Reliable Context

RELIABLE CONTEXT / RESEARCH & PRACTICAL GUIDANCE

JSON Evidence Binding Test: Right Value, Right Record?

> Reliable Context is a kernel of truth: a system of record for organizations that fuse private and public information and must defend every value they act on. One canonical record per real thing, a cited source and an observation date behind every value, unknowns declared rather than left blank, and any past state of the record reproducible.

Published by Reliable Context · 2026-11-01 · json evidence binding test · data provenance validation json · value path binding · llm json record misplacement · json pointer rfc 6901 · structured output evaluation · data provenance vs data lineage · deterministic transformation baseline · source attribution for data values · audit trail for data values

Original article: https://reliablecontext.com/articles/json-evidence-binding-test-right-record



In this article

1. Executive Summary
2. Introduction and Background
3. Key Changes in How Structured Output Should Be Evaluated
4. Binding Ledger Protocol: Implementation Considerations and Process Changes
5. Provenance, Lineage and Evidence Binding
6. Data Analysis and Evidence
7. Implications and Future Directions
8. Frequently Asked Questions (FAQs)
9. Conclusion

Executive Summary

A JSON document can parse, satisfy its schema and contain every correct string, yet still bind a value to the wrong record, parent or hierarchy level. A **JSON evidence binding test** asks, for each source record supplied to a model, whether that record's content appears exactly once at the location the grouping rules intend. This report, current as of 31 October 2026, builds a reusable evaluation ledger for that question around one recent paper and a handful of standards.

The anchor is Zhang et al., "Right Values, Wrong Places," revision 2 dated **14 October 2026**. It introduces JSONPlacement, a task that supplies every value and every grouping rule, so a misplaced value cannot come from misreading text. The authors separate record-assignment errors from hierarchy-realization errors. They report that whole-record moves make up **98.6%** of strictly traceable synthetic misplacements (arxiv.org). On their original structure, one 12B open-weight model preserved **97.2%** of target values yet bound only **53.4%** to the correct path (arxiv.org). These are author-reported results in a synthetic setting, not deployment guarantees.

Schema validity does not close the gap. One vendor documents schema-compliant responses through constrained decoding (platform.claude.com); another states that structured outputs can still contain mistakes (developers.openai.com); and an independent benchmark of 21 models reports exact leaf-value accuracy of **0.693 to 0.830** on text despite near-perfect schema compliance (arxiv.org). A pointer such as RFC 6901 names a location in one document (datatracker.ietf.org); it does not establish who the value is about.

The **proposed protocol** has six parts: a frozen source manifest with scoring-only record identifiers, a per-record ledger, matched conditions that vary input order and output representation one at a time, a deterministic baseline, fresh held-out fixtures, and resampling by document rather than by leaf. In the paper, ordering the input by target traversal and writing record-level paths were complementary, and combining them solved every new synthetic instance (arxiv.org). Record-level paths also lengthened outputs and increased truncation.

The decision rule is plain. If the mapping from records to hierarchy is fully specified, conventional transformation code is the appropriate tool. If a model must organize, change the representation only after diagnosing which error type dominates. No experiment was run for this report, and transfer to extraction with inferred assignments remains untested (arxiv.org).

Introduction and Background

Context engineers increasingly hand structured text from a language model to another program. Two questions are routinely conflated: whether the output is well formed, and whether each value still belongs to the subject it was supplied with. A syntactic validator answers only the first. JSON Schema is a declarative language for defining structure and constraints for JSON data (json-schema.org). It can restrict a value to a fixed set of values (json-schema.org), although one vendor documents that a returned enum value might differ from the schema only in capitalization (platform.claude.com). It cannot say which source record a permitted value was drawn from.

This report treats that gap as an information problem, not a sector problem. It is a protocol proposal: nothing in these pages was measured by the author of this report.

Definitions used throughout

- **Value preservation:** the share of target values that appear somewhere in the output with the right type and content, regardless of path. The JSONPlacement paper defines it as target leaves matched one-to-one to output values, ignoring paths.
- **Source-record identity:** a stable identifier for each supplied record, held by the evaluator. It is an independent input to scoring, not something recovered from the path the model chose.
- **Intended parent and path:** the location the grouping rules assign to a record, expressed as a JSON Pointer in a particular output document.
- **Output schema validity:** conformance to a schema. It certifies shape and types, and it says nothing about which record a value came from.
- **Correct binding:** every content field of a supplied record is written once, with its original name, type and value, at that record's intended path, with no extra or conflicting declarations.
- **Untraceable record:** a record whose output location cannot be established without a convenient guess, because its content is duplicated, altered or ambiguous. It is labeled as such and is never assigned a match for the sake of a tidy denominator.

Why string matching cannot certify binding

JSON Pointer (RFC 6901, April 2013) is a string syntax for identifying a specific value within a JSON document (datatracker.ietf.org). The standard warns that a given pointer is not guaranteed to reference an actual value (datatracker.ietf.org). A pointer is therefore a locator inside one document. It carries no claim about subject identity, truth, or persistence across a reorganization.

A small transformation example (Hypothetical Example)

Consider a fictional fixture with two source records. The fixture identifiers R01 and R02 are held by the evaluator and are not shown to the model.

```
[
  {"school": "North", "class": "C1", "student": "S-07", "name": "A. Reyes", "grade": "B"},
```

```
  {"school": "North", "class": "C2", "student": "S-07", "name": "B. Ortiz", "grade": "B"}
]
```

The rule is: nest students under school, then class, then student identifier. The expected pointer for R01's grade is /schools/North/classes/C1/students/S-07/grade, and for R02's grade it is /schools/North/classes/C2/students/S-07/grade. Suppose the model writes R02's whole record under class C1 and R01's whole record under class C2. The output parses and passes a schema. A leaf check on the grade field passes at both paths, because both expected values are the string "B". Only the name field, or the evaluator's record identifier, reveals that each path now describes a different student. If the fixture checked only grades, a bag of correct strings and a valid schema would coexist with two wrong subject-value assertions.

This is the shape of failure the JSONPlacement paper documents: a student in Class 1 written under Class 2 keeps name and grade, but a downstream program now counts the student in the wrong class.

Key Changes in How Structured Output Should Be Evaluated

From valid JSON to value-path binding

The revision of 14 October 2026 is titled "Right Values, Wrong Places: Diagnosing and Fixing Record Misplacement in LLM-Generated JSON" and is by Zhang, Wu, Li, Wang, Yao and Li; older index entries may show an earlier title, and this report uses the v2 title and methods. The JSONPlacement task supplies flat records, grouping fields that jointly determine each record's target path, and content fields whose names determine leaf paths. A simple program generates the reference, so every instance has one unambiguous answer, and the models only reorganize the input, with no retrieval, joins or text extraction (arxiv.org).

The headline observation is the gap between preserving a value and binding it. On the original structure, no 8 to 12B model solved more than one of 24 instances, while one 12B model preserved 97.2% of values and bound 53.4%.

Two error types, kept separate

A **record-assignment error** writes a record under another valid record path, such as a Class 1 student under Class 2 (arxiv.org). A **hierarchy-realization error** changes the parent and child relations between containers, for example writing a school at the root instead of inside the schools container (arxiv.org). A ledger must keep them apart because the published remedies differ, and labels such as omission, duplication, conflict and hierarchy displacement can overlap and must not be summed as mutually exclusive categories.

Whole-record movement is the dominant pattern in the traceable subset. In four earlier synthetic conditions, 80.8% of misplaced leaves could be traced to a unique destination (arxiv.org), and 98.6% of those belonged to whole-record moves. If fields erred independently, about 0.06 records would be expected to move whole; the authors observed 6,293.

Diagnose before intervening

The paper derives two interventions from the error patterns and fixes its predictions before generating new instances. Ordering the input by target traversal targets assignment errors; replacing nested output with record-level paths targets hierarchy errors. On new synthetic instances, ordering nearly eliminated records written to other valid paths and whole-record omissions for all three models (arxiv.org), while record-level paths showed the opposite pattern and lengthened outputs so that more reached the token limit (arxiv.org).

A dose-response check supports the ordering explanation: for one model, exact matches rose from 0 under full shuffling to 10 under local shuffling and 24 under full ordering. Location in the output does not by itself identify the cause, since both closure errors and key-selection errors can produce out-of-tree records. The authors state that their evidence establishes behavioral patterns, not a unique internal mechanism, and this report draws no conclusion about hidden model mechanisms.

What schemas and constrained decoding do and do not certify

JSONSchemaBench (Geng et al., 27 February 2025 revision) is a benchmark for constrained decoding with 10K real-world JSON schemas (arxiv.org). It pairs the schema set with the official JSON Schema Test Suite, a language-agnostic collection that validator implementers use to test their libraries (github.com). Its quality component evaluates specific reasoning tasks. It separates coverage, efficiency and quality, which is the right separation, but it does not supply a source-record-binding certification, and this report makes no claim that the paper ignores quality or that constrained decoding generally harms accuracy.

Vendor documentation is consistent with that reading. One vendor states that structured outputs ensure the model always generates responses that adhere to the supplied schema (developers.openai.com) and also that they can still contain mistakes (developers.openai.com). Another documents schema-compliant responses through constrained decoding (platform.claude.com), with exceptions, including an output that may be incomplete and fail the schema when it hits the token limit (platform.claude.com). Truncation matters here because the record-path intervention raises it. These are vendor statements about their own features, and each is limited to schema adherence.

Adjacent evidence on syntax versus semantics

Several recent studies separate syntactic compliance from content correctness, with different tasks, so their rates are not comparable to each other or to JSONPlacement.

- **StructHallu-Drift** (Hasan, SURGeLLM, 3 July 2026) proposes a taxonomy that disentangles syntactic validity from semantic fidelity (aclanthology.org). It mutates each of 100 schemas at three levels (aclanthology.org), and among outputs that passed schema validation, the residual semantic hallucination rate was 8.0% (aclanthology.org), and the authors conclude that no single validation layer suffices (aclanthology.org). Its field-confusion type, a value in the wrong schema field (aclanthology.org), is the nearest analogue, but its task changes the schema and differs from a fixed-schema organization test.
- **Structured Output Benchmark** reports near-perfect schema compliance alongside much lower exact leaf-value accuracy (Value Accuracy of 0.693 to 0.830 on the text modality across 21 models; arxiv.org), and it is question-driven extraction, not reorganization of supplied records.

- **Tool-output processing** reports a qualitative failure of picking a value for the right key from the wrong record in JSON responses (arxiv.org). That is the same family of error in a question-answering setting.

Binding Ledger Protocol: Implementation Considerations and Process Changes

The protocol below is a proposal. It translates the paper's design into an auditable procedure and adds elements the paper does not supply, such as array-aware pointer fixtures.

Step 1: Freeze a source manifest

Create a fixture set before any model call. Each source record receives a stable fixture identifier, and the manifest stores the original source text or cells, the explicit grouping rules, and a separately computed expected output. The JSONPlacement artifacts illustrate the practice: dataset and per-task hashes in the run metadata identify the exact archived inputs (arxiv.org). A canonical serialization makes your own hash repeatable: RFC 8785 describes an invariant format so that hashing operations are reliably repeatable (datatracker.ietf.org) and requires input without duplicate property names (datatracker.ietf.org) while keeping array element order (datatracker.ietf.org).

Resolve or disclose ambiguous mapping rules before running the model. Identifiers used only for scoring must not alter the model's input, because an identifier-visible arm is a distinct intervention. The paper's unique-identifier diagnostic shows why: making grouping identifiers unique cut records written to other valid paths sharply, yet exact matches changed little because omitted and out-of-tree records remained.

Step 2: Validate parse and schema first, then bind

Run the checks in this order and keep each stage's denominator.

- **Parse and schema compliance:** count all attempted outputs, including truncated and invalid ones. The paper counts truncation, invalid JSON and uninterpretable representations as whole-instance failures.
- **Complete-record comparison:** compare each supplied record, field names and types included, against the output. Preserve extra, missing and duplicated records instead of aligning them away.
- **Subject-value-path binding:** require the right value at the right pointer for the right fixture record. A matching leaf is insufficient when the same value occurs more than once.
- **Conditional diagnosis:** classify placement errors only for preserved, traceable records, and report that count beside the all-record denominator.

The paper's scoring is a useful reference. It compares reference and output by full paths, node types and leaf values, ignoring object-key order and applying no output repair. Matching first preserves correct bindings, then matches remaining equal-valued entries, so no output value can satisfy multiple targets. The evaluator also retains every declaration of duplicate keys, which matters because RFC 8259 says object names SHOULD be unique (datatracker.ietf.org) while many implementations report the last name and value pair (datatracker.ietf.org). A parser that silently keeps the last duplicate can hide a conflicting declaration.

Step 3: Keep the ledger

Table 1 below lists the fields of the proposed binding manifest. Each row of the ledger corresponds to one supplied source record or one source field-value pair.

Ledger field	Content	Why it is needed
Input version and hash, retrieval date	Digest of the frozen manifest and the date the inputs were retrieved or generated	Makes a rerun comparable
Source record ID and locator	Fixture ID, plus the cell, row or span in the original source	Identity is an independent input, not inferred from the model's path
Subject and field/value	The entity the record describes, the field name, the typed value	Distinguishes a value from the subject it is asserted about
Expected parent and pointer	JSON Pointer under the grouping rules	Names a location within one particular document
Actual parent and pointer	Where the evaluator found the record's content, or "none"	Supports assignment versus hierarchy diagnosis
Value changed	Yes or no, with the observed value	Separates a moved record from an altered one
Duplicated or omitted	Count of complete occurrences in the output	Extra copies must stay visible, not be aligned away
Traceability status	Traceable, untraceable, or ambiguous, with the reason	Untraced values are reported, not forced into a match
Supporting evidence	The source text or cell that justifies the expected path	Lets a reviewer defend the oracle, not only the score

The table works as a checklist, not a data model. Its design choice is that expected and actual locations are recorded per source record, so a reviewer can answer "which record is this value from?" without re-running anything. Exact tracing requires known target bindings and identifiable source records; when alignment cannot be established, label the record untraceable. Source records with identical complete tuples cannot support unique movement attribution either.

Step 4: Pointer fixtures that survive RFC 6901

Pointer handling is a place where an evaluator can introduce its own errors. Include fixtures for each of the following.

- **Escaped slash and tilde:** a key such as "Q3/Q4~adj" (a fictional fixture key) is written with the slash as ~1 and the tilde as ~0, giving the reference token Q3~1Q4~0adj. When decoding, the order matters: the RFC says that applying the substitutions in the stated order avoids the error of turning ~01 into a slash (datatracker.ietf.org).
- **Array indices:** a token addresses an array element by zero-based index (datatracker.ietf.org), and the dash token refers to a nonexistent member past the end, which results in an error when evaluated (datatracker.ietf.org).

- **Reordering:** RFC 8259 defines an array as an ordered sequence (datatracker.ietf.org), so a permitted reordering changes indices without changing the subject. A pointer change caused by permitted reordering is not automatically a semantic reassignment; key such fixtures by record identity.
- **Index shifts under edits:** in JSON Patch, an add shifts later elements one position (datatracker.ietf.org) and operations are applied sequentially (datatracker.ietf.org), so a pointer is meaningful only relative to a document state.
- **Duplicate names:** if a name is not unique in an object, the referenced member is undefined (datatracker.ietf.org).
- **Unicode forms:** member names match by code points, so differently normalized spellings are different keys (datatracker.ietf.org).

One boundary deserves emphasis. The JSONPlacement target contains only objects and leaf values, with no array alignment involved, so its published results say nothing about array-ordered targets.

Step 5: Run matched conditions

Hold the model, decoding budget and grouping policy fixed, and disclose the settings, as the paper does with its temperature and token budget. Run four prespecified conditions on the same fixtures:

- **Original input order, nested output** (baseline).
- **Grouped input order, nested output** (targets assignment errors).
- **Original input order, full record paths** (targets hierarchy errors).
- **Grouped input order, full record paths** (combined).

Input order matters beyond this task: in a long-context study, multi-document question-answering performance could drop by more than 20% when relevant information moved (aclanthology.org), and a synthetic JSON key-value task probed whether models simply retrieve matching tokens from the middle (aclanthology.org). Disclose output-length changes and context limits. Repeating a complete record path has a measurable length cost, and in the paper's path condition some outputs were truncated. Smaller error counts in a truncation-heavy cell cover fewer answers, so a higher conditional mean with fewer scorable outputs need not represent an improvement.

Use fresh, held-out fixtures for the decision after designing the intervention. Tuning against a test set leaks knowledge into the model choice, so evaluation metrics no longer report on generalization performance (scikit-learn.org); a test set should still be held out for final evaluation (scikit-learn.org). The paper's own fresh validation fixed its predictions in advance and scored every output without retries or repair.

Step 6: Add a deterministic baseline

Where the supplied rules are executable, implement them in conventional code. This checks the fixture oracle and answers whether a language model is necessary for this organization step. Keep downstream factual extraction separately evaluated. The baseline must preserve order only where order is semantically meaningful, and common tools differ on this point.

- **jq:** group_by returns an array sorted by the grouping value (jqlang.github.io), and unique_by keeps only one element per key (jqlang.github.io), so it is not a lossless grouping; unsorted key listing is only roughly in insertion order (jqlang.github.io).
- **Python:** itertools.groupby needs input already sorted on the key (docs.python.org), unlike SQL grouping, which works regardless of input order (docs.python.org); dictionary order is guaranteed to be insertion order (docs.python.org).
- **pandas:** with sorting disabled, groups appear in their original order (pandas.pydata.org), and rows keep their order within each group (pandas.pydata.org).
- **JavaScript:** Object.groupBy returns an object with one property per group (developer.mozilla.org), and integer-like keys are listed first in ascending numeric order (tc39.es), so record identifiers such as "10" and "2" can be silently reordered. Other keys follow creation order (tc39.es), and Map.groupBy uses the unique callback values as keys (developer.mozilla.org).
- **PostgreSQL:** aggregate input ordering is unspecified by default unless an ORDER BY is written inside the call (www.postgresql.org), and the binary JSON type does not preserve the order of object keys (www.postgresql.org) and keeps only the last value for duplicate keys (www.postgresql.org).

One vendor documents that structured outputs are produced in the same order as the ordering of keys in the schema (developers.openai.com), so key order there reflects the schema, not the data. Because RFC 8259 defines an object as an unordered collection of name and value pairs (datatracker.ietf.org), and parsing libraries differ in whether they expose member order (datatracker.ietf.org), compare object members by key, not position, and compare arrays by position only where the schema declares order meaningful.

Step 7: Report by depth, count, repetition and ambiguity

Stratify results by group depth, record count, repeated values and ambiguity; in the paper, value preservation and binding nearly coincided at 16 records per instance and diverged at larger counts. Report these quantities:

- **Schema-valid output fraction:** over all attempted outputs.
- **Correctly bound records:** correct records divided by all expected records.
- **Exact-document agreement:** the fraction of outputs entirely correct.
- **Extra and duplicate records:** counted, not aligned away.
- **Placement errors conditional on preserved, traceable records**, beside the all-record denominator.
- **Hierarchy realization versus wrong-record assignment**, using the published rules, with mixed or ambiguous failures left visible.

Do not infer record accuracy by multiplying independent field scores, because fields move together. Resample by generated instance or source document, not by leaf count; the paper treats conditions sharing a base instance as paired observations. Eval-statistics guidance points the same way: when questions are drawn in related groups, clustered standard errors are appropriate (arxiv.org). For paired conditions on the same fixtures, a paired bootstrap resamples one set of indices and applies it to every sample (docs.scipy.org), and for a binary per-document outcome McNemar's test applies, which Dietterich (1998) found to have low type I error (mlanthology.org), with an exact variant based on the binomial distribution (www.statsmodels.org). An unpaired bootstrap would instead independently resample the elements of each array (docs.scipy.org). Pass one element per document, not per leaf, to whichever routine is used.

Step 8: Extend to extraction only as a new experiment

Only after the supplied-record test is stable, extend to source-text extraction with independently annotated subject, field, evidence span and path. This is a transfer experiment, not something the original paper validated: its task supplies complete records and explicit grouping rules, and input ordering applies only once grouping fields are explicit. The extraction stage needs its own identity resolution and its own ledger.

Provenance, Lineage and Evidence Binding

Readers often ask how provenance differs from lineage, what counts as source attribution for a value, and what an audit trail for values must hold. The literature is less tidy than many guides suggest.

Data provenance versus data lineage

The W3C PROV data model, a Recommendation dated 30 April 2013, defines provenance as information about entities, activities and people involved in producing a piece of data or thing (w3.org). An entity may be physical, digital, conceptual or another kind of thing (w3.org), a derivation is a transformation of one entity into another (w3.org), and attribution ascribes an entity to an agent (w3.org). PROV-O expresses that model in OWL2 (w3.org).

OpenLineage describes itself as an open framework for data lineage collection and analysis (openlineage.io); its jobs are process definitions that consume and produce datasets (github.com), and its column-lineage facet records which input columns are used to produce which output columns (openlineage.io), and distinguishes a direct derivation from an indirect influence where the output column value is impacted by the value of an input (openlineage.io). That is flow-level metadata about data moving through jobs. It does not name the JSON Pointer a value occupied inside a particular document.

Authoritative sources do not agree on a crisp distinction. The database-theory survey by Cheney, Chiticariu and Tan (2009) says provenance describes the origins and the history of data (homepages.inf.ed.ac.uk), and treats the earlier lineage notion as the source tuples that witness an output tuple's existence (homepages.inf.ed.ac.uk). A vendor blog states the common practitioner split: provenance is the historical record of data, including where it was created (datahub.com), and the two terms answer different questions (datahub.com). That source is a practitioner view, not a standard. A working convention for this report: **lineage** concerns the flow of data through systems and jobs; **provenance** concerns the origin and derivation of a particular value; **evidence binding** is the further requirement that a value be attached to the correct subject record and location in the output.

Where a value came from: why and where provenance

Database provenance offers precise vocabulary for the neighboring question. Buneman, Khanna and Tan distinguish why-provenance, the source data that influenced the existence of the data (repository.upenn.edu), from where-provenance, the locations in the source databases from which the data was extracted (repository.upenn.edu). Where-provenance describes where a piece of data is copied from (homepages.inf.ed.ac.uk), and it is not invariant under equivalent queries (homepages.inf.ed.ac.uk). The binding ledger is a where-provenance record for an LLM transformation step: it maps a source location to an output location. By analogy, an expected pointer is valid only for a stated target structure and hierarchy rule.

Source attribution, fusion and the audit trail

Source attribution for data values has three layers that should be kept apart. The first is the origin: the document, cell or span a value was read from. The second is the binding: the subject and field to which the value was assigned in a given representation. The third is the observation: when and by whom the value was read. A resolvable citation covers only the first layer, and a valid schema covers none.

Knowledge fusion work makes the same separation from the other direction. Dong et al. define knowledge fusion as identifying true subject-predicate-object triples extracted by multiple information extractors (arxiv.org), and Knowledge Vault computes calibrated probabilities of fact correctness (research.google), reporting about 38 times more confident facts than the largest previous comparable system (lunadong.com). These methods decide which claim about a subject is likely true; they presuppose that each claim is already bound to the right subject. A misplacement upstream turns a true value into a false claim before fusion starts. For organizations that fuse private and public information, the practical audit trail is therefore a chain: source locator, subject, field, value, output location, observation date, and any change since.

A system of record is where that chain lives after a value is accepted. Publisher note: Reliable Context, which publishes this report, offers a system of record of this kind. That is a description of the publisher's offering and is not evidence for any claim in this report. A system of record complements the placement test: the test measures whether a transformation step preserved bindings, and the record keeps the chain of custody for accepted values. A reader who needs a vector database, a data catalog or a master data suite should use those tools; they are different products.

Data Analysis and Evidence

The evidence is thin and specific. The JSONPlacement results come from one research group, in one controlled setting, for open-weight models of 4B to 31B parameters. No source in this report measures the proposed ledger, and none supports production claims.

Table 2 below reproduces the baseline value-preservation and value-path binding figures for the original structure, and the exact-match counts for the four matched conditions on new synthetic instances, as reported by the authors.

Measure (JSONPlacement, Original structure)	Qwen3.5-9B	Gemma 4 12B IT	Ministral 3 8B Instruct
Baseline value preservation (VP), 24 instances	92.62%	97.22%	95.83%
Baseline value-path binding accuracy (VPA)	83.51%	53.39%	66.49%
Baseline exact matches, original instances	0/24	1/24	0/24
New instances, exact matches: baseline	0/24	3/24	0/24
New instances, exact matches: ordered input, nested output	24/24	8/24	17/24
New instances, exact matches: original order, record paths	0/24	12/24	0/24
New instances, exact matches: ordered input and record paths	24/24	24/24	24/24

Measure (JSONPlacement, Original structure)	Qwen3.5-9B	Gemma 4 12B IT	Ministral 3 8B Instruct
Median output tokens, ordered nested to combined (Original)	3,480 to 3,640	3,541 to 3,989	3,123 to 3,533

Source: Zhang et al., Tables 2 and 3 and Appendix E.2, author-reported. Counts are exact matches out of 24 per cell, from runs at temperature 0 with a 16,384-token output budget.

Three readings of *Table 2* are supported. First, preserved and bound values diverge widely in the baseline: the gap is 9 points for one model and 44 points for another. Second, the interventions are complementary and model-dependent: input ordering alone lifted one model to 24 of 24, while another needed the combination. Third, the combined condition costs tokens: median output length rose by roughly 160 to 450 tokens per instance on Original, and record paths alone produced truncations in some cells. Nothing here shows that the same gains hold for other models, for extraction, or at production scale; the authors note that the combined intervention has not been validated across multiple real-data sources.

Repeated values deserve a separate comment. The JSONPlacement synthetic values are unique within each instance, while in real data repeated individual values still preclude applying the stricter single-occurrence leaf criterion (arxiv.org). The proposed ledger turns that limitation into a reported quantity: count the share of records whose content tuple is duplicated, and stratify by it.

Table 3 below places the controls discussed so far side by side, with what each can and cannot certify about binding.

Control	What it can certify	What it cannot certify
JSON Schema validation	Shape, types and permitted values (json-schema.org)	Which source record a value came from
Constrained decoding or vendor structured outputs	Schema adherence within a supported schema subset (one vendor lists up to 5000 object properties (developers.openai.com)), subject to documented exceptions such as truncation (platform.claude.com)	Value-to-record binding; documented as able to contain mistakes (developers.openai.com)
Tree-shape grammar (author-reported)	Fixes container nesting, key names and identifier formats stated in the prompt	Record bindings or values; the variant that also fixed the top-level group count is gold-informed (arxiv.org)
JSON Pointer	A location in one document	Subject identity, truth, or persistence across reorganization
Binding ledger with oracle (proposed)	Per-record expected versus actual location, with traceability status	Truth of the source itself; transfer to extraction
Deterministic assembly	The executable grouping rules, with order handled explicitly	Cases where the mapping is not fully specified
System of record (publisher note: Reliable Context)	A system of record should store the source locator, observation date and change history behind each accepted value	It is not a placement test; it complements one

Table 3 shows that no single control answers the binding question. The gold-informed row deserves attention: a grammar that also fixes the number of top-level groups uses knowledge of the answer, and with it the model in that diagnostic was correct on every instance. Such a constraint is a diagnostic, not a deployable control, whenever the group count would not be known in advance. The final row describes the publisher's own offering and is not evidence about the other rows.

Implications and Future Directions

A decision rule for context engineers

The decision is among three options: retain model-based organization, change the representation, or use deterministic assembly.

- **Use deterministic assembly** when the mapping is fully specified and executable. The paper suggests sorting generated records by their grouping fields before organization and notes that a program can assemble the hierarchy (arxiv.org). Then conventional transformation code is the appropriate tool, and the model's job, if any, moves upstream to producing the records.
- **Change the representation** when a model must organize and the ledger shows a dominant error type. Assignment errors point to input organization; hierarchy errors point to the output form. Ordering presupposes that grouping fields are explicit, so identity must have been resolved earlier.
- **Retain model-based organization** when held-out, cluster-resampled results meet a threshold fixed before the run, for the depth, record count and repetition profile that production data will have, and when truncation under any longer representation stays within the budget.
- **Do neither** when alignment cannot be established. Label the records untraceable and fix the identity input first.

What remains unknown

- **Transfer to extraction:** untested in the paper, which leaves open whether its interventions help where assignments must first be inferred from text.
- **Mechanism:** behavioral patterns do not identify an internal cause, and record-level paths change output length, bracket structure and grouping-key repetition together, so effects cannot be attributed to one factor.
- **Public artifacts:** the paper's footnote gives a link to an anonymized code repository, and its evidence index lists archive-relative run identifiers and generation commits. This report did not open the repository or run any code, so runnable reproduction should not be promised until the artifact is pinned and exercised.
- **Array-ordered targets:** the published task contains only objects and leaf values, so array-aware binding is open.

Frequently Asked Questions (FAQs)

What is a JSON evidence binding test?

It is an evaluation in which each supplied source record has an independent identity, and the evaluator checks that the record's complete content appears exactly once at the location the grouping rules assign. It extends value-level checks by adding the subject and the path.

How do you verify that a value belongs to the right record?

Compare against an oracle computed separately from the model, keyed by fixture record identity, and compare complete typed records, not only leaves. When values repeat, a leaf match is insufficient, and records with identical complete tuples cannot support unique attribution; label them untraceable.

How do you verify integrity between a source and a record?

Hash the frozen inputs and store the source locator and evidence for every expected binding. Rerun against the same hash to reproduce a result. Canonical serialization such as RFC 8785 helps make the hash repeatable. Integrity of the transformation is separate from the truth of the source.

How does this relate to fusing private and public data?

Fusion methods choose among conflicting claims, and they assume each claim is already bound to the right subject. A binding test protects that assumption at the point where a model reorganizes records; it does not decide which source is more reliable.

Conclusion

A correct string in a valid document can still be asserted about the wrong subject. The October 2026 revision of the JSONPlacement paper makes this concrete by supplying every value and rule, so the remaining failure is organization, and by reporting that models can preserve most values while binding far fewer to the right path. Its separation of record-assignment from hierarchy-realization errors, and its matched input-order and output-path interventions, give a testable way to diagnose before changing a system.

The ledger protocol turns those ideas into a procedure: a frozen manifest with scoring-only identifiers, per-record expected and actual locations, RFC 6901 pointer fixtures with escaping and arrays, four matched conditions, a deterministic baseline, held-out fixtures and document-level resampling. Untraceable records stay untraceable.

The evidence has firm limits. The results are author-reported, from one research group, in a setting where transfer to inferred assignments is untested. Nothing in this report was run, and the proposed ledger has no measured performance. Where the mapping is fully specified, use conventional code. Where a model must organize, test binding, not only validity, and let the dominant error type choose the change.

Source links

Links cited in this article, in order of first appearance. Full addresses are provided for readers using extracted text.

1. https://arxiv.org/html/2608.25358v2
2. https://platform.claude.com/docs/en/build-with-claude/structured-outputs
3. https://developers.openai.com/api/docs/guides/structured-outputs
4. https://arxiv.org/pdf/2604.25359
5. https://datatracker.ietf.org/doc/html/rfc6901
6. https://json-schema.org/overview/what-is-jsonschema
7. https://json-schema.org/understanding-json-schema/reference/enum
8. https://arxiv.org/html/2501.10868v3
9. https://github.com/json-schema-org/JSON-Schema-Test-Suite
10. https://aclanthology.org/2026.surgellm-1.22.pdf
11. https://arxiv.org/pdf/2510.15955
12. https://datatracker.ietf.org/doc/html/rfc8785
13. https://datatracker.ietf.org/doc/html/rfc8259
14. https://datatracker.ietf.org/doc/html/rfc6902
15. https://aclanthology.org/2024.tacl-1.9.pdf
16. https://scikit-learn.org/stable/modules/cross_validation.html
17. https://jqlang.github.io/jq/manual/v1.7
18. https://docs.python.org/3/library/itertools.html
19. https://docs.python.org/3/library/stdtypes.html
20. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.groupby.html
21. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/groupBy
22. https://tc39.es/ecma262/multipage/ordinary-and-exotic-objects-behaviours.html
23. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/groupBy
24. https://www.postgresql.org/docs/current/functions-aggregate.html
25. https://www.postgresql.org/docs/current/datatype-json.html
26. https://arxiv.org/pdf/2411.00640
27. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.bootstrap.html
28. https://mlanthology.org/neco/1998/dietterich1998neco-approximate
29. https://www.statsmodels.org/stable/generated/statsmodels.stats.contingency_tables.mcnemar.html
30. https://www.w3.org/TR/prov-dm
31. https://www.w3.org/TR/prov-o
32. https://openlineage.io/docs
33. https://github.com/OpenLineage/OpenLineage/blob/main/spec/OpenLineage.md
34. https://openlineage.io/docs/spec/facets/dataset-facets/column_lineage_facet
35. https://homepages.inf.ed.ac.uk/jcheney/publications/provdbsurvey.pdf
36. https://datahub.com/blog/data-lineage-vs-data-provenance
37. https://repository.upenn.edu/cis_papers/210

38. https://arxiv.org/abs/1503.00302
39. https://research.google/pubs/pub45634
40. https://lunadong.com/publication/kv_kdd.pdf

THE PUBLISHER

About Reliable Context

Reliable Context is a system of record for organizations that fuse private and public information and must defend every value they act on.

The guarantees

One canonical record per real thing: duplicates resolve by rule, ambiguity is shown rather than hidden, and related organizations relate without ever being merged. A verbatim, cited source and an observation date sit behind every value. Unknowns are declared as not found, blocked or not applicable with a reason, never left as an empty cell. Any past state of the record is reproducible, together with what changed since.

Where it runs

It runs inside the customer's perimeter, one database per organization. Permitted sources, model routing, reader clearance and retention are set as the customer's own policies, and every value is exportable with its chain of custody for an auditor or regulator.

What it is not

It is not a vector database or a RAG framework, not a data catalog, MDM or data-quality suite, not a CRM or data-enrichment vendor, and not a chatbot or AI assistant. Where a reader needs one of those, the honest answer is to say so and point them there.

Visit

See Reliable Context.

Reliable Context is in production use in two sectors; more than six thousand of its identity decisions were made by rule and none has been reversed. Articles published here explain general practice, cite their primary and academic sources, and name the jurisdiction of any regulatory rule they describe. They are not legal, regulatory or compliance advice for a specific organization, and requirements change.

Website: https://reliablecontext.com

This article is educational. Verify consequential medical, legal, financial, regulatory and technical decisions with appropriate professionals and the cited primary sources. Company services are described separately from the article's research findings.