivery location. Evidence

may therefore describe artifact existence, identity, schema properties, cross-artifact relations, or content required for semantic comparison.

Evidence must satisfy three conditions before it can support a requirement. It must be *delivery-grounded*, rather than derived solely from the agent's own claim; it must provide a *resolvable locator* to the requirement's target; and it must correspond to the state being verified rather than only to an earlier artifact version. These conditions prevent an agent-authored summary or stale observation from serving as proof of terminal delivery.

Evidence acquisition combines deterministic processing with semantic resolution. Structured properties, such as artifact identities, SQL targets, and schemas, are extracted when available. When multiple candidates may represent the delivery, the verifier resolves the terminal artifacts using their observable properties, interaction provenance, and terminal-state information.

3.4 GROUNDED CONTRACT–EVIDENCE RECONCILIATION

Given the active contract and resolved delivery evidence, reconciliation evaluates each binding requirement independently before producing the task-level verdict. The semantic component compares a normalized requirement with the evidence associated with its target and determines whether the evidence supports fulfillment, contradicts the requirement, or remains insufficient.

Because verification operates under a weak oracle, semantic non-conformance is accepted only when it is grounded on both sides of the comparison: semantic contradiction $\Rightarrow$ contract-grounded $\wedge$ delivery-grounded. The contract side must trace to an active requirement, while the delivery side must resolve to observed evidence through a *valid locator*. The verifier also retains the expected and observed states underlying the comparison. A purported contradiction lacking either grounding is treated as insufficient evidence rather than as a confirmed failure.

This grounding is particularly important when several artifacts, revisions, or candidate interpretations coexist. Requirement–evidence resolution constrains semantic judgment to a concrete comparison target and allows the verifier to check whether a reported failure refers to both a binding requirement and an observed delivery. When the available information cannot support such a comparison, the verifier returns INCONCLUSIVE instead of forcing a binary decision.

A non-conformance result includes not only a verdict but also the violated requirement, expected and observed states, and the evidence *locator* supporting the comparison. This structured output makes failures actionable and enables subsequent verification to focus on the affected requirement.

3.5 DETERMINISTIC VALIDATORS AND MONOTONIC FUSION

Semantic reconciliation provides broad coverage, while some contract properties can be verified directly using deterministic tools. We therefore augment semantic judgment with deterministic validators. These validators are contradiction-oriented: they establish non-conformance only when their applicability conditions are satisfied and an explicit mismatch is observed. Insufficient deterministic evidence does not itself imply either conformance or non-conformance.

Our evaluated instantiation uses three complementary validators. The *Schema* validator compares explicitly constrained schema properties between the contract and the resolved delivery. The *Artifact Completeness* validator detects required targets that are absent from the observed delivery. Let $\mathcal{T}_{\text{exp}}$ and $\mathcal{T}_{\text{obs}}$ denote the expected and observed target sets, respectively; missing targets are $\mathcal{M} = \mathcal{T}_{\text{exp}} \setminus \mathcal{T}_{\text{obs}}$. A non-empty $\mathcal{M}$ constitutes evidence of omission only when both sets have been reliably resolved. Finally, the *Namespace* validator detects explicit conflicts between expected and observed qualified delivery identities. It reports a violation only when both identities are sufficiently specified, avoiding failures caused merely by incomplete location information.

Let $d_i^s \in \{\text{PASS}, \text{NON-CONFORMANT}, \text{INCONCLUSIVE}\}$ denote the grounded semantic decision for requirement r_i. For validator k, let $a_{ik} \in \{0, 1\}$ indicate applicability and $v_{ik} \in \{0, 1\}$ indicate whether the validator establishes a grounded violation. We combine the two branches as

$$d_i = \begin{cases} \text{NON-CONFORMANT}, & \exists k : \; a_{ik} = 1 \wedge v_{ik} = 1, \\ d_i^s, & \text{otherwise.} \end{cases} \tag{7}$$

The fusion is asymmetric by design. A grounded deterministic contradiction can overturn a semantic PASS or INCONCLUSIVE decision, whereas the absence of such a contradiction does not override a semantic failure or convert INCONCLUSIVE into PASS. Deterministic validators therefore strengthen the decision when directly verifiable evidence exists, without forcing a binary judgment on requirements that remain semantic or under-observed.

3.6 IMPLEMENTATION AND COMPUTATIONAL PROFILE

The framework is independent of verification frequency. During deployment, the active contract and delivery evidence can be updated and reconciled as the task evolves. In our evaluation, each trajectory and verification boundary are fixed, so we reconstruct the active contract up to that boundary, resolve delivery evidence against the terminal task state, and run the full verification pipeline once on the resulting snapshot. The snapshot contains artifact identities, relevant content, schema information, and other observable properties required for verification. It is shared by semantic reconciliation and applicable deterministic validators, ensuring a consistent view of the contract and delivery state. The same snapshot can be reused across runs, enabling controlled and reproducible comparison.

The computational overhead of the deterministic stages is small relative to semantic inference. With n active requirements, a resolved artifacts, and f parsed schema fields, artifact indexing is linear in a, completeness checking can be implemented in $O(n + a)$ using indexed target sets, and schema comparison is linear in the number of compared fields. Namespace validation is likewise linear in the number of explicit delivery locators. The dominant cost therefore remains contract reconstruction and semantic reconciliation; detailed judge token usage and latency are reported in Appendix G.4. Deterministic validation can be rerun cheaply whenever the verified task state changes.

4 EXPERIMENTS

Our experiments investigate a central scientific question: *when the verification target evolves through interaction, which latent task states must be recovered before correctness becomes reliably decidable, and how does their recovery alter the behavior of different verification strategies?* We address this by first comparing representative verification paradigms, then using structural controls to separate the roles of target reconstruction, evidence resolution, and deterministic validation, and finally examining robustness and the residual failure boundary.

4.1 EXPERIMENTAL SETUP

Evaluation corpus. We evaluate on 147 end-to-end data-agent trajectories with dialogue, accepted decisions, tool context, delivered artifacts, and a terminal completion state. The corpus contains 65 production trajectories (40 conformant, 25 non-conformant), 31 controlled counterfactual failures, and 51 context-stress variants (17 conformant, 34 non-conformant) derived from 47 known parent tasks. The cases span nine conformance-failure families, including artifact omission, schema and scope violations, semantic substitution, revision drift, and stale-evidence drift. Counterfactuals introduce contract-grounded violations whose terminal evidence is directly inspectable; context-stress variants add non-persistent requests or explanatory turns while preserving the Gold verdict. Appendix D details corpus construction, annotation, provenance, and the paired stress protocol.

Compared verification paradigms. We compare final-output judging (Zheng et al., 2023; Liu et al., 2023) (*B1*), holistic trajectory judging (Lù et al., 2025) (*B2*), an agentic judge with tool access (Zhuge et al., 2025) (*B3*), a generated-rubric verifier with LLM grading (Cook et al., 2024; Ding, 2026) (*B4*), and two component-matched controls. *B2 w/ canonical input* keeps the same one-shot holistic decision rule of *B2* but reorganizes the same observable information into indexed dialogue, candidate artifacts, schema candidates, and SQL inventories; it receives no active contract, requirement–evidence bindings, deterministic-validator outputs, or Gold labels. Appendix E.2 gives a side-by-side raw-versus-canonical example. The complementary control *B4 w/ deterministic validators* isolates the effect of deterministic checking. It retains *B4's* frozen rubric and LLM-based grading, while applying the same deterministic validators used by CER-VERIFY to rubric items that can be grounded to concrete artifact properties. Full method definitions, prompts, and interface differences are given in Appendix E.

Method	Verification input	Correct	Pass retain	NC recall	Abstain
B1: Final-output Judge	final request + final delivery	93/147	28/57	65/90	8/147
B2: Trajectory Judge	raw trajectory; holistic verdict	61/147	23/57	38/90	47/147
B3: Agent-as-a-Judge w/ tools	raw trajectory + agentic tools	48/147	14/57	34/90	76/147
B4: Generated Rubric w/ LLM	frozen inferred specification	89/147	35/57	54/90	34/147
B2 w/ canonical input	structured input; same holistic verdict	101/147	46/57	55/90	13/147
B4 w/ deterministic validators	B4 specification + our validators	97/147	26/57	**71/90**	22/147
CER-VERIFY (ours)	active contract + grounded evidence + validators	**121/147**	**55/57**	66/90	**9/147**

Table 1: Primary verdict results on the reporting 147-case corpus (57 conformant, 90 non-conformant).

Metrics and protocol. We emphasize four verdict metrics: *Correct*, *Pass retention* on conformant deliveries, *non-conformance (NC) recall* on failed deliveries, and *Abstention*. An *Abstention* is scored as incorrect but is distinct from an explicit false acceptance. NC escape, coverage/selective-risk statistics, clustered bootstrap intervals, paired tests, and cost accounting are reported in Appendix F. Unless otherwise stated, the main comparison uses DeepSeek V4 Flash (DeepSeek-AI, 2026) with reasoning disabled; Gold labels are joined only after predictions are persisted.

4.2 Existing Verifiers Fail at Different Stages

Table 1 shows that expanding the judge's observation scope or tool access does not by itself solve the verification problem. The final-output judge detects 65/90 non-conformant deliveries but retains only 28/57 conformant ones, while the holistic trajectory judge is correct on only 61/147 cases. Reorganizing the same observable information into canonical input raises the holistic judge to 101/147 correct decisions. Because its decision rule remains holistic and receives no recovered contract or evidence binding, this improvement primarily reflects the benefit of structured interaction representation rather than contract–evidence reconciliation.

Tool access fails at a different stage. The agentic judge with tools is correct on 48/147 cases and abstains on 76/147; the audit attributes 62 abstentions to artifact resolution. It can inspect state, yet often cannot determine which terminal artifact should serve as evidence for the binding requirement. Conversely, attaching deterministic validators to B4's generated specification increases NC recall from 54/90 to 71/90 but lowers pass retention from 35/57 to 26/57. The same validators can therefore become overly aggressive when the inferred specification is wrong.

CER-VERIFY reaches 121/147 correct verdicts, retains 55/57 conformant deliveries, detects 66/90 non-conformant deliveries, and abstains nine times. The point is not that one operating point dominates every individual metric; B4 with deterministic validators blocks more failures. Rather, the comparison shows three distinct upstream failure loci—interaction representation, target/state resolution, and specification quality—before the final correctness decision. To additionally capture the operational impact of incorrectly blocking conformant cases, Appendix F.1 introduces a Conformance Score that jointly accounts for pass retention and failure detection. This complementary view further highlights the advantage of CER-VERIFY in balancing failure detection against unnecessary blocking.

4.3 Verification Requires Target and State Inference

We next isolate the intermediate inference steps that precede the final correctness decision. When the verification target evolves through interaction, the verifier must first recover which requirements remain binding and then resolve the terminal evidence for those requirements. Delivery conformance can only be assessed after these target- and state-resolution steps.

The first transition isolates the effect of target reconstruction. The *latest-plan target* uses the latest saved plan candidate and final user message, whereas the *revision-aware* variant reconstructs active

requirements from the full interaction. Correct verdicts increase from 84 to 91, but the more informative change is pass retention, which rises from 41/57 to 50/57 while NC recall does not improve. Recovering the active contract therefore primarily prevents valid deliveries from being judged against stale, superseded, or otherwise incorrect requirements.

Explicit *requirement–evidence binding* then raises pass retention to 55/57, while abstention increases from 3 to 20 and NC recall falls from 41/90 to 24/90. This is not a uniform accuracy gain. Rather, explicit requirement–evidence binding changes the verifier's decision strategy by constraining semantic judgments to requirement-specific evidence. Without deterministic validation, this more evidence-conditioned strategy is also more conservative: pass retention improves, but many non-conformant cases do not consistently yield an explicit failure verdict. In the weak-oracle setting, target recovery and state resolution therefore shift verification from unconstrained holistic judging toward evidence-grounded comparison, while deterministic validation remains necessary for reliably resolving grounded discrepancies.

Structure	Correct	P-ret.	NC-rec.	Abst.
Latest-plan target	84	41/57	43/90	11/147
Revision-aware target	91	50/57	41/90	**3/147**
+ Evidence binding	79	**55/57**	24/90	20/147
+ Deterministic validators (CER-VERIFY)	**121**	**55/57**	**66/90**	9/147

Table 2: Structural decomposition on the same 147 cases. The first two transitions progressively recover the active target and bind terminal evidence. P-ret., NC-rec., and Abst. denote pass retention, non-conformance recall, and abstention.

These transitions support treating target and state recovery as part of verification rather than preprocessing. Appendix F reports structural decomposition transition matrix, per-cohort results, and paired verdict analyses using exact two-sided McNemar tests, which characterize the direction and consistency of correctness changes between verifier variants.

4.4 Robustness and Remaining Failure Boundary

Robustness to context and verifier variation. The 51 context-stress variants add non-persistent requests or explanatory turns while preserving the delivery contract and Gold verdict. Table 3 therefore tests whether verification remains stable when the interaction changes without changing what was delivered or what should be verified. For *B2 w/ canonical input*, the perturbation remains in the dialogue view, while the artifact-side representation is held fixed. The contrast with raw trajectory judging shows that canonicalization largely restores invariance by separating dialogue variation from a fixed artifact-side representation. CER-VERIFY preserves similar invariance at a stronger overall verification level. This suggests that structured representation primarily stabilizes the verification problem, while contract–evidence reconciliation improves the quality of the resulting decision.

The `avg@3` and multi-judge configuration studies probe two complementary sources of verifier variation. Under a fixed judge configuration, CER-VERIFY remains stable across independent runs, indicating limited sensitivity to run-level sampling noise. Its qualitative advantage also persists across different judge configurations, although the trade-off between pass retention and non-conformance detection varies across models. Because each configuration reruns all semantic inference stages, these differences may arise not only from the final judgment but also from how the active requirements and their supporting evidence are reconstructed and interpreted. The results therefore reveal a configuration-specific semantic inference tendency within an otherwise shared verification architecture. Appendix G reports the complete context-stress, multi-configuration, and `avg@3` analyses.

Method	Correct: Parent → Stress	Consistency ↑
B2	35 → 15 / 51	23/51
B2 w/ canonical input	38 → 38 / 51	**45/51**
B4	36 → 32 / 51	31/51
CER-VERIFY	**48 → 43 / 51**	44/51

Table 3: Context-stress invariance over 51 parent–variant pairs. Stress variants add irrelevant context without changing the Gold verdict. Consistency counts pairs receiving the same predicted verdict.

Remaining failure boundary. The residual audit shows that most remaining errors lie in the intermediate structured representations rather than the decomposition itself. Schema failures mainly stem from incomplete projection of recovered requirements into concrete checks, while artifact-side failures arise from uncertainty in resolving the authoritative terminal artifact or interpreting its absence. These limitations are amplified in realistic agent settings, where verification must operate

over a combinatorial space of task types, dialogue histories, revisions, tool paths, artifact forms, and execution timing. The resulting interactions create long-tail states that are difficult to enumerate exhaustively and expose coverage gaps in requirement projection and terminal-state resolution. We therefore view the remaining failure boundary primarily as a coverage problem in these structured recovery modules, rather than a limitation of the decomposition itself. Further improvements should focus on making these intermediate representations more faithful and extensible, rather than simply strengthening the final LLM judgment. Appendix H provides the full audit protocol, failure-locus breakdown, and representative case-level rationales.

5 LIMITATIONS

The 147-case corpus spans a broad range of SQL creation and editing scenarios, covering common DDL and ETL workflows. Nevertheless, the space of data-engineering tasks is substantially broader, and some workflow patterns are not represented in the current evaluation. The current deterministic validators also cover a deliberately narrow evidence set and may miss violations that leave these signals unchanged, such as dependency changes with the same output schema. The earlier structural variants differ not only in the component being introduced, but also in how information is presented to the verifier. They should therefore be interpreted as comparisons between verification structures rather than strict one-component ablations. In contrast, the deterministic-validation transition reuses the same semantic judgments and therefore provides a cleaner controlled comparison. Broader task shifts, richer evidence types, and longer interaction histories remain open evaluation settings.

6 RELATED WORK

Data-agent evaluation has expanded from query generation to realistic database through benchmarks such as BIRD, Spider 2.0, DAComp and Squirrel (Li et al., 2023; Lei et al., 2025a;b; Ye et al., 2026). Correspondingly, recent Text-to-SQL verification methods move beyond syntactic validity: PV-SQL derives checks from the user question and repairs generated SQL (Tian & Zhang, 2026), STEF aligns semantic specifications extracted from questions and SQL (Arif & Singh, 2026), VET makes intermediate query reasoning executable (Wang et al., 2026), and G^2SQL verifies both generation plans and resulting SQL (Li et al., 2026). Broader agent evaluation further considers interactive trajectories and terminal environment states: AgentBench evaluates interactive reasoning and action (Liu et al., 2024); SWE-bench, WebArena, OSWorld, and τ-bench use executable tests, task-specific evaluators, or annotated states (Jimenez et al., 2024; Zhou et al., 2024; Xie et al., 2024; Yao et al., 2025); and related verification work checks completion claims against programmatic state (Advani, 2026), inspects processes and environments (Zhuge et al., 2025), or acquires runtime evidence through active probes (Long et al., 2026). Multi-turn SQL systems use dialogue memory, while state-verification methods compare execution results against predefined target states (Guo et al., 2026). While these approaches increasingly strengthen query-level correctness or terminal-state verification, they generally assume that the request or intended state is already specified. In our setting, both the verification target and its supporting evidence evolve during interaction. Verification must therefore recover the latest authoritative contract and reconcile it with the corresponding terminal evidence. A more detailed review of related work can be found in Appendix B.

7 CONCLUSION

For data agents that modify persistent systems, verification becomes difficult when the target itself evolves through interaction. We formulate *silent delivery non-conformance* as a weak-oracle setting in which an executable final state may diverge from what was ultimately accepted, requiring both the binding contract and relevant terminal evidence to be reconstructed before correctness can be assessed. Our experiments show that common verification approaches fail at multiple stages before the final judgment, including interaction representation, recovery of binding requirements, and resolution of terminal evidence. These results support treating target and state recovery as part of verification itself, rather than as preprocessing to a final correctness decision. They also explain why deterministic checks can be unreliable when applied to a mis-specified target or unresolved evidence. Once these upstream uncertainties are reduced, residual errors concentrate in requirement projection and artifact resolution rather than generic semantic judgment.

8 Disclosure of Generative AI Use in Writing

Generative AI tools were used only for language editing, including improving grammar, clarity, and readability of the manuscript. These tools were not used to generate scientific claims, conduct analyses, or determine the conclusions of this work. All AI-assisted edits were reviewed and approved by the authors, who take full responsibility for the final content of the manuscript.

References

Laksh Advani. From confident closing to silent failure: Characterizing false success in LLM agents. *arXiv preprint arXiv:2606.09863*, 2026. URL https://arxiv.org/abs/2606.09863.

Taslim Jamal Arif and Kuldeep Singh. Agent-agnostic evaluation of SQL accuracy in production text-to-SQL systems. *arXiv preprint arXiv:2604.28049*, 2026. doi: 10.48550/arXiv.2604.28049. URL https://arxiv.org/abs/2604.28049.

Jonathan Cook, Tim Rocktäschel, Jakob Foerster, Dennis Aumiller, and Alex Wang. TICKing All the Boxes: Generated checklists improve LLM evaluation and generation. In *NeurIPS 2024 Workshop on Language Gamification*, 2024. URL https://arxiv.org/abs/2410.03608.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

Liang Ding. AdaRubric: Task-adaptive rubrics for reliable LLM agent evaluation. *arXiv preprint arXiv:2603.21362*, 2026. URL https://arxiv.org/abs/2603.21362.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2305.11738.

Taicheng Guo, Hai Wang, Chaochun Liu, Mohsen Golalikhani, Xin Chen, Xiangliang Zhang, and Chandan K. Reddy. MTSQL-R1: Towards long-horizon multi-turn text-to-SQL via agentic training. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 33905–33938, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1563. URL https://aclanthology.org/2026.acl-long.1563/.

Yuxin Jiang, Yufei Wang, Xingshan Zeng, Wanjun Zhong, Liangyou Li, Fei Mi, Lifeng Shang, Xin Jiang, Qun Liu, and Wei Wang. FollowBench: A multi-level fine-grained constraints following benchmark for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://arxiv.org/abs/2310.20410.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.06770.

Tengjun Jin, Yuxuan Zhu, and Daniel Kang. ELT-bench: An end-to-end benchmark for evaluating AI agents on ELT pipelines. *arXiv preprint arXiv:2504.04808*, 2025. URL https://arxiv.org/abs/2504.04808.

Yuta Koreeda and Christopher D. Manning. ContractNLI: A dataset for document-level natural language inference for contracts. In *Findings of the Association for Computational Linguistics: EMNLP*, 2021. URL https://arxiv.org/abs/2110.01799.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, et al. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows. In *International Conference on Learning Representations*, 2025a. URL https://arxiv.org/abs/2411.07763.

Fangyu Lei, Jinxiang Meng, Yiming Huang, Junjie Zhao, Yitong Zhang, Jianwen Luo, Xin Zou, Ruiyi Yang, Wenbo Shi, Yan Gao, Shizhu He, Zuo Wang, Qian Liu, Yang Wang, Ke Wang, Jun Zhao, and Kang Liu. DAComp: Benchmarking data agents across the full data intelligence lifecycle, December 2025b. URL https://arxiv.org/abs/2512.04324.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. Can LLM already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36:42330–42357, 2023. URL https://arxiv.org/abs/2305.03111.

Jinyang Li, Nan Huo, Yan Gao, Jiayi Shi, Yingxiu Zhao, Ge Qu, Yurong Wu, Chenhao Ma, Jian-Guang Lou, and Reynold Cheng. Tapilot-Crossing: Benchmarking and evolving LLMs towards interactive data analysis agents. *arXiv preprint arXiv:2403.05307*, 2024. URL https://arxiv.org/abs/2403.05307.

Xiang Li, Jinguo You, Heng Li, Jun Peng, Xi Chen, Ziheng Guo, Kun Li, and Tianyi Xu. G²SQL: Guided & guarded Text-to-SQL generation with two-stage verification. 311:131276, 2026. ISSN 0957-4174. doi: 10.1016/j.eswa.2026.131276. URL https://www.sciencedirect.com/science/article/pii/S0957417426001892.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2308.03688.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-eval: NLG evaluation using GPT-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 2511–2522, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.153. URL https://aclanthology.org/2023.emnlp-main.153/.

Minghui Long, Yanjie Zhao, and Haoyu Wang. LogicHunter: Testing LLM agent frameworks with an agentic oracle. *arXiv preprint arXiv:2607.06195*, 2026. URL https://arxiv.org/abs/2607.06195.

Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stańczak, Peter Shaw, Christopher J. Pal, and Siva Reddy. AgentRewardBench: Evaluating automatic evaluations of web agent trajectories. In *Conference on Language Modeling*, 2025. URL https://arxiv.org/abs/2504.08942.

Shanxiang Mao, Muyang Liu, Junru Chen, Dongliang Nan, and Jian Ma. A trustworthy intent-driven LLM-agent workflow with state verification for power system simulation. *SSRN Electronic Journal*, 2026. doi: 10.2139/ssrn.7074124. URL https://dx.doi.org/10.2139/ssrn.7074124.

Yuan Tian and Tianyi Zhang. PV-SQL: Synergizing database probing and rule-based verification for text-to-SQL agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 25827–25845, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1286. URL https://aclanthology.org/2026.findings-acl.1286/.

Dongyu Wang, Jingyu Li, Lan Zhang, Ganggang Yu, and Liang Huang. VET: Verifiable execution tracing for reliable text-to-SQL generation. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 30867–30883, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1544. URL https://aclanthology.org/2026.findings-acl.1544/.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2404.07972.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2406.12045.

Jing Ye, Yiwen Duan, Yonghong Yu, Victor Ma, Yang Gao, and Xing Chen. Beyond Text-to-SQL: Can LLMs really debug enterprise ETL SQL?, January 2026. URL https://arxiv.org/abs/2601.18119.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3911–3921. Association for Computational Linguistics, 2018. doi: 10.18653/v1/D18-1425. URL https://aclanthology.org/D18-1425/.

Yuge Zhang, Qiyang Jiang, XingyuHan XingyuHan, Nan Chen, Yuqing Yang, and Kan Ren. Benchmarking data science agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5677–5700, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.308. URL https://aclanthology.org/2024.acl-long.308/.

Chris Zheng and Geng Yang. CONTINUITY: Security-context contracts for composable LLM agent controls. *arXiv preprint arXiv:2609.05269*, 2026. doi: 10.48550/arXiv.2609.05269. URL https://arxiv.org/abs/2609.05269.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://arxiv.org/abs/2306.05685.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023. URL https://arxiv.org/abs/2311.07911.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2307.13854.

Mingchen Zhuge, Changsheng Zhao, Dylan R. Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. Agent-as-a-judge: Evaluate agents with agents. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 80569–80611. PMLR, 2025. URL https://proceedings.mlr.press/v267/zhuge25a.html.

A Silent Delivery Non-Conformance: A Schema-Drift Example

Figure 2 illustrates silent delivery non-conformance through a schema-drift example. The interaction establishes a revised schema as the active contract, yet the final DDL silently changes a field type while remaining executable. Local execution therefore passes, but the delivered state does not conform to the latest accepted contract.

B Related Work

Text-to-SQL and data-agent verification. Data-agent benchmarks cover multi-step data science, enterprise SQL, and interactive analysis (Zhang et al., 2024; Lei et al., 2025a; Li et al., 2024). Recent verifiers move beyond parsing. PV-SQL extracts checks from the current question and repairs a generated query (Tian & Zhang, 2026); STEF aligns semantic specifications extracted from a question and its SQL without reference SQL (Arif & Singh, 2026). VET makes intermediate query reasoning executable (Wang et al., 2026), while G^2SQL checks an internal SQL-generation plan and then reviews the generated SQL (Li et al., 2026). These methods address query verification, semantic alignment, or execution traces. Their target is a current question or generation process. Our target is the final delivery under the latest contract accepted through proposal, revision, and cancellation.

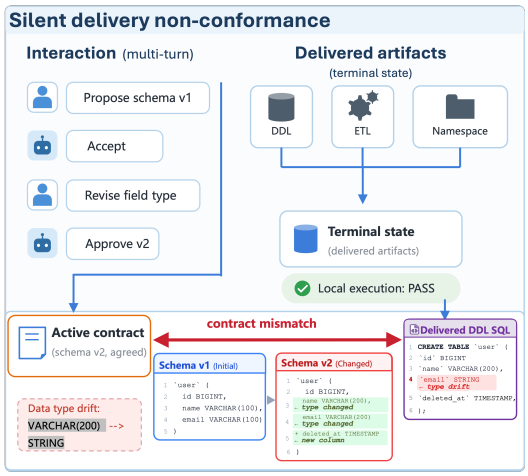


Figure 2: **Silent delivery non-conformance.** An executable delivery can still violate the latest accepted contract through column changes, type drift, or stale revisions.

Long-horizon and terminal-state verification. Interactive-agent benchmarks evaluate reasoning and action over long trajectories (Liu et al., 2024). SWE-bench, WebArena, OSWorld, and τ-bench use tests, task-specific evaluators, or annotated final states (Jimenez et al., 2024; Zhou et al., 2024; Xie et al., 2024; Yao et al., 2025). MTSQL-R1 adds database execution and persistent dialogue memory for multi-turn query coherence (Guo et al., 2026); dialogue memory, however, does not by itself mark which revision is binding. A power-system workflow verifies execution evidence against required state conditions, but those conditions belong to a fixed intended simulation task (Mao et al., 2026). False-success work compares completion claims with programmatic state (Advani, 2026); Agent-as-a-Judge inspects intermediate processes and environments (Zhuge et al., 2025); LogicHunter uses documentation, source, and runtime probes as an active oracle for framework bugs (Long et al., 2026). We share their use of state evidence but must first recover the authoritative, interaction-derived contract.

End-to-end constraints and contracts. LLM judges scale open-ended evaluation but exhibit presentation and self-enhancement biases (Zheng et al., 2023). TICKing generates instruction-specific checklists before grading (Cook et al., 2024); a wrong or incomplete checklist can still determine the final score. IFEval and FollowBench show the value of exact checks for explicit constraints (Zhou et al., 2023; Jiang et al., 2024), while ReAct and CRITIC use tools and external feedback during reasoning (Yao et al., 2023; Gou et al., 2024). ContractNLI grounds contract entailment in document spans (Koreeda & Manning, 2021). CONTINUITY carries security-authorization contracts across components so that external effects have current authorization witnesses (Zheng & Yang, 2026). Our lifecycle is structurally similar, but the protected property is delivery semantics rather than authorization. The contract is also not declared in advance: it is the active set of user-accepted requirements recovered under a weak oracle and checked against terminal artifacts.

C METHOD AND IMPLEMENTATION DETAILS

This appendix expands the three-stage decomposition used in the main paper: active-contract reconstruction, terminal-state observation, and contract–evidence reconciliation.

C.1 REVISION-AWARE CONTRACT RECONSTRUCTION

At each verification boundary, the visible interaction up to that boundary is first converted into a canonical indexed representation that preserves message order, speaker, and the provenance of proposals, confirmations, revisions, and cancellations. CER-VERIFY then reconstructs a decision ledger from this representation, decomposing accepted decisions into requirement-level entries and linking each entry to the interaction event from which it was inferred. This canonicalization separates the temporal structure of the interaction from the subsequent semantic reconstruction, rather than asking the judge to reason directly over an unstructured transcript.

Requirements are resolved in interaction order through an append-only sequence of accepted revisions. Each revision records its parent, acceptance source, active requirement IDs, and lifecycle change (Table 4). A later explicit revision supersedes only the conflicting scope of an earlier accepted requirement, while unaffected obligations remain active. Cancellation deactivates the affected requirement, and a later reopening creates a new active transition without erasing the prior history. Agent restatements do not create revisions, and approval to execute does not by itself establish a different delivery contract. The resulting active projection therefore represents the latest accepted state of each requirement while preserving provenance to earlier revisions. This distinction matters because recency alone is not an oracle for intent: the latest user message may modify only part of an earlier plan, and the latest agent proposal may never have been accepted. Downstream reconciliation therefore operates on the active projection of the reconstructed ledger, rather than on the complete transcript or simply the most recent message.

Record	Required content
Revision	Stable ID, parent ID, accepted source locator, change type, and active requirement IDs.
Requirement	Stable ID, lifecycle status, kind, expected object, provenance locator, and verification method.
Artifact	Canonical identity, terminal locator, content hash, schema projection, and validity status.
Assertion	One active requirement compiled into an expected object, a checkable predicate, and evidence hints.

Table 4: Minimal records used to reconstruct the latest accepted contract.

For example, an accepted revision may replace requirement `R002` with `R003`. The active contract then contains `R001,R003`; evidence satisfying the superseded `R002` cannot compensate for a violation of the active `R003`. Cancelled and superseded requirements are retained in the ledger for provenance but are excluded from required coverage.

In the released annotation schema, each requirement is additionally assigned a semantic kind—*entity*, *attribute*, *relation*, *constraint*, *scope*, *action*, *delivery*, or *natural language*—and a verification route: *deterministic*, *clean-reviewer*, *human-only*, or *unsupported*. These annotations specify both what type of obligation a requirement represents and how its satisfaction can be established during verification.

C.2 EVIDENCE ACQUISITION AND BINDING

The evidence branch independently reconstructs observable properties of the terminal delivery. Evidence items retain a source, locator, observed value, provenance, and currency. A requirement can be supported or contradicted only when the evidence is delivery-grounded, resolves to the requirement's target, and refers to the state being verified. Agent-authored completion summaries may identify candidate artifacts but are not treated as evidence that their own claims are correct.

Contract activity and evidence freshness are treated separately. A requirement remains active until it is superseded or cancelled, whereas an evidence item is current only if it refers to the terminal artifact

version used at the verification boundary. When multiple artifacts or versions coexist, deterministic preprocessing extracts candidate identities, SQL targets, and schema properties before semantic resolution. Explicit requirement-to-evidence binding narrows each comparison to a concrete target. If the target or observed state cannot be resolved with sufficient confidence, the semantic branch returns INCONCLUSIVE rather than forcing a binary decision.

These resolved bindings are materialized as an evidence ledger, in which each row links one active requirement to the terminal evidence used to assess it. Each row receives one of three statuses: SATISFIED, VIOLATED, or UNRESOLVED. To preserve auditability, the ledger retains the evidence source, locator, observed value, and provenance metadata rather than copying an agent-authored completion claim into the evidence field. Table 5 shows a compact example from a released case, where three active schema requirements are contradicted by the terminal DDL.

Assertion	Evidence	Source	Verifier	Status
Target has `comment_review_count`	Terminal DDL instead has `commented_review_count`	artifact A001	Schema	VIOLATED
Target has `comment_review_rate`	Terminal DDL instead has `commented_review_rate`	artifact A001	Schema	VIOLATED
Target has `nps_score`	Terminal DDL instead has `nps`	artifact A001	Schema	VIOLATED

Table 5: Example evidence ledger. Each row binds an active requirement to terminal delivery evidence and records the resulting verification status.

C.3 DETERMINISTIC VALIDATION, SEMANTIC FALLBACK, AND MONOTONIC FUSION

The evaluated implementation uses three contradiction-oriented deterministic validators. The *Schema* validator compares explicitly constrained schema properties against the resolved terminal DDL. The *Artifact Completeness* validator checks whether the explicitly required target set is contained in the terminal artifact inventory. The *Namespace* validator detects conflicts between sufficiently specified expected and observed qualified identities. Assertions that cannot be resolved by an eligible deterministic validator are routed to semantic review. Table 6 summarizes the routing conditions and decision rules.

Route	Eligible evidence	Decision rule
Schema	Active schema requirements and resolved terminal DDL	Report a contradiction when an explicitly constrained schema property is violated, including applicable field, type, order, or partition constraints.
Completeness	Explicit active target set and terminal artifact inventory	A nonempty expected-minus-observed target set constitutes a violation.
Namespace	Expected qualified identity and sufficiently resolved observed locator	An explicit identity conflict constitutes a violation; insufficient qualification yields no deterministic verdict.
Semantic review	Relevant terminal evidence not covered by an eligible deterministic validator	Assess entailment, contradiction, or uncertainty against the active requirement, with supporting evidence locators.

Table 6: Evidence is routed according to the active requirement and available terminal evidence, independent of the case label.

A grounded deterministic contradiction may overturn a semantic PASS or INCONCLUSIVE decision. The absence of a deterministic contradiction does not convert INCONCLUSIVE to PASS and does not override a semantic failure. This asymmetric fusion is intentionally conservative: deterministic tools strengthen the decision only on properties for which both applicability and contradiction can be established.

C.4 Representative trajectories and evidence traces

The traces below retain only events needed to identify the accepted revision, terminal delivery, evidence, and verdict. They are structured traces, not reconstructed chat logs.

Example-01: supported conformant delivery

USER Restricts delivery to workflow `wf_dws_study_1d`.
AGENT / PROPOSAL Two-task DAG: knowledge statistics then chapter statistics; daily at 06:00.
USER / ACCEPTANCE Confirms this proposal. **[ACCEPTED]**
AGENT / DELIVERY Reports the active workflow with the same two tasks and schedule.
ARTIFACT Terminal inventory contains both corresponding SQL artifacts.
VERIFIER Active requirements have addressable terminal support and no grounded conflict.
VERDICT ACCEPT (PASS).

Example-02: silent delivery non-conformance

AGENT / PROPOSAL Defines target fields including `comment_review_count`, `comment_review_rate`, and `nps_score`.
USER / ACCEPTANCE Approves the proposal and then authorizes SQL implementation. **[ACCEPTED]**
AGENT / DELIVERY Reports completion without disclosing a contract change.
ARTIFACT Terminal DDL instead contains `commented_review_count`, `commented_review_rate`, and `nps`.
VERIFIER The Schema tool records three missing and three unexpected fields.
VERDICT REJECT (NON-CONFORMANT).

Example-03: insufficient terminal evidence

USER / REQUEST Asks to repair a specific workspace SQL artifact.
AGENT / PROPOSAL Proposes a cast and partition-pruning repair.
USER / ACCEPTANCE Confirms the repair. **[ACCEPTED]**
AGENT / DELIVERY Claims the SQL, published version, and workflow were updated.
ARTIFACT The verifier-visible terminal bundle contains a comparison-plan document, but not the claimed modified SQL.
VERIFIER The delivery claim identifies what should be checked but cannot prove the terminal SQL state.
VERDICT ABSTAIN (INCONCLUSIVE).

C.5 Reproducibility and provenance

The unified results are assembled from frozen per-case predictions; Gold labels and reason annotations are joined only after predictions have been persisted. The structural study uses the same 147-case roster as the primary evaluation (Table 2). The *Latest-plan target* and *Revision-aware target* are newly sampled structural variants; CER-VERIFY reuses its frozen checkpoint. The structural result bundle records experiment protocol, run/checkpoint locations, and aggregation script.

D Data Construction and Evaluation Protocol

D.1 Failure taxonomy

We organize non-conformance cases into operational failure families based on how the delivered state diverges from the accepted contract. As summarized in Table 7, the taxonomy spans local artifact mismatches, cross-artifact inconsistencies, scope errors, and failures caused by revision or evidence drift.

Family	Operational definition
Omission	A required artifact, field, or delivery component is absent.
Mutation	A delivered name, type, or value differs from the accepted one.
Constraint	A key, nullability, uniqueness, or other explicit constraint is violated.
Partition	Partition fields or strategies differ from the accepted plan.
Substitution	A plausible but semantically different artifact is delivered.
Scope	The artifact exists under the wrong catalog, database, path, or task scope.
Cross-artifact	Individually plausible artifacts disagree or break a declared dependency.
Revision drift	The delivery follows a superseded rather than the latest accepted revision.
Evidence drift	A completion claim relies on evidence invalidated by a later state change.

Table 7: Failure families represented by the evaluation suite.

D.2 Corpus construction and composition

The reporting corpus contains 147 trajectories with 57 conformant and 90 non-conformant deliveries. It is the disjoint union of 65 production trajectories (40/25 Pass/Non-Conformant), 31 controlled counterfactual failures, and 51 context-stress variants (17/34 Pass/Non-Conformant). The 51 stress variants are derived from 47 unique parents; four parents have both a standard and a longer stress variant.

The 65 production trajectories are collected from authentic user–agent interactions in an enterprise-facing data-agent system deployed for data-engineering tasks. Users may interact with agents backed by different LLM families, including GLM, DeepSeek, GPT, and HY-family models, so the collected trajectories reflect heterogeneous model behaviors rather than a single underlying agent model. We restrict the corpus to tasks that can modify persistent data state or produce artifacts intended for persistent downstream use. During filtering, we exclude non-persistent interactions that do not result in such state changes or deliverables, including database exploration, metadata inspection, requests for clarification or explanation of domain knowledge, and discussions of business metrics without a persistent data-engineering delivery. This filtering aligns the evaluation corpus with the work's verification target: whether the terminal delivered state conforms to the requirements accepted during the interaction.

D.3 Controlled counterfactuals

The counterfactual cohort is constructed from Gold-Passed source trajectories by introducing controlled delivery-level violations guided by the nine-category failure taxonomy defined in Table 7. For each source trajectory, candidate mutations are instantiated only from failure categories that are applicable to its accepted contract and terminal artifact; we do not force every trajectory to support every failure type. A mutation is admitted only when the violated requirement is explicit in the accepted contract, the expected failure is unambiguous, and the resulting violation is directly observable in the terminal delivery evidence. We further exclude mutations whose interpretation depends on latent user intent or ambiguous semantic assumptions.

Two domain experts independently inspect each mutated trajectory to verify that the intervention correctly instantiates the intended failure category and that the resulting delivery violates at least one mandatory requirement. Each admitted counterfactual is therefore assigned a Gold-Failed label. This taxonomy-guided construction provides systematic coverage of representative delivery failures while preserving a clear causal link between the injected mutation and the expected verdict. Because the cohort contains only induced failures, however, it is designed to measure verifier sensitivity to known violations and does not by itself estimate pass retention or the class distribution encountered in deployment.

D.4 Context-stress variants

Context-stress variants are constructed to test whether verifier decisions remain stable under additional interaction that is irrelevant to the terminal delivery contract. Starting from an existing parent trajectory, we introduce non-persistent requests, explanatory exchanges, or other contextual turns that increase interaction complexity without modifying the accepted delivery requirements or the

terminal artifact. The added context therefore should not alter the delivery-level Gold verdict. All 51 parent–variant pairs satisfy $Y_{\text{parent}} = Y_{\text{variant}}$.

We evaluate each parent–variant pair by whether the verifier preserves its decision under added irrelevant context. Because the Gold verdict is unchanged, a robust verifier should behave consistently across the pair. We additionally distinguish *Harmful flips*, where a correct parent prediction becomes incorrect on the variant, from *Beneficial flips*, where an incorrect parent prediction becomes correct. Both indicate sensitivity to context that should be irrelevant to the verification target, while harmful flips directly capture degradation caused by the added context. The full context-stress results are reported in Table 13.

D.5 Trajectory length characteristics

Figure 3 provides descriptive statistics of the recorded interaction traces in the reporting corpus. These statistics characterize corpus scale rather than task difficulty or the number of substantive conversational turns.

Message count should not be interpreted directly as the number of substantive conversational turns. In the deployed system, execution progress is often emitted as separate assistant messages—for example, "retrieving schema," "entering solution design," "generating SQL," "waiting for execution," and "validating table creation"—and each such update is counted as an individual message. For example, trajectory `T01` contains 26 dialogue messages but only 6 user messages, with the remaining 20 produced by the assistant. The collected trajectories contain only user and assistant roles; tool events are not represented as separate dialogue messages.

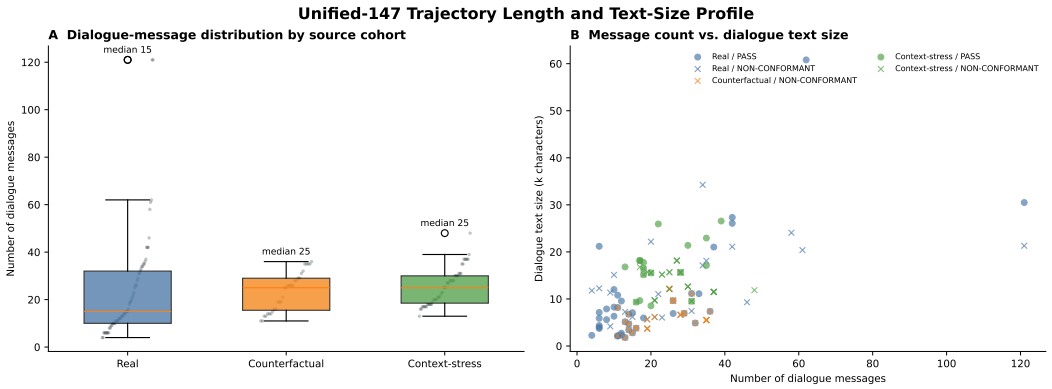


Figure 3: Trajectory-length profile of the reporting corpus. **A**, distribution of dialogue-message counts by source cohort. **B**, relationship between dialogue-message count and total dialogue text size, separated by cohort and Gold label. Message counts include assistant progress updates emitted during execution and therefore characterize trace length rather than the number of substantive conversational turns.

D.6 Evaluation Unit, Gold labels, and annotation protocol

The unit is a complete task trajectory. A case is labeled PASS only when every mandatory requirement in the latest accepted revision is satisfied by the terminal delivery. It is NON-CONFORMANT when at least one mandatory requirement has an observable violation. Evaluated verifiers may output INCONCLUSIVE, but admitted Gold cases are binary because their stored annotation views are adjudicable. Annotators read the contract before the final artifact and cannot inspect verifier predictions, mutation metadata, or prelabels. Agent self-report alone does not establish delivery.

Two independent annotator views and third-party adjudication are the target protocol. The current corpus includes legacy and controlled material for which the independent annotation audit is still pending; therefore the paper does not claim final population-level label reliability.

D.7 VERDICT METRICS

Pass retention is the fraction of Gold-Passed cases predicted PASS. NC recall is the fraction of Gold-Failed cases predicted NON-CONFORMANT. NC escape is the fraction of Gold-Failed cases predicted PASS; abstentions are not counted as escapes. Coverage is the fraction receiving a non-abstaining decision, and selective risk is the error rate among decided cases. Correct scores abstention as incorrect.

D.8 STATISTICAL PROTOCOL AND INTERPRETATION

Confidence intervals in the original evaluation protocol use 5,000 bootstrap samples, clustering variants that share a parent. Paired verdict comparisons use exact two-sided McNemar tests (as reported in Table 11). Because the corpus contains parent-linked variants and most methods have a single full-corpus stochastic run, pairwise p-values are treated as supporting descriptive evidence rather than as the sole basis for a claim. The current production trajectory block also remains preliminary with respect to complete independent double annotation; no result is interpreted as a deployment prevalence estimate or cross-domain generalization claim.

E COMPARED METHODS AND EVALUATION INTERFACES

E.1 STANDARD BASELINES

B0: Agent self-report. Trusts the terminal completion claim and performs no independent verification.

B1: Final-output judge. Judges the user requirements, terminal message, and final artifacts while omitting intermediate proposals and revisions.

B2: Trajectory judge. Receives the raw visible trajectory and final artifacts and produces one holistic verdict. It is not given an explicit active contract, requirement-level evidence bindings, or deterministic-validator outputs.

B3: Agent-as-a-Judge + tools. May inspect artifacts and invoke schema/SQL tools, but has no persistent accepted-revision ledger or fixed requirement-level reconciliation rule.

B4: Generated rubric + LLM. First freezes an LLM-generated task specification and then uses a second LLM call to grade the delivery against that specification.

B5: Deterministic validators only. Applies deterministic validators to explicitly parsed targets and abstains when a unique machine-checkable target cannot be resolved.

E.2 CANONICAL INPUT ISOLATES REPRESENTATION FROM RECONCILIATION

B2 with canonical input preserves B2's one-call holistic verdict and the same observable task information. The intervention is purely representational: free-form interaction and artifacts are reorganized into stable, indexed fields so that the judge does not need to discover basic document structure from raw transcript formatting. It still must infer which requirements remain active and whether the delivery satisfies them. A simplified example comparing the raw trajectory view with the canonical input view is shown below.

Illustrative raw view.

Raw trajectory view

Turn 1, user: Create tables A, B, and C in `ads`.
Turn 2, assistant: Proposed schema for A/B/C.
Turn 3, user: Keep A and B, replace C with D; use `DECIMAL(5,4)` for `rate`.
Turn 4–6: tool calls, intermediate files, explanations, completion claim.
Artifacts: several SQL/DDL files with filenames and contents.

Illustrative canonical view of the same observable information.

Canonical input view

Indexed dialogue: [U1] create A/B/C; [A1] proposed schema; [U2] keep A/B, replace C with D, `rate DECIMAL(5,4);...`
Candidate artifacts: A001=`a.sql`, A002=`b.sql`, A003=`d.sql`.
Extracted schema candidates: A001:{...}; A002:{...}; A003:{`rate DOUBLE`, ...}.
SQL inventory: created targets and qualified namespaces.

Crucially, the canonical view does *not* provide the conclusion that C was superseded by D, does not mark `DECIMAL(5,4)` as the active accepted expectation, does not bind that expectation to A003, and does not include any deterministic-validator verdict. Those inferences remain the judge's responsibility. This is why B2 raw versus B2 canonical measures the effect of interaction representation, whereas the gap from B2 canonical to CER-VERIFY additionally reflects revision-aware target reconstruction and requirement–evidence reconciliation.

E.3 COMPONENT-MATCHED B4 CONTROL

B4 with deterministic validators keeps B4's generated specification and applies the same Schema, Artifact Completeness, and Namespace validators used by CER-VERIFY. Our reconstructed contract is not used to repair missing, stale, or over-specified B4 expectations. This control therefore tests whether deterministic validation alone can compensate for an incorrectly inferred verification target.

E.4 VERIFICATION PROMPTS

CER-VERIFY uses one primary weak-oracle prompt and an optional review prompt. Deterministic assertion checks and final precedence are orchestration logic, not additional hidden prompts. The boxes below are faithful abridgments:

Primary weak-oracle judge (faithful abridgment)

Instruction. Decide whether the agent's final delivered state agrees with the last effective decisions and final delivery claims. Build a decision ledger before judging.

Inputs. Indexed conversation, accepted-plan candidates, artifact candidates, schema candidates, SQL inventories, and terminal delivery evidence.

Rules. Resolve active versus superseded decisions; do not use a completion claim as proof; prefer direct artifact and authoritative tool evidence; return uncertainty when a required property lacks evidence.

Output format. Structured JSON containing artifact resolution, active contracts, self-consistency, task completion, business risks, and whether review is required.

Conditional reviewer (faithful abridgment)

Instruction. Review only an ambiguous artifact-resolution or high-confidence semantic contradiction flagged by the primary judge.

Inputs. The relevant indexed evidence and the primary structured decision.

Rules. Do not invent requirements or unavailable artifacts. Identify the conflicting locator and state whether the primary decision is supported, contradicted, or unresolved.

Output format. A structured review decision and concise evidence-grounded reason.

E.5 JUDGE CONFIGURATIONS

The primary comparison uses DeepSeek V4 Flash with reasoning disabled. Selected methods are additionally repeated with GLM-5.2, Claude Opus 4.8, GPT-5.5, and MiniMax M3 under their reported thinking configurations. Backbone and reasoning mode change together, so cross-configuration differences are descriptive rather than causal model-family effects.

F Additional Experimental Results

F.1 Secondary conformance score

To capture the operational impact of incorrectly blocking conformant cases, we also retain the 0–100 Conformance Score as a secondary summary. Let A be correctly accepted conformant cases, B false blocks, and D correctly blocked non-conformant cases. For each correctly blocked case i with parseable Gold reason identifiers, let r_i be the fraction of required identifiers present in the structured prediction. If G failed cases have such identifiers and N is the corpus size,

$$\text{Score} = 100\,\frac{\max\{0,\ A - B + D + \sum_i r_i\}}{N + G}. \tag{8}$$

The main paper emphasizes component metrics because the scalar combines correctness, false blocking, and diagnosis credit and does not replace the coverage–risk trade-off.

F.2 Full unified-147 comparison

The main paper reports a compact verdict table. Table 8 retains the full set of standard baselines, component-matched controls, Conformance Score, NC escape, and abstention. The full results highlight the trade-off between non-conformance detection and pass retention: stronger blocking can catch more failures, but may also incorrectly reject valid deliveries. This distinction matters in production, where false blocks can disrupt otherwise correct workflows and are therefore costly. CER-Verify provides the best balance under this metric, retaining 55/57 conformant cases while detecting 66/90 non-conformant ones, and achieves the highest Conformance Score (81.2). B4 with deterministic tools detects more failures (71/90), but at substantially lower pass retention (26/57).

Method	Verification input	Score	Correct	Pass retain	NC detect	NC escape	Abstain
Standard baselines							
B0: Agent Self-report	terminal claim only	34.5	57/147	**57/57**	0/90	90/90	**0/147**
B1: Final-output Judge	final request + final delivery	40.8	93/147	28/57	65/90	20/90	8/147
B2: Trajectory Judge	raw trajectory; holistic verdict	33.6	61/147	23/57	38/90	28/90	47/147
B3: Agent-as-a-Judge w/ tools	raw trajectory + agentic tools	23.3	48/147	14/57	34/90	11/90	76/147
B4: Generated Rubric w/ LLM	frozen inferred specification	54.5	89/147	35/57	54/90	18/90	34/147
B5: Deterministic tools only	parsed targets; tools only	29.7	39/147	9/57	30/90	**8/90**	100/147
Component-matched controls							
B2 w/ canonical input	structured input; same holistic verdict	62.8	101/147	46/57	55/90	25/90	13/147
B4 w/ deterministic tools	B4 specification + our validators	51.9	97/147	26/57	**71/90**	9/90	22/147
CER-Verify (ours)	active contract + grounded evidence + validators	**81.2**	**121/147**	55/57	66/90	16/90	9/147

Table 8: Complete unified-147 result table. Score is retained as a secondary summary; component verdict metrics expose the different operating points.

F.3 Structural decomposition in detail

As shown in Table 9, the structural ladder is not monotonic in overall correctness. Revision-aware target reconstruction primarily reduces false blocks, while explicit evidence binding introduces a more conservative operating point with increased abstention. The final transition provides the cleanest controlled attribution: adding deterministic validators corrects 42 cases with no regressions, while keeping the semantic calls unchanged.

F.4 Deterministic validator ablation

As presented in Table 10, Schema, Artifact Completeness, and Namespace change 30, 7, and 5 decisions, respectively, and every change is a correction in the frozen 147-case ablation. The later validators therefore add recall only on properties for which the prerequisite target/evidence resolution has already succeeded.

Added layer	Changed	Corrected	Regressed	Net correct
Revision-aware target	43	25	18	+7
+ Evidence binding	50	17	29	-12
+ Deterministic validators	42	42	0	+42

Table 9: Adjacent verdict transitions in the structural study. This table describes the observed operating-point shifts; only the last row is an equal-semantic-call comparison.

Configuration	Score	Correct	Pass retain.	NC detected	NC escape	Abstain
CER-VERIFY w/o deterministic validators	48.8	79/147	55/57	24/90	47/90	20/147
+ Schema tool	73.9	109/147	55/57	54/90	26/90	11/147
+ Artifact Completeness tool	78.2	116/147	55/57	61/90	21/90	9/147
+ Namespace tool	81.2	121/147	55/57	66/90	16/90	9/147

Table 10: Cumulative deterministic-validator ablation on unified-147. All rows reuse the same frozen semantic calls.

F.5 PAIRED COMPARISONS

We additionally examine paired verdict changes between verifier variants using exact two-sided McNemar tests (see Table 11). For each comparison, "Left only" counts cases for which only the method on the left produces the correct verdict, whereas "Right only" counts cases for which only the method on the right is correct. Cases on which both methods agree in correctness do not contribute to the McNemar statistic.

Across all four comparisons, the discordant pairs favor the method on the right. Adding completeness checking to the schema-only verifier corrects 7 cases without introducing a reverse error (0 vs. 7), corresponding to a net paired accuracy increase of 4.8 percentage points over the 147 cases. Relative to the schema-only verifier, CER-VERIFY corrects 12 cases with no reverse changes (0 vs. 12; +8.2 points). The same directional pattern holds for the stronger controls: CER-VERIFY yields 25 gains versus 5 regressions relative to B2 canonical (+13.6 points), and 39 gains versus 15 regressions relative to B4 + tools (+16.3 points).

The exact McNemar p-values indicate that these paired disagreements are unlikely to be directionally balanced under the standard matched-pair assumption. We nevertheless treat these tests as supporting paired evidence rather than stand-alone inferential claims. Several evaluation cases are variants derived from the same parent trajectory and are therefore not fully independent observational units; moreover, stochastic verifier variants are represented by a single run. The tests consequently characterize the direction and consistency of paired verdict changes on this evaluation corpus, rather than providing an independent estimate of population-level significance.

Comparison	Left correct only	Right correct only	p
Schema tool vs. + Completeness tool	0	7	0.0156
Schema tool vs. CER-VERIFY	0	12	0.0005
B2 w/ canonical vs. CER-VERIFY	5	25	0.0003
B4 w/ tools vs. CER-VERIFY	15	39	0.0015

Table 11: Exact two-sided McNemar comparisons of paired verdict correctness over 147 evaluation cases. "Left-only correct" and "Right-only correct" report discordant pairs. Tests are reported as supporting paired evidence because the corpus contains parent-linked variants and stochastic methods are evaluated with a single run.

F.6 RESULTS BY CORPUS CONSTRUCTION

Table 12 reports all nine main DeepSeek methods separately on real, counterfactual, and context-stress cohorts. These splits serve different roles: naturalistic evaluation, sensitivity to controlled violations, and invariance to irrelevant interaction context.

Cohort	Method	Correct	Pass retain	NC recall	NC escape	Abstain
Real (65)	B0 Self-report	40/65	40/40	0/25	25/25	0/65
	B1 Final-output	40/65	22/40	18/25	6/25	4/65
	B2 Trajectory	25/65	16/40	9/25	10/25	23/65
	B3 Agent+tools	24/65	12/40	12/25	5/25	28/65
	B4 Rubric	38/65	25/40	13/25	8/25	15/65
	B5 Tools only	10/65	5/40	5/25	0/25	55/65
	B2 w/ canonical	43/65	33/40	10/25	11/25	7/65
	B4 w/ tools	35/65	18/40	17/25	4/25	11/65
	CER-VERIFY	**52/65**	**39/40**	13/25	9/25	**4/65**
Counterfactual (31)	B0 Self-report	0/31	–	0/31	31/31	0/31
	B1 Final-output	23/31	–	23/31	6/31	2/31
	B2 Trajectory	21/31	–	21/31	5/31	5/31
	B3 Agent+tools	17/31	–	17/31	6/31	8/31
	B4 Rubric	19/31	–	19/31	4/31	8/31
	B5 Tools only	10/31	–	10/31	4/31	17/31
	B2 w/ canonical	20/31	–	20/31	7/31	4/31
	B4 w/ tools	24/31	–	24/31	3/31	4/31
	CER-VERIFY	**26/31**	–	**26/31**	3/31	**2/31**
Stress (51)	B0 Self-report	17/51	17/17	0/34	34/34	0/51
	B1 Final-output	30/51	6/17	24/34	8/34	2/51
	B2 Trajectory	15/51	7/17	8/34	13/34	19/51
	B3 Agent+tools	7/51	2/17	5/34	0/34	40/51
	B4 Rubric	32/51	10/17	22/34	6/34	11/51
	B5 Tools only	19/51	4/17	15/34	4/34	28/51
	B2 w/ canonical	38/51	13/17	25/34	7/34	2/51
	B4 w/ tools	38/51	8/17	**30/34**	2/34	7/51
	CER-VERIFY	**43/51**	**16/17**	27/34	4/34	3/51

Table 12: Verdict results by corpus construction. The counterfactual cohort contains no Gold-Passed cases, so pass retention is undefined there.

G ROBUSTNESS, STABILITY, AND EFFICIENCY

G.1 CONTEXT-STRESS PAIRED ANALYSIS

All 51 context-stress variants preserve their parent's Gold verdict. Table 13 reports both verdict consistency and correctness consistency, separating invariance from absolute correctness.

Canonical input has the highest verdict consistency, while CER-VERIFY has the highest parent and variant accuracy. This supports a two-part interpretation: canonicalization reduces sensitivity to irrelevant context, whereas contract-aware reconciliation improves the quality of the stabilized decision.

Method	Parent acc.	Variant acc.	Verdict consist.	Correctness consist.	Both correct	Both incorrect	Harmful flips	Beneficial flips
B2	35/51	15/51	23/51	27/51	13/51	14/51	22/51	2/51
B2 w/ canonical	38/51	38/51	**45/51**	**45/51**	35/51	10/51	**3/51**	3/51
B4	36/51	32/51	31/51	37/51	27/51	10/51	9/51	5/51
CER-VERIFY	**48/51**	**43/51**	44/51	44/51	**42/51**	**2/51**	6/51	**1/51**

Table 13: Paired context-stress analysis. All 51 variants preserve their parent's Gold verdict.

G.2 JUDGE-CONFIGURATION ROBUSTNESS

Table 14 shows that CER-VERIFY has the largest number of correct verdicts within each of the five evaluated judge configurations. Absolute operating points differ, so the table should not be read as a ranking of model families.

G.3 THREE-RUN STOCHASTIC STABILITY

Figure 4 evaluates stochastic stability under a fixed DeepSeek V4 Flash configuration. We use `avg@3` to denote the arithmetic mean of three independently sampled run-level Scores; it is an evaluation statistic rather than majority voting or a deployment-time multi-sampling strategy.

Judge configuration	Method	Score	Correct	Pass retain	NC detect	Abstain
DeepSeek V4 Flash, w/o thinking	B2	33.6	61/147	23/57	38/90	47/147
	B4	54.5	89/147	35/57	54/90	34/147
	B2 w/ canonical	62.8	101/147	46/57	55/90	13/147
	B4 w/ tools	51.9	97/147	26/57	**71/90**	22/147
	CER-VERIFY	**81.2**	**121/147**	**55/57**	66/90	**9/147**
GLM-5.2, w/ thinking	B2	15.6	25/147	11/57	14/90	96/147
	B4	61.0	104/147	37/57	67/90	21/147
	B2 w/ canonical	64.7	104/147	48/57	56/90	17/147
	B4 w/ tools	47.5	98/147	20/57	**78/90**	13/147
	CER-VERIFY	**80.5**	**118/147**	**56/57**	62/90	**7/147**
Claude Opus 4.8, w/ thinking	B2	20.4	34/147	11/57	23/90	89/147
	B4	44.5	79/147	39/57	40/90	38/147
	B2 w/ canonical	53.1	89/147	46/57	43/90	24/147
	B4 w/ tools	55.1	101/147	31/57	**70/90**	23/147
	CER-VERIFY	**70.3**	**107/147**	**56/57**	51/90	**8/147**
GPT-5.5, w/ thinking	B2	22.9	43/147	5/57	38/90	86/147
	B4	44.2	90/147	16/57	74/90	25/147
	B2 w/ canonical	57.0	94/147	28/57	66/90	37/147
	B4 w/ tools	40.2	93/147	11/57	**82/90**	**15/147**
	CER-VERIFY	**63.9**	**105/147**	**39/57**	66/90	**15/147**
MiniMax M3, w/ thinking	B2	14.7	29/147	7/57	22/90	87/147
	B4	55.8	93/147	35/57	58/90	31/147
	B2 w/ canonical	55.0	91/147	45/57	46/90	27/147
	B4 w/ tools	45.9	97/147	16/57	**81/90**	15/147
	CER-VERIFY	**77.6**	**116/147**	**55/57**	61/90	**9/147**

Table 14: Judge-configuration robustness on the same 147 cases.

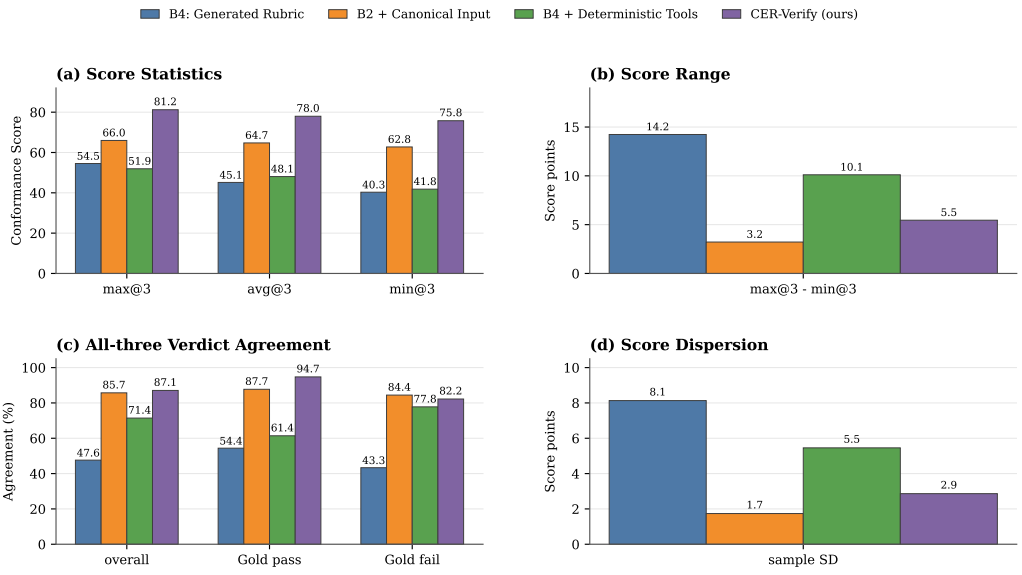


Figure 4: Run-to-run stability of four methods over three independent executions on the unified 147-case corpus. The panels report run-level Conformance Score statistics, Score range, all-three verdict agreement, and sample standard deviation.

CER-VERIFY retains the strongest `avg@3` performance and the highest all-three verdict agreement, while B2 with canonical input exhibits the lowest run-to-run dispersion at a lower performance level. The rubric-based controls vary substantially more across runs. For CER-VERIFY, pass retention remains nearly unchanged across executions, whereas non-conformance detection varies more noticeably, suggesting that the remaining stochasticity is concentrated primarily in residual semantic decisions rather than in the structured verification scaffold.

This analysis covers four selected methods under one judge configuration. It therefore establishes within-configuration stochastic stability, rather than cross-configuration reproducibility or a matched-cost comparison.

G.4 Inference cost

The reported pipelines are not budget-matched: source-run transport/cache conditions differ, and canonicalization is not charged to the B2 control (Table 15). Deterministic validators execute locally and add no measurable latency at the reported precision. The CER cumulative ablation reuses a single set of semantic calls, so its four rows should be understood as alternative views of the same LLM-call set rather than separately incurred inference costs.

Method	Mean tokens	Median tokens	Mean latency (s)	Median latency (s)
B0 Self-report	0	0	0.0	0.0
B1 Final-output	5,312	3,881	5.3	2.8
B2 Trajectory	6,915	6,491	2.7	2.1
B3 Agent+tools	26,444	25,480	55.8	43.1
B4 Rubric	14,167	12,609	29.8	23.3
B5 Tools only	0	0	0.0	0.0
B2 w/ canonical	13,774	12,053	5.7	2.6
B4 w/ tools	13,924	12,504	23.6	21.3
CER-VERIFY	21,771	17,568	30.6	23.4

Table 15: Per-trajectory inference cost for the main DeepSeek comparison. The pipelines are not budget-matched.

H Failure Boundary and Error Attribution

H.1 Audit protocol

Diagnostic cases are selected mechanically from the frozen predictions and Gold labels. For each selected case, the original interaction, active contract state, artifact binding, and available tool evidence are inspected, and one mutually exclusive primary cause is assigned according to the earliest causal failure in the verification pipeline. Contract-state errors precede artifact-resolution errors, which precede downstream evidence/tool symptoms unless an explicit checker defect is the direct cause. The three audited sets have different denominators and may overlap in case IDs, so their columns are diagnostic and not additive.

H.2 Why tool access alone abstains

As reported in Table 16, B3 abstains on 76/147 cases in the frozen unified result. Of these, 62 are attributed to artifact resolution, nine to contract ambiguity, four to tool uncertainty, and one to insufficient evidence. The dominant issue is therefore not the absence of tools but failure to resolve observations to the terminal artifacts required by the contract.

Failure locus	B4 w/ tools False blocks	B3 Abstentions	CER-VERIFY Residual errors
Contract state / revision reconciliation	6	9	4
Artifact / namespace resolution	3	62	9
Schema expectation / comparison	6	0	10
Semantic evidence / decision policy	0	1	3
Tool / checker uncertainty	4	4	0
Total	19	76	26

Table 16: Primary failure loci in three diagnostic error sets. Denominators differ across columns and case IDs may overlap; counts are diagnostic and are not additive.

H.3 When exact checks false-block

The B4-plus-tools audit contains 19 false blocks. Six originate from obsolete, superseded, or never-accepted requirements; six from incorrect schema expectations; three from wrong artifact/namespace binding; and four from direct tool/checker errors. Representative cases include treating source dependencies as missing output artifacts, enforcing assistant-proposed directory conventions as mandatory requirements, and converting underspecified generic numeric types into exact schema constraints.

H.4 Residual errors of CER-Verify

The 26 residual errors consist of 16 false accepts, eight failed-case abstentions, one passed-case abstention, and one false block. Primary causes are schema comparison (10), artifact/namespace resolution (9), contract/revision reconciliation (4), semantic verification (2), and evidence/decision policy (1). Thus 19/26 residual errors lie in schema representation/comparison or artifact resolution.

Representative residual cases illustrate the current boundary: schema extraction may fail to preserve accepted precision or closed-schema constraints; missing promised artifacts may remain unresolved rather than becoming a grounded omission; and semantic-only requirements such as lineage or filter logic can escape when no deterministic validator exposes the relevant property.