YESTERDAY'S FACTS, TODAY'S VIOLATIONS: SCHEMA TIGHTENING IN LLM-WRITTEN KNOWLEDGE GRAPHS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents increasingly write into the knowledge graphs that later ground their answers, and validating each write against a schema is already production practice. In every such system the schema is *monotonic*: it is frozen or only extended, so data that once passed validation keeps passing. The constraints users ask for run the other way. Writing $\mathcal{V}(S)$ for the set of graphs that validate against shapes S, "every contract must carry at least one document" is a *tightening*, a change with $\mathcal{V}(S') \subsetneq \mathcal{V}(S)$. It retroactively invalidates data that has not changed, along with answers already delivered from that data. Incremental SHACL validation does not cover this case, because it assumes that the data changes while the shapes stay fixed. We treat a constraint change as a first-class, versioned event. After a policy gate and an impact preview, the system revalidates the whole graph and emits each newly broken fact as a conflict event that records the shape version that judged it and the agent behavior that wrote the value; the value is then repaired, or the event is deferred with the violated shape as its reason. No existing benchmark specifies what breaks under a tightening or what the correct repair is, so we build one by inversion: we delete known values, inject tightenings, and diff the validation reports, and each deleted value becomes the gold repair. On QUDT and LUBM, a single tightening takes the report from 120 to $81{,}705$ violations without touching the data. Our full loop clears 98.0% of the violations it attempts but repairs only 75.0% correctly, and the gap between the two *widens* from 13.3 to 23.0 pp as repair prompts get stronger. A system that reports validity alone therefore overstates its repair rate and silently corrupts the graph it is meant to protect.

1 INTRODUCTION

LLM agents increasingly read documents, write facts into a knowledge graph, and later answer from that graph. Validating every write against a schema is already standard practice at scale: one production system ingests 150–250K facts per day this way at a reported precision of 98.8% (ODKE+). These systems share an unstated premise: the schema does not change. They either freeze it or only extend it, and at least one does so by design, making its extensions monotonic so that "prior definitions are never altered" (ESC). If the schema only grows, data already admitted can never become invalid, and revalidation is never needed because the case that would require it is excluded by design.

Most constraints that users request, however, tighten the schema. "Every contract must carry at least one document" makes the standard stricter while adding no vocabulary, and an extend-only system cannot express it. A tightening turns facts that passed yesterday into violations today without a single line of data changing, and it invalidates any answer already delivered on the basis of those facts. Existing tools do not handle this direction. Incremental SHACL validation assumes that the data changes while the shapes stay fixed (UpSHACL; SHACL under Updates; Re-SHACL), and widely used implementations handle a shape update by reloading the shapes graph and revalidating from scratch (RDF4J).

A systematic review of LLM-based ontology engineering finds that only one of 49 task-level studies (2.0%) addresses maintenance, and none mentions SHACL (Li et al., 2026); a recent workshop lists ontology evolution and versioning among its topics but accepted no paper on them (LLM4KGOE).

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

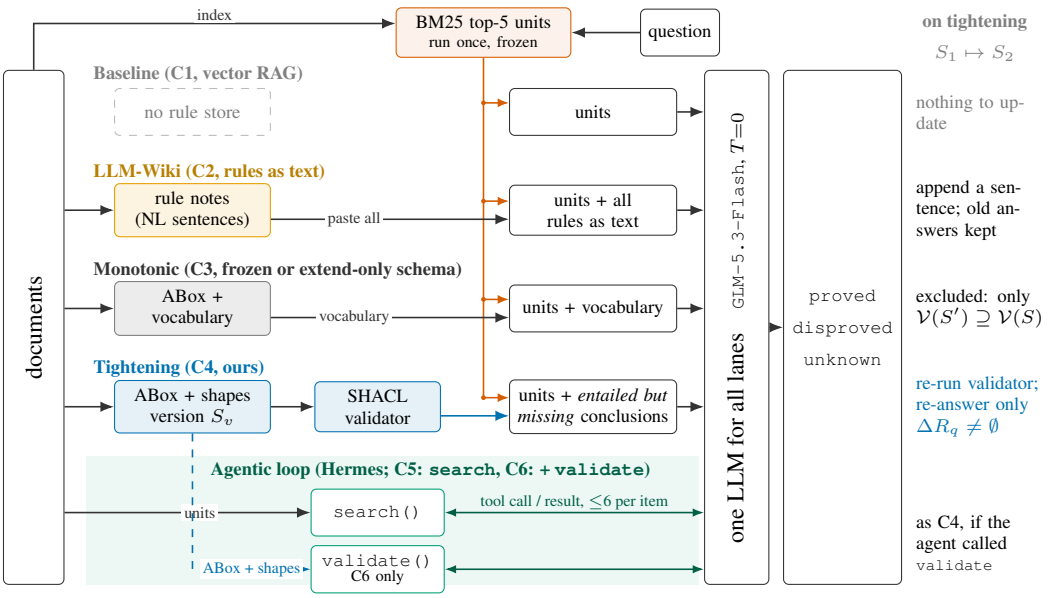

Figure 1: The compared systems as flows. Documents are indexed once, and the frozen BM25 top-5 units (Robertson & Zaragoza, 2009) reach every prompt through the shared bus; the lanes differ in what ingestion builds and what else reaches the prompt. C2 stands for systems that accumulate natural-language rules (LLM-Wiki; ACE), C3 for monotonic schemas that are frozen or only extended (ESC). Only C4 executes the rules before the LLM reads them. In the agentic loop, the agent retrieves for itself, and C5 and C6 differ by the `validate()` tool alone. The right column is what each system does when the shapes are tightened.

The work closest to ours lists "robustness under ontology change" as its own future work (Zhou et al., 2026).

We address this gap by treating a constraint change as a first-class, versioned event. Our contributions are:

- A write-path loop that passes each proposed patch through a policy gate and an impact preview, revalidates the whole graph on approval, and emits every newly broken fact as a conflict event recording which version of the shapes judged it and which agent behavior wrote the value. Both fields already exist in the dataset extension of the SHACL validation report (Dao et al., 2026). The receiver repairs the value or defers to a human, citing the violated constraint. The same constraints are executed at query time to supply the answering LLM with conclusions that the rules entail but the graph lacks.
- A benchmark built by inversion. The closest prior benchmark covers only ABox violations and leaves schema change out of scope (VIO); we borrow its methodology and reverse its direction, deleting a known value, injecting a constraint that targets it, and diffing the validation reports, so that the deleted value is the gold repair. Items whose deleted slot admits several correct values are counted separately and excluded from the accuracy denominator.
- A measurement protocol that separates a violation disappearing (validity) from a violation being repaired correctly (accuracy). A similar gap has been reported for property-graph repair (Terdalkar et al., 2025); we show it in our setting and track how it changes as repair methods improve.

On public knowledge graphs, a single tightening takes the validation report from 120 to 81,705 violations without touching the data. Stacking the repair prompt in four stages raises validity from 71.1% to 98.0% but accuracy only from 57.8% to 75.0%, so the gap widens from 13.3 to 23.0 percentage points as methods get stronger. A system that reports validity alone thus overstates its repair rate and silently corrupts the graph it is meant to protect. On the read path, executing constraints helps question answering only when the rules are present in the data and retrieval is

incomplete, and it does not help on a factual corpus where most reported violations are false alarms; on a corpus with rules in its documents, constraint reach is 80.1% and executing the rules improves accuracy by $+0.144$ over stating the same rules in natural language.

2 Related Work

Validation-feedback repair. Several systems return validator output to an LLM for repair: explaining SHACL violations in natural language (xpSHACL), supplying allowed values (VeriHarness), repairing over knowledge graphs lifted from logs and policy documents (OntoLogX; PolicyKG), or translating textual constraints into SHACL for validation (AML-SHACL). Repair performance has also been evaluated systematically (VIO; Terdalkar et al., 2025). In all of these the shapes are given and fixed, and only the data is repaired; the closest evaluation explicitly places schema change out of scope (VIO). We therefore use this line as baselines (Section 3.2) and adopt the finding of Terdalkar et al. (2025) that format, validity, and accuracy must be measured separately, although their setting (property graphs with Cypher constraints and 7B–14B open models) differs from ours.

Schema change in knowledge graph construction. Industrial extraction pipelines either leave the ontology unchanged, generating ontology fragments only to guide extraction (ODKE+), or extend it monotonically, as in executable schema contracts whose "prior definitions are never altered" (ESC). Research systems point the same way: autonomous ontology evolution filters proposed edits by a consistency check with no SHACL, human, or instance data in the loop (Evo-DKD), competency-question-driven ontology extension is an offline drafting aid (OntoExtend), and taxonomy expansion is concerned with accumulating "noisy, redundant, or hierarchically inconsistent" structure rather than with tightening (ReLTEx); see KGC Survey for a survey and Ontology Enrichment; Event Ontology Expansion; Concept Placement for concept-placement datasets. Because an extend-only schema can never invalidate stored data, retroactive revalidation never arises in this line, and a request such as "every contract must carry at least one document" cannot be expressed in it either.

Natural-language rule accumulation. A separate line accumulates natural-language rules instead of formal constraints, extracting insights from experience, refining manuals and playbooks, or jointly learning rules and policies (ExpeL; AutoManual; Dynamic Cheatsheet; ACE; ReasoningBank; JERP; RPMS; LEAP); rules alone can yield sizeable gains (RPMS). Closest to our work is Error Book (LLM-Wiki), which turns recurring errors into "reusable constraints," injects them into the next compilation prompt, repairs pages already built, and closes an entry once its error stops recurring. Closing, however, removes a line from the prompt; it does not revisit judgments made under that rule, and because rules are applied at index time rather than query time, earlier outputs are never re-examined when the standard changes. The line itself has named the erosion caused by repeated rewriting *context collapse* (ACE), but addresses it by preventing collapse and provides no way to retract a rule. A natural-language rule is easy to add but hard to revert, whereas an executable shape can be removed as easily as it is added.

Constraint enforcement before writing. Zhou et al. (2026) compile an OWL ontology into executable tool interfaces so that agents must pass through them to create or modify the knowledge graph, "enforcing semantic constraints during generation rather than through post-hoc validation." Their approach is closest to ours in shape, but it does not use SHACL and keeps the TBox fixed. They list as future work "systematically evaluating robustness under ontology change, modifying the ontology and re-running the compilation and instantiation pipeline to quantify how executable tool interfaces and prompts adapt to evolving schemas," which is the problem this paper addresses. Appendix B discusses further adjacent work.

3 Method

We first define tightening (Section 3.1), then describe the write path that propagates it through the graph (Section 3.2), the read path that executes the same constraints at query time (Section 3.3), and the benchmark that evaluates both (Section 3.4). Implementation details are in Appendix C.

Table 1: Three kinds of schema change, classified by how the patch moves $\mathcal{V}(S)$.

	What happens	Retroactive violations	Our loop
$\mathcal{V}(S') \subsetneq \mathcal{V}(S)$	Tightening: what was allowed is forbidden	Yes	Subject of this paper
$\mathcal{V}(S') = \mathcal{V}(S)$	Vocabulary extension: allowed set unchanged	No	Auto-approved
$\mathcal{V}(S') \supsetneq \mathcal{V}(S)$	Relaxation: what was forbidden is allowed	No	Rule removal; no rollback

3.1 TIGHTENING

A schema can change by extending the vocabulary (new classes or properties) or by tightening a constraint (new shapes). Both add something, but only the second can invalidate stored facts: a new class has no instances, so even a shape that targets it has nothing to check, whereas a new constraint applies to facts already stored, and some of them may fail it although the data has not changed.

We count retroactive violations per validation result. Each SHACL result is identified by its focus node, result path, constraint component, source shape, and value (Knublauch & Kontokostas, 2017). Writing $R(G, S)$ for the set of results of graph G against shapes graph S, the retroactive violations of a patch $\delta : S \mapsto S'$ are

$$\Delta R(G, \delta) \;=\; R(G, S') \setminus R(G, S), \tag{1}$$

computed after normalizing results on these five fields, since result nodes are blank nodes. The definition counts report rows rather than graph-level conformance: public knowledge graphs usually violate some shapes before any change, so conformance would read *no* before and after however many facts break in between.

Writing $\mathcal{V}(S)$ for the set of graphs that validate against S, a patch falls into one of three cases (Table 1). Only a tightening, $\mathcal{V}(S') \subsetneq \mathcal{V}(S)$, can produce retroactive violations; any patch with $\mathcal{V}(S') \supseteq \mathcal{V}(S)$ has $\Delta R = \emptyset$ by definition. Vocabulary extension is the equality case, since the allowed set does not change, and it is the only change prior monotonic systems admit (ESC). Relaxation is accepted by removing the shape, but repairs already made under the removed rule are not rolled back, because that would require replaying side effects in reverse; we report how many actions the rule triggered instead.

Scope. Violations after a tightening fall into three layers: the value layer (ABox checked by SHACL), the inference layer (SHACL over the inference closure), and the consistency layer (TBox checked by a DL reasoner); see Appendix A. We repair the value layer, where most violations occur and the fix is to supply or change a value. The inference layer is covered up to detection, because repairing a violated inferred triple requires tracing it back to asserted facts and their authors, a composition no prior work provides. The consistency layer is kept out of the automatic path, since removing an axiom is itself a constraint change that must go back through approval.

3.2 WRITE PATH

We treat a constraint change as a first-class, versioned event processed in four stages (Figure 2, Algorithm 1).

Stage 1: proposal and approval. TBox structural changes are written in KGCL (Hegde et al., 2025); shape tightenings, which KGCL cannot express, use our own format. A policy gate rejects unsatisfiable, mis-modeled, or duplicate changes (Appendix C.1). The candidate shape is then run against the current data to preview $|\Delta R(G, \delta)|$, which is compared with an impact threshold. Vocabulary extensions are approved automatically; tightenings require human approval, keeping the expert rather than the LLM as the authority over the schema (Elenchus). Each approved patch issues a new version of the shapes graph.

Algorithm 1 Write path for a constraint patch

Require: graph G partitioned into named graphs by agent behavior; shapes graph S at version v; patch $\delta : S \mapsto S'$; impact threshold θ

Stage 1: proposal and approval

if POLICYGATE(δ) rejects **then return** REJECTED
$n \leftarrow |R(G_T, S') \setminus R(G_T, S)|$ ▷ impact preview on the targets G_T of δ
if δ is not a vocabulary extension **and** ($n > \theta$ **or** human declines) **then**
  return DEFERRED
$v \leftarrow v + 1$; issue S' as shapes-graph version v

Stage 2: retroactive revalidation

$\Delta R \leftarrow R(G, S') \setminus R(G, S)$ ▷ results keyed on five fields; Eq. 1
assert $R(G, S) \setminus R(G, S') = \emptyset$ ▷ a tightening cannot remove violations
for each result $r \in \Delta R$ **do**

  Stage 3: routing back

  $e \leftarrow (r,\ \texttt{sourceShapeGraph} = v,\ \texttt{focusGraph} = \text{PRODUCER}(r))$

  Stage 4: repair or defer

  $x \leftarrow \text{REPAIR}(e)$ ▷ one `INSERT DATA` statement, or DEFER
  if $x \neq$ DEFER **and** $R(G \oplus x, S')$ no longer contains r **then**
    $G \leftarrow G \oplus x$
  else
    send e to the human queue with reason $\texttt{sourceShape}(r)$

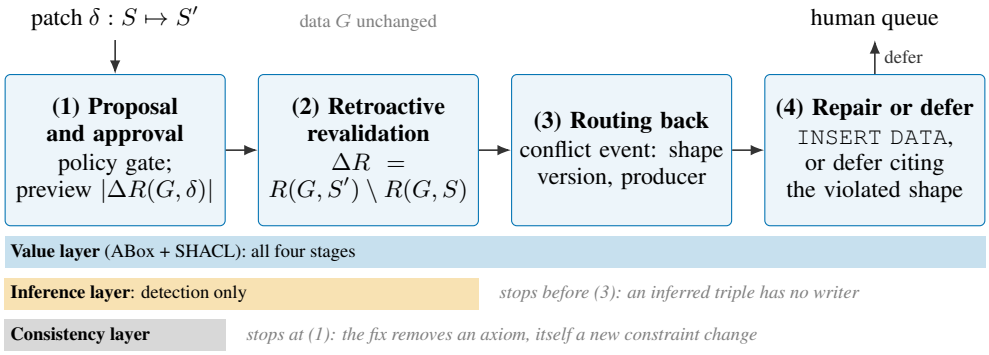

Figure 2: The write-path loop and where it stops in each layer.

Stage 2: retroactive revalidation. Incremental SHACL validation assumes that the data changes while shapes are fixed (UpSHACL; SHACL under Updates; Re-SHACL), and common implementations reload the shapes graph and revalidate from scratch (RDF4J). We therefore compute $R(G, S)$ and $R(G, S')$ and take their difference, restricting candidate nodes to the changed shape's targets with the SHACL target definition `rdf:type/rdfs:subClassOf*` (Knublauch & Kontokostas, 2017). This shortcut misses no node for value and `subClassOf` changes while running about 2,000× faster than a reasoner, but loses recall for `equivalentClass`, property restrictions, and `sameAs` merges, which are routed to the reasoner path (Appendix C.2). We do not materialize inferences before validation: it leaves gold detection recall essentially unchanged at 19–99× the cost (Appendix C.3). Every run checks that $R(G, S) \setminus R(G, S')$ is empty, as a tightening cannot remove violations.

Stage 3: routing back. Each broken fact is emitted as a conflict event that records which version of the shapes judged it and which agent behavior wrote the value. Both fields exist in the RDF dataset extension of the SHACL report, `shds:sourceShapeGraph` and `shds:focusGraph` (Dao et al., 2026; SHACL-DS); our one design decision is to partition named graphs by agent behavior, so the producer appears in the standard report rather than having to be inferred from logs, where the best reported agent-level attribution accuracy is 53.5% (Who&When). Routing has an inherent limit: a `sh:minCount` violation means nobody attached a value, and an action never taken has no producer. Appendix C.4 shows an event and discusses these limits.

Table 2: The four QA conditions. All share the prompt skeleton and the frozen retrieval.

Condition	Rule block	Rule knowledge	Validator run
C1 Vector RAG, no rules	None	Retrieved only	No
C2 Natural-language rules	All rules in natural language	Full	No
C3 Loose schema	Predicate vocabulary only	Retrieved only	No
C4 Tightened schema (ours)	Vocabulary + validator output	Full	Yes

Stage 4: repair or defer. We stack the repair prompt in four stages: a re-prompt with the violation text; adding a natural-language explanation of the violation (xpSHACL); adding the allowed values or, when they cannot be enumerated, the value's datatype and pattern with example values (VeriHarness); and our full loop, which additionally requires a single `INSERT DATA` statement and offers the option to defer to a human queue. A correct format, a disappearing violation, and a correct value are different events: an insertion can satisfy the shape with the wrong value. Following Terdalkar et al. (2025), we therefore report format and validity over attempted tasks and accuracy over attempted tasks with a unique gold answer (Section 3.4), and always report accuracy together with the deferral rate, so that accuracy bought by automating less is visible (Appendix C.5). A deferral states which shape was violated, and we check each stated reason against the validator's output instead of trusting the model's self-report.

3.3 Read Path

The same constraints are executed at query time. Rules are encoded in SHACL-SPARQL so that `Body` $\Rightarrow$ `Head` reads "if Body holds, Head must be in the graph"; a violation is then a conclusion the rules entail but the graph lacks, and the validator's report is added to the prompt next to the retrieved passages. The validator sees only base facts and applies each rule once, leaving chained reasoning to the LLM (Appendix C.6). A negated body atom holds only when the negation is stated as a base fact, since in BoardgameQA a negation that is neither stated nor derived leaves the answer `unknown`. Rules whose body needs a semantic judgment of a value are not compiled, leaving 87.6% of rules as shapes; we trade reach for fewer false alarms, and the remaining source of false alarms is a rule that loses a priority conflict but still fires.

Four conditions differ only in their rule block (Table 2, Figure 1). C2 and C4 hold the same rule knowledge, C2 as natural-language sentences (as in rule-accumulation methods (LLM-Wiki; ACE)) and C4 as executed conclusions, so their difference measures whether executing constraints helps more than stating them. Retrieval (BM25, top five units (Robertson & Zaragoza, 2009)) is run once and shared, and is deliberately lossy: it fetches every gold rule for 772 of 1,000 items (770 of the 997 scored in Table 6), while the validator, which read all rules at ingestion, still holds the missed ones. A constraint can only help where it fires, so we report *reach*, the fraction of items on which the validator reports a violation.

3.4 Benchmark by Inversion

No existing dataset specifies what breaks under a tightening and what the correct repair is; the closest benchmark covers only ABox violations and leaves schema change out of scope (VIO). We borrow its methodology and invert the direction. Starting from a public knowledge graph, we record $R(G, S)$, delete sampled values while logging each (node, path, value), inject tightenings as *new* shapes (modifying an existing shape would change `sh:sourceShape` and make old violations look new), and compute ΔR by Equation 1. Violations in ΔR that join the deletion log on (node, path) become repair tasks, and the deleted value is the gold answer. Deletions are sampled at random, stratified by constraint kind, class, and value count, and only afterwards split into unique and non-unique items; accuracy is scored on unique items only, decided by rules (the value remains, the property is functional, or the answer is derivable by another path) rather than by an LLM. Because the gold is the original value, our accuracy is strictly a restoration rate. Appendix C.7 gives the full procedure.

Table 3: Retroactive violations from a single tightening, counted as results keyed on the five fields of Section 3.1. Varied: shapes graph ($S \to S'$). Fixed: data graph, validation engine. As a cross-check, pySHACL's raw result counts on the same graphs move in the same direction and at the same scale (QUDT $25 \to 80{,}962$; LUBM $14{,}685 \to 14{,}751$).

KG	Tightening shapes	$\lvert R(G,S)\rvert$	$\lvert R(G,S')\rvert$	$\lvert\Delta R\rvert$	$\lvert R(G,S) \setminus R(G,S')\rvert$
QUDT	62	120	81,705	81,585	0
LUBM (univ1)	7	11,282	11,414	132	0

Table 4: Repair of 244 retroactive violations (70 with a unique gold answer). Varied: repair method. Fixed: KG, tightening, evaluated LLM, retry count. Columns have different denominators (Section 3.2); values in %. †: cited, not run by us; property graphs with Cypher constraints and other models, so only the existence of a gap is comparable.

Method	Format ↑	Validity ↑	Accuracy ↑	Re-viol. ↓	Deferral
Re-prompt (baseline)	100.0	71.1	57.8	0.0	0.0
+ Violation explanation (xpSHACL)	98.8	77.1	54.2	0.0	0.0
+ Allowed-value list (VeriHarness)	98.9	95.7	64.4	0.0	0.0
Full loop (ours)	99.3	**98.0**	**75.0**	0.0	38.9
Open LLMs (Terdalkar et al., 2025)†	72–96	42–74	≤ 38	—	—

4 Experiments

We ask what a tightening breaks (Section 4.1), whether the breaks can be repaired (Section 4.2), and whether the same constraints help question answering (Section 4.3); further analyses are in Appendix D.

Setup. We build the repair benchmark from QUDT (QUDT) (200 deletions, 62 tightening shapes) and LUBM univ1 (Guo et al., 2005) (200 deletions, 7 shapes), and evaluate question answering on 2WikiMultiHopQA (Ho et al., 2020), MuSiQue (Trivedi et al., 2022), and BoardgameQA (Kazemi et al., 2023), whose documents contain the rules (Table 13 in Appendix D.1). All experiments use `GLM-5.3-Flash` at temperature 0 and pySHACL 0.40.1 (pySHACL), with one seed; deletion samples are replayed from committed logs.

4.1 What Does a Tightening Break?

A single tightening that touches no data multiplies the report by orders of magnitude (Table 3). QUDT nearly conforms before the change and exceeds eighty thousand violations after it. LUBM already has about eleven thousand violations, so conformance reads *no* both before and after, and the change is visible only when counting rows. The last column is zero for both graphs. The definition in Section 3.1 requires this, so the column checks the implementation and does not constitute a finding.

4.2 Can What Broke Be Repaired?

We apply the four repair stages to the 244 retroactive violations of Section 4.1 (QUDT 112, LUBM 132); all 70 items with a unique gold answer are in QUDT (Table 4). Validity rises from 71.1% to 98.0%, but accuracy only from 57.8% to 75.0%, so the gap widens from 13.3 to 23.0 percentage points and peaks at 31.3 for the allowed-value list (Figure 3). Adding the violation explanation raises validity while lowering accuracy. Reported by validity alone, every stage would look like an improvement. The full loop's 75.0% accuracy comes with 38.9% of tasks deferred to a human, and no automatically applied repair re-triggered the same violation. Denominators and breakdowns by constraint kind are in Appendix D.2.

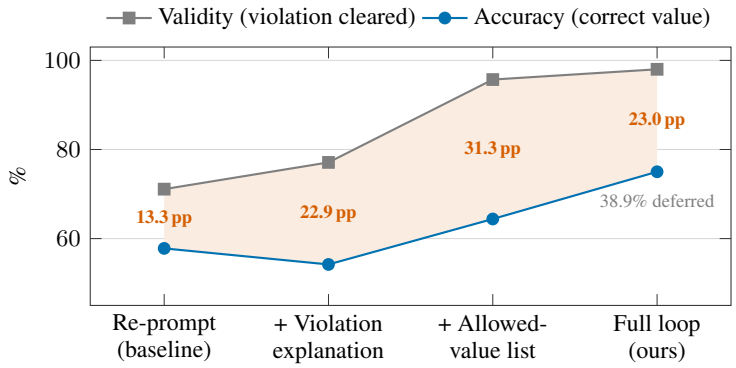

Figure 3: Validity and accuracy diverge as repair methods get stronger.

Table 5: Constraint reach and QA difference by corpus. Varied: corpus and constraint. Fixed: prompt skeleton of the four conditions, evaluated LLM.

Corpus / constraint	**Reach**	**False alarm rate**	Δ(C4−C3)
2WikiMultiHopQA, `maxCount 1`	10.3%	82%	−0.013
MuSiQue	1.3%	—	−0.009
2WikiMultiHopQA, agentic retrieval	0%	—	nothing to measure
BoardgameQA (`KnowledgeLight-depth2`)	**80.1%**	not measured	**+0.212**

4.3 Do Constraints Help Answering?

Reach decides first. Factual corpora state no norms, so the only constraints available are cardinalities, and on 2WikiMultiHopQA 82% of their violations are facts such as co-directors or remarriages. Reach stays at 0–10% and the difference is zero or negative (Table 5). BoardgameQA states its rules in the documents; 87.6% of its rules become shapes, reach is 80.1%, and C4 outperforms C3 by +0.212.

Executing rules beats stating them, most where retrieval fails. C4 outperforms C2, which holds the same rules in natural language, by +0.144 (188 items gained against 44 lost, $p = 2.2 \times 10^{-22}$, McNemar; Table 6). The gain is largest where retrieval missed a gold rule, where C4 reaches 0.683, nearly twice C2, but about half of it comes from items where retrieval fetched every gold rule (0.894 against 0.803). The validator mainly makes answering possible: conditions without rules fall back to `unknown` on 685–688 of 997 items, twice the gold rate, while C4 does so on 402, close to the gold 333. Within retrieval failures, C2 degrades from 0.714 to 0.053 as the validator reports more conclusions, while C4 improves from 0.429 to 0.789 (Appendix D.3).

Reach tracks the difference across configurations. Across five BoardgameQA configurations, reach tracks Δ(C4−C2) (Pearson $r = 0.98$, Spearman $\rho = 0.9$, $n = 5$; Figure 4), from +0.144 at reach 0.801 to −0.046 at reach 0.265; the one inversion is between the two highest-reach configurations. Low reach has a cause: configurations whose rules more often contain attribute conditions lose those rules to the attribute filter (share 32.7% to 84.8%). Filling the conditions with a lookup judge raises reach on the weakest configuration from 0.265 to 0.743 and turns its difference from −0.046 to +0.120 (Appendix D.4). Constraints thus pay off when the rules are in the data and retrieval is incomplete; where most reports are false alarms, as on 2WikiMultiHopQA, they do not.

An agent that retrieves for itself leaves nothing to add. The agentic conditions C5 and C6 start from the question alone, call `search` (BM25 top five) up to six times, and differ only in the `validate()` tool (first 200 items per configuration). In the five depth-2 configurations, repeated search fetches every gold rule on 92–98% of items, and both agents exceed C4 in each of them (C5 0.800–0.905). There C6 never exceeds C5: pooled over the five, C6 alone is right on 44 items and

Table 6: BoardgameQA (`KnowledgeLight-depth2`), 997 items answered by C1–C3; the 26 items C4 left unanswered count as wrong. Random baseline 0.333. Varied: condition. Fixed: items, retrieval, evaluated LLM.

Condition	Acc ↑	Retrieval success (770)	Retrieval failure (227)
C1 Vector RAG, no rules	0.639	0.808	0.066
C2 Natural-language rules	0.701	0.803	0.357
C3 Loose schema	0.634	0.803	0.062
C4 Tightened schema (ours)	**0.846**	**0.894**	**0.683**
Δ(C4−C3)	+0.212	+0.091	+0.621
Δ(C4−C2)	**+0.144**	+0.091	**+0.326**

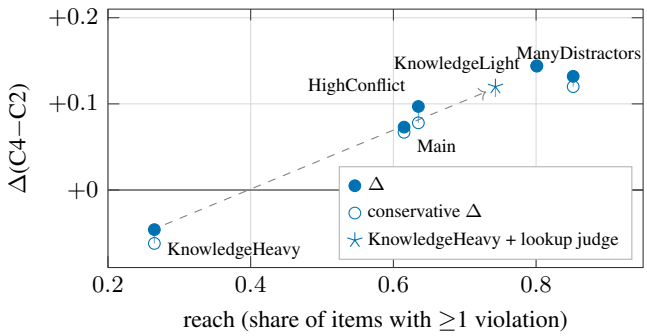

Figure 4: Constraint reach tracks the QA difference Δ(C4−C2) across five BoardgameQA configurations (Spearman $\rho = 0.9$, Pearson $r = 0.98$, $n = 5$); filling attribute conditions with the lookup judge moves the weakest one (star).

C5 alone on 69 ($p = 0.024$), although C6 calls `validate()` on 79–94% of items (Appendix D.3). Once retrieval is nearly complete, the executed rules add nothing the retrieved sentences do not already give, which is the third condition above seen from the other side. Of the two remaining configurations, on Main-depth1 both agents stay within 0.005 of C4 (C5 0.900, C6 0.895, C4 0.895), and Main-depth3, where search fetches every gold rule on only 73% of items, is the one configuration where C6 is ahead (0.774 against 0.739, not significant).

5 Conclusion

Schemas behind LLM-written knowledge graphs have so far been frozen or only extended. We treat tightening a schema as a versioned, first-class event, handled by a loop that previews, approves, revalidates, and routes each retroactive violation, and evaluated on a benchmark built by inversion. A single tightening takes the validation report from 120 to 81,705 violations without touching the data, and stacking the repair prompt raises validity to 98.0% while accuracy reaches only 75.0%, so the gap widens from 13.3 to 23.0 percentage points. A system that reports validity alone can count hallucinated values as successful repairs, so repair must be reported on both rates. On the read path, executed constraints outperform the same rules stated in natural language by +0.144, and only where the atoms the validator can evaluate reach the question; an agent whose own searches already return nearly every rule gains nothing from calling the validator. These results cover the value layer, one evaluated LLM, and one seed; the approval gate and producer tracing are designed but not yet measured (Appendix D.6). The next step is to execute rule priorities inside the validator. Shapes carry no priority, so the validator reports both conclusions of a conflict (Appendix D.4); silencing the losing rule requires knowing whether the winning rule fired, which depends on the attribute conditions in its body, and the attribute judges that restore reach on the weakest configuration are the natural candidates for that decision.

AI USE STATEMENT

We used large language models (LLMs) to aid and polish the writing of this paper, such as rephrasing sentences and improving clarity. The research questions, method, experiments, and results are the authors' own; the authors reviewed every suggested edit and take full responsibility for the text. Separately, LLMs are components of the systems we evaluate (the repair step and the question-answering configurations); that use is part of the method and is described in Sections 3.2 and 4.3.

REPRODUCIBILITY STATEMENT

Every experiment uses one evaluated LLM (`GLM-5.3-Flash`) at temperature 0, pySHACL 0.40.1, and one seed; Table 14 lists all settings. The retroactive-violation benchmark (Section 3.4) is replayed from committed deletion logs rather than regenerated from a random seed, because the traversal order of the target set differs across processes and the same seed does not yield the same sample (Appendix C.7). For QUDT we validate against a copy of the upstream shapes in which the 57 property shapes lacking `sh:path` are filled by three rules, leaving the original file untouched. BM25 retrieval for question answering is run once outside the conditions and frozen, so all non-agent conditions read identical retrieval results. The code, the deletion logs, the patched QUDT shapes, the frozen retrieval results, and the prompts will be provided as anonymized supplementary material.

REFERENCES

Ralph Abboud, İsmail İlkan Ceylan, Thomas Lukasiewicz, and Tommaso Salvatori. BoxE: A box embedding model for knowledge base completion. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

ACE. Agentic context engineering: Evolving contexts for self-improving language models, 2025.

Agent Behavioral Contracts. Agent behavioral contracts, 2026.

AgentSpec. AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents. In *International Conference on Software Engineering (ICSE)*, 2026. arXiv:2503.18666.

AML-SHACL. Automated validation of textual constraints against AutomationML via LLMs and SHACL, 2025.

AutoManual. AutoManual: Constructing instruction manuals by LLM agents via interactive environmental learning, 2024.

Franz Baader and Renata Wassermann. Contractions based on optimal repairs. In *Principles of Knowledge Representation and Reasoning (KR)*, 2024.

Franz Baader et al. Optimal repairs in ontology engineering as pseudo-contractions in belief change. In *ACM/SIGAPP Symposium on Applied Computing (SAC)*, 2023.

Ivana Balažević, Carl Allen, and Timothy Hospedales. Multi-relational poincaré graph embeddings. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Belief Revision Contracts. Preregistered belief revision contracts, 2026.

Co-evolving OWL and SHACL. Co-evolving OWL ontologies and SHACL shapes. In *Knowledge Capture Conference (K-CAP)*, 2025. Original not consulted.

CoAgent. CoAgent: Concurrency control for multi-agent systems, 2026.

Concept Placement. Language-model-based architecture for automatic concept placement in ontologies. *Information*, 2026. doi: 10.3390/info17080766.

Corrective Hints. When corrective hints hurt, 2026.

DAK. Discoverable agent knowledge, 2026.

Dao, Ghislain Atemezing, and Christophe Debruyne. Keeping track of errors: A study of SHACL-DS for RDF dataset validation on the ERA RINF knowledge graph, 2026.

DEMM-Bench. DEMM-Bench, 2026.

Doyle to AGM. From Doyle to AGM: A survey and an implementation roadmap for belief change, 2026.

Dynamic Cheatsheet. Dynamic cheatsheet: Test-time learning with adaptive memory, 2025.

Elenchus. Elenchus: Generating knowledge bases from prover–skeptic dialogues, 2026.

ESC. Executable schema contracts: From automatic ingestion to multi-source retrieval, 2026.

Event Ontology Expansion. Event ontology expansion via LLM-based conceptualization, 2026.

Evo-DKD. Evo-DKD: Autonomous ontology evolution with dual decoders. In *IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2026. arXiv:2507.21438.

ExpeL. ExpeL: LLM agents are experiential learners, 2023.

Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics*, 3(2–3), 2005. VERIFY.

Harshad Hegde et al. A change language for ontologies and knowledge graphs. *Database*, 2025. doi: 10.1093/database/baae133.

Xanh Ho et al. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *International Conference on Computational Linguistics (COLING)*, 2020. VERIFY.

InWriting. Thinking before constraining: A unified decoding framework, 2026.

JERP. Joint learning of experiential rules and policies for LLM agents, 2026.

KARMA. KARMA: Multi-agent LLMs for automated knowledge graph enrichment, 2025.

Tejaswi Kasarla, Max van Spengler, and Pascal Mettes. Balanced hyperbolic embeddings are natural out-of-distribution detectors. *International Journal of Computer Vision*, 2026. arXiv:2506.10146.

Mehran Kazemi et al. BoardgameQA: A dataset for natural language reasoning with contradictory information. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

KGC Survey. LLM-empowered knowledge graph construction: A survey, 2025.

Holger Knublauch and Dimitris Kontokostas. Shapes constraint language (SHACL). W3C Recommendation, `https://www.w3.org/TR/shacl/`, 2017. VERIFY.

LEAP. In-context principle learning from mistakes. In *International Conference on Machine Learning (ICML)*, 2024. arXiv:2402.05403.

Let Me Speak Freely. Let me speak freely? A study on the impact of format restrictions on performance of large language models. In *Proceedings of EMNLP: Industry Track*, 2024. arXiv:2408.02442.

Li, Daniel Garijo, and María Poveda-Villalón. Large language models for ontology engineering: A systematic literature review. *Semantic Web*, 2026. URL `https://journals.sagepub.com/doi/full/10.1177/22104968261465514`.

LLM-Wiki. Retrieval as reasoning: Self-evolving agent-native retrieval via LLM-Wiki, 2026.

LLM4KGOE. LLM4KGOE 2026: Workshop at ESWC 2026. `https://koncordantlab.github.io/LLM4KGOE-ESWC/`, 2026.

Maximilian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. VERIFY.

No Rule is Forever. No rule is forever. In *IEEE International Conference on Data Engineering (ICDE)*, 2025.

Nugget Optimization. Iterate until retrieved: Factual nugget optimization for discoverable continual corrections in agentic RAG, 2026.

ODKE+. ODKE+: Ontology-guided open-domain knowledge extraction with LLMs, 2025.

OntoExtend. OntoExtend: CQ-driven RAG for ontology extension, 2026.

OntoLogX. OntoLogX, 2025.

Ontology Enrichment. Ontology enrichment from texts: A biomedical dataset for concept discovery and placement, 2023.

PolicyKG. PolicyKG, 2026.

María Poveda-Villalón and Asunción Gómez-Pérez. OOPS! (OntOlogy pitfall scanner!). *International Journal on Semantic Web and Information Systems*, 2014.

pySHACL. pySHACL 0.40.1. `https://github.com/RDFLib/pySHACL`, 2026. VERIFY; add access date.

QUDT. QUDT — quantities, units, dimensions and data types ontologies. `https://qudt.org/`, 2026. VERIFY; add access date.

RDF4J. RDF4J ShaclSail documentation. `https://rdf4j.org/documentation/programming/shacl/`, 2026.

Re-SHACL. Efficient validation of SHACL shapes with reasoning. *Proceedings of the VLDB Endowment*, 17, 2024. URL `https://www.vldb.org/pvldb/vol17/p3589-acosta.pdf`.

ReasoningBank. ReasoningBank: Enabling agents to learn from experience. Google Research blog, 2025.

ReLTEx. ReLTEx: Reliable LLM-based taxonomy expansion, 2026.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4), 2009. VERIFY.

RPMS. RPMS: Enhancing LLM-based embodied planning through rule-augmented memory synergy, 2026.

SHACL 1.2 Node Expressions. SHACL 1.2 node expressions. W3C First Public Working Draft, 2026.

SHACL 1.2 Overview. SHACL 1.2 overview. W3C Data Shapes Working Group, Editor's Draft, 2026.

SHACL 1.2 Rules. SHACL 1.2 rules. W3C Working Draft, 2026.

SHACL-DS. From RDF graph validation to RDF dataset validation with SHACL-DS, 2025.

SHACL under Updates. SHACL validation under graph updates. In *International Semantic Web Conference (ISWC)*, 2025. arXiv:2508.00137.

Structure Snowballing. From hallucination to structure snowballing: The alignment tax of constrained decoding in LLM reflection, 2026.

Hrishikesh Terdalkar, Angela Bonifati, and Andrea Mauri. Graph repairs with large language models: An empirical study, 2025.

Harsh Trivedi et al. MuSiQue: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 2022. VERIFY.

UpSHACL. UpSHACL. In *International Semantic Web Conference (ISWC)*, 2025. URL https://github.com/DE-TUM/UpSHACL.

VeriHarness. VeriHarness, 2026.

VIO. Systematic evaluation of knowledge graph repair with large language models, 2025.

Who&When. Which agent causes task failures and when? In *International Conference on Machine Learning (ICML)*, 2025. arXiv:2505.00212.

xpSHACL. xpSHACL: Explainable SHACL validation, 2025.

Zep. Zep: A temporal knowledge graph architecture for agent memory, 2025.

Zhou et al. Ontology-to-tools compilation for executable semantic constraint enforcement in LLM agents, 2026.

ZodiacEdge. ZodiacEdge, 2023.

A THREE LAYERS OF VIOLATIONS

Table 7 compares the three layers of violations discussed in Section 3.1. This paper repairs the value layer, covers the inference layer up to detection, and keeps the consistency layer out of the automatic path.

Table 7: Three layers of violations. This paper repairs the value layer.

Layer	Validator	Nature of violation	Repair action
Value (ABox + SHACL)	SHACL	This value of this node is wrong	Change the value
Inference (TBox+ABox + SHACL)	SHACL over the inference closure	A triple nobody wrote violates	Trace back which asserted fact to fix
Consistency (TBox + OWL DL)	DL reasoner	A class is unsatisfiable	Remove an axiom

B ADDITIONAL RELATED WORK

Agent-side constraints and memory. Several lines touch parts of our problem without addressing retroactive revalidation. One line validates corrections by whether they are retrievable rather than by constraint satisfaction (Nugget Optimization). Runtime enforcement frameworks write agent rules as (trigger, predicate, enforcement) tuples (AgentSpec). Multi-agent knowledge graph enrichment can assign conflicts to a dedicated resolving agent instead of returning them to the agent that caused them (KARMA). Temporal agent memory marks contradicted facts with an invalidation time rather than deleting them (Zep); the mark attaches to the fact, not to the standard that judged it. Other work declares what an agent can prove or pre-registers behavior and belief revision as contracts (DAK; Agent Behavioral Contracts; Belief Revision Contracts; CoAgent).

Relaxation and contraction. Relaxing constraints has a long theoretical lineage: contraction in belief change, with work connecting optimal repairs to contraction now reaching EL ontologies (Baader & Wassermann, 2024; Baader et al., 2023). Incremental maintenance of inference materialization under rule-set change also exists for RDF (ZodiacEdge; No Rule is Forever). Implementing this lineage, however, is still at the roadmap stage with neither algorithm nor code (Doyle to AGM), and its targets (TBox axiom removal or inference rules) lie in a different layer from shape removal.

SHACL 1.2. The SHACL 1.2 drafts on rules and node expressions (SHACL 1.2 Rules; SHACL 1.2 Node Expressions) may blur the boundary between the value and inference layers. The overview, however, states that validation under OWL entailment remains out of scope (SHACL 1.2 Overview), and all of these documents are Working Drafts, so we do not rely on them.

C METHOD DETAILS

C.1 PATCH FORMATS AND POLICY GATE

Patches are written in two formats. TBox structural changes (adding and deprecating classes and properties, moving parents) are written in KGCL, which already provides a standard data model, a human-readable controlled natural language, and `diff`/`apply` implementations (Hegde et al., 2025). Shape tightenings, however, cannot be expressed in KGCL. An exhaustive check of the 74 change classes in its official schema found no predicate for shapes, SHACL, or cardinality constraints. We therefore write only constraint changes in our own format. KGCL was designed for growing ontologies, and the missing standard coincides with the research gap described in Section 2.

The rules of the policy gate are fixed in advance and written as (trigger, predicate, enforcement) 3-tuples (AgentSpec); three of them reject a patch unconditionally (Table 8).

Table 8: Unconditional rejection clauses of the policy gate.

Rejection clause	What it blocks	Cost
Root unsatisfiable class	A logically unsatisfiable TBox	One DL reasoner call
Critical level of an ontology-pitfall validator (Poveda-Villalón & Gómez-Pérez, 2014)	Passes the reasoner but mis-modeled	One HTTP call
Duplicate class creation (ReLTEx)	TBox bloat from repeated extension	Embedding similarity

Table 9: Recall and latency of the target-definition shortcut versus the reasoner path, by kind of TBox change.

TBox change	Recall (shortcut)	Missed nodes	Shortcut	Reasoner
Values only	1.0000	0	5.2 ms	10.7 s
`subClassOf` add/move	1.0000	0	6.8 ms	11.2 s
`equivalentClass`	0.7039	911	5.3 ms	12.6 s
Property restriction	0.6199	1,328	5.6 ms	14.5 s
`sameAs` merge	0.9559	100	5.8 ms	11.0 s

C.2 Candidate-Narrowing Shortcut

Table 9 measures for which kinds of TBox change the target-definition shortcut of Section 3.2 is safe.

The table also defines the policy's switching condition: only patches that touch the last three constructs take the reasoner path. The last two columns show why the reasoner is not used for every patch, since it costs roughly 2,000× the latency.[1] These latencies were measured on a 20,000-triple sample.

C.3 Inference Materialization Before Validation

We decide by measurement whether to materialize inferences before validation. One strand of the literature holds that without inference one misses constraints that hold only after OWL reasoning (Co-evolving OWL and SHACL); another holds that full materialization produces "erroneous or redundant validation results" (Re-SHACL). The two claims are compatible, since the first concerns recall and the second precision, and a single number would hide both effects, so we report them in separate rows. We use two public knowledge graphs of different character: a synthetic university domain (Guo et al., 2005) and a units and quantities ontology (QUDT) (Table 10).

The two knowledge graphs behave in opposite ways. On LUBM, inference dissolves 11,279 violations and creates none; on QUDT, it dissolves 3 and creates 311. Each strand of the literature therefore describes one of the two graphs. Gold detection recall, however, is essentially unchanged, as inference adds no gold detections and slightly lowers them on QUDT. With recall unaffected, the decision rests on cost and side effects, and we do not materialize inferences for this task.[2]

C.4 Conflict Events and Producer Tracing

A standard SHACL report provides only `sh:focusNode` and `sh:sourceShape`. Named graphs are merged before validation, which, in the words of Dao et al. (2026), "obscures triple provenance." The RDF dataset validation extension adds two predicates: the data graph the focus node came from (`shds:focusGraph`) and the shapes graph that raised the error

[1] The recall comparison is valid because both settings see the same sample, but absolute latency depends on scale.

[2] Only the values we deleted have known gold answers, so we treat the other new violations as unverified rather than as false violations.

Table 10: Effect of materializing inferences before validation.

	LUBM-1	QUDT
Violations dissolved by inference	11,279	3
Violations created by inference	0	311
Detection recall (no inference → full)	0.66 → 0.66	0.479 → 0.469
Cost (no inference → full)	4.1 s → 79.9 s	6.6 s → 654.9 s

	Field	Example value	Standard report	Conflict event
key for ΔR	`sh:focusNode`	`:qty_42`	✓	✓
	`sh:resultPath`	`qudt:numericValue`	✓	✓
	`sh:sourceConstraintComponent`	`sh:DatatypeConstraintComponent`	✓	✓
	`sh:sourceShape`	`:NumericValueShape`	✓	✓
	`sh:value`	`"3.5"`	✓	✓
	`shds:focusGraph`	`<graph/behavior/extract-units>` *producer: the agent behavior that wrote the value*	—	✓
	`shds:sourceShapeGraph`	`<shapes/v2>` *shape version issued at approval, stage (1)*	—	✓

Figure 5: A conflict event versus a standard validation result.

(`shds:sourceShapeGraph`) (Dao et al., 2026; SHACL-DS). `shds:focusGraph` identifies the producer and `shds:sourceShapeGraph` identifies the version of the standard issued in Stage 1. The extension adds no overhead at scale: on 33.6M triples and 458 shapes, SHACL-DS validation reportedly takes 565.8 s against 590.9 s for standard SHACL (Dao et al., 2026). Figure 5 shows the fields of one event next to a standard report. The validation engine we use does not implement this extension (pySHACL); we fill in the same information by looking up the named graph containing each focus node.

Named graphs are an RDF standard and both predicates belong to the validation extension, so our only design decision is the partitioning criterion. We partition named graphs by agent behavior, which places the behavior that produced a violation inside the standard report without a separate event log. The alternative, inferring the producer from logs, is unreliable. The best reported accuracy at identifying the responsible agent from multi-agent logs is 53.5%, and 14.2% at the step level (Who&When), while systems overclaim that logs alone suffice to answer governance questions at a rate of 75% (DEMM-Bench).

Two structural limitations remain. The first concerns a node touched by several behaviors: `focusGraph` points to *the graph where the node appears* rather than *the behavior that wrote the value*, so the true cause may lie in another graph. Finer partitioning reduces this problem. The second is inherent to the constraint. An `sh:minCount` violation means *nobody attached a value*, and an action never taken has no producer in an event log or anywhere else. The same property prevents this constraint from being scored automatically (Section 3.4).

Producer tracing is limited to the value layer. In the inference layer, `focusGraph` points to the graph holding the inferred triple, not to the owner of the responsible axiom. Tracing there would take two steps, computing a justification for the responsible axioms and then finding the behavior that wrote them, and no prior work composes the two. We therefore mark this cell as not fillable in principle. Tracing also covers only values written by a behavior; triples created at query time and values entered directly by humans have no producer.

C.5 Repair Design and Metric Denominators

The write path and the repair path constrain the output format at different points. Enforcing a format over the whole generation degrades reasoning (Let Me Speak Freely), whereas narrowing it

Table 11: Denominators of the repair metrics.

Metric	Denominator
Deferral rate	All tasks with a response (full-loop abstention + gate hand-off)
Format, validity	Attempted tasks (abstention is not an attempt)
Accuracy	Attempted tasks with a unique gold answer (Section 3.4)
Re-violation rate	Automatically applied tasks

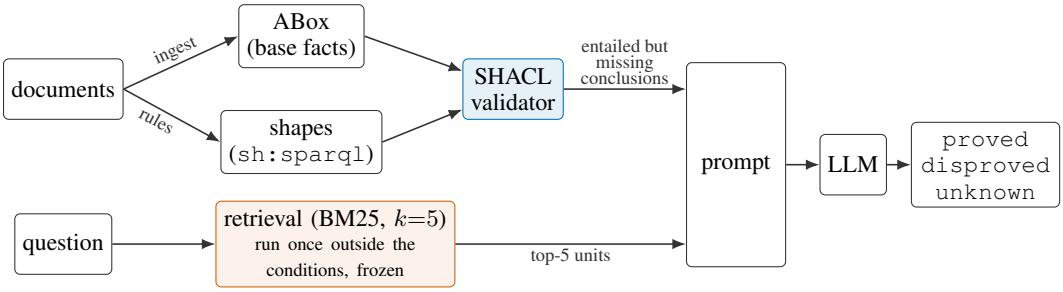

Figure 6: The read path.

only after a stretch of free reasoning improves it (InWriting). Enforcing a format throughout self-correction yields a perfect format while semantic errors remain (Structure Snowballing), and adding a single sentence of logical principle substantially hurts performance (Corrective Hints). Together these results suggest constraining the choice of content, leaving the reasoning process unconstrained, and applying the constraint only after reasoning ends. We therefore measure the write path and the repair path separately, since a combined table would hide the failure mode of self-correction.

The three gates narrow the task set successively, and each metric has its own denominator (Table 11).

Removing gate hand-offs from the validity denominator would be circular: the full loop hands off whatever is invalid, so validity would automatically be 100%. Gate hand-offs count only toward the deferral rate.

Each deferral carries a reason. Instead of ending with *I don't know*, the system states which constraint of which shape was triggered, and this reason links the write path to the read path of Section 3.3. The reason cannot be taken from the evaluated LLM's self-report, because we observed the LLM citing constraint violations as its reason in a condition where the validator was never run. We therefore count a reason only when it matches the violation log that the validator displayed.

C.6 Rule Encoding for the Read Path

Figure 6 shows the read path.

Rules are encoded in SHACL-SPARQL as described in Section 3.3; an example:

```
# Rule1: ~(X, suspect, peafowl) => ~(X, want, crab)
SELECT $this WHERE {
  $this <urn:np:suspect> <...:peafowl> .                    # body (neg.)
  FILTER NOT EXISTS { $this <urn:np:want>   <...:crab>    } # head absent
}
```

Negated facts are stated with separate predicates, because *absent* and *stated to be false* differ and that difference produces the third label. We use a standard SHACL engine rather than a hand-built validator (Knublauch & Kontokostas, 2017; pySHACL). The encoding required three fixes. A negated body atom must match a stated negative fact rather than the absence of the positive one, as explained below. Sharing a `targetClass` across items makes shapes fire across item boundaries. Reporting only the rule name leaves the report useless for items where retrieval missed that rule's sentence, so the report states the conclusion instead.

Table 12: Filter applied before a rule is compiled into a shape.

Filter	What it removes	Why
Attribute condition	Rules whose body has an atom requiring semantic evaluation of a value	Judgment needs arithmetic or external knowledge

In BoardgameQA a negation holds only when it is stated as a fact or derived by a rule; otherwise the answer is `unknown`. Reading a negated body atom $\neg(X, p, o)$ as the absence of (X, p, o) would therefore fire rules whose negated premise is neither stated nor derived, and the validator would report a conclusion as entailed on items whose gold label is `unknown`. A negated atom is thus satisfied only by a stated negative fact in the base ABox. Because the validator applies each rule once, negations that another rule derives are not used; rules that need them stay silent, which costs reach rather than precision. An earlier encoding that read absence as negation was exposed by the agent experiment, where the condition given the validator as a tool scored clearly below the search-only condition; all numbers in this paper use the corrected encoding.

One filter is applied before a rule becomes a shape (Table 12).

With this filter, 87.6% of rules (3,463) become shapes on KnowledgeLight-depth2. We accept the loss in reach, judging that reporting nothing is preferable to reporting a false conclusion. The filter does not make false alarms zero: a rule that loses a priority conflict still fires when its body holds (Appendix D.4).

C.7 Benchmark Procedure

The procedure has seven steps.

```
(1) Validate original -> R(G,S)  existing violations excluded from
                                 scoring (need not conform, Sec. 3.1)
(2) Delete values      creates the answer key:
                       deletion log = {node, path, original value}
(3) Inject tightening  only add new shapes
(4) Revalidate         -> R(G,S')
(5) dR = R(G,S') \ R(G,S)          retroactive violations (5-field key)
(6) dR JOIN deletion log on (node, path)   keep linked ones only
(7) Classify kept items unique / non-unique  -> accuracy denominator
```

The deleted value is the gold answer. Step 3 injects tightenings only as new shapes, because modifying an existing shape changes `sh:sourceShape` and old violations would be mistaken for new ones. For the same reason, real systems that perform retroactive revalidation should also add shapes rather than modify them.[3]

Deletions are chosen by stratified random sampling over constraint kind $\times$ class $\times$ value count, since pure random sampling skews the distribution of constraint kinds and classes and prevents comparison across rows. The corruption depends on the constraint kind: a minimum-cardinality target loses the triple, a datatype target has only its type stripped, and a class target has its `rdf:type` replaced. Sampling only unique items would select easy items and inflate accuracy, so we sample first and split into unique and non-unique items afterwards.

Accuracy is measured only on items with a unique gold answer, meaning that inserting any value other than the deleted one can be judged wrong. A slot for the sensors of a room, which can have many, is unscorable because any sensor makes the violation disappear, whereas a slot where only a number's type was stripped still holds the value and has one answer. Uniqueness is decided by three rules: the value is still present, the property is functional, or the answer is derivable by another path such as an inverse triple. An item that meets none of them is non-unique.

[3]The deletion sample is replayed from a committed log rather than regenerated from a random seed, because the traversal order of the target set differs across processes and the same seed does not yield the same sample. Deletions whose value or subject is a blank node cannot be replayed in principle.

Table 13: Data.

Data	Used in	Size	Notes
QUDT (QUDT)	§4.1, §4.2	200 deletions, 62 tightening shapes	Upstream SHACL with 57 missing `sh:path` filled
LUBM (univ1) (Guo et al., 2005)	§4.1, §4.2	200 deletions, 7 tightening shapes	Synthetic university domain
2WikiMultiHopQA (Ho et al., 2020)	§4.3	1,000 items	Factual corpus
MuSiQue (Trivedi et al., 2022)	§4.3	1,000 items	Factual corpus
BoardgameQA (Kazemi et al., 2023)	§4.3	1,000 items per config, 7 configs	Rules in the documents; 3 labels

Table 14: Settings.

Setting	Value
Evaluated LLM	`GLM-5.3-Flash`, temperature 0
Validation engine	pySHACL 0.40.1 (SHACL-SPARQL)
Retrieval (QA)	BM25, top 5 units, run once outside the conditions and frozen
Repair-loop retries	1, then defer
Seeds	1

We do not use an LLM to judge uniqueness. If an LLM set the scoring criterion, the test-taker and the grader would be the same tool, and results could not be attributed to the evaluated LLM rather than to the grader. A high share of the third rule also calls for caution, since those items are unique only because the inverse triple was not deleted and can be solved by copying it. We count and report the share of items decided by each rule.

Items under minimum-cardinality constraints are mostly non-unique, which leaves their accuracy cells empty. This follows from the same property that gives an untaken action no producer (Section 3.2): some constraints can be neither scored nor traced automatically.

The answer key has one further limitation. The gold is the value in the original knowledge graph, whose correctness we did not verify, so what we call accuracy is strictly the rate of restoring the original value.[4]

D Additional Results

D.1 Data and Settings

Table 13 lists the data and Table 14 the settings.

D.2 Repair: Denominators and Breakdowns

Table 15 gives the denominators behind Table 4.

Did the allowed-value list contain the answer? The effect of the allowed-value list depends on whether the list is small enough to contain the answer. On items whose allowed values enumerate to 40 or fewer, the full loop's accuracy is 85.7% (n=14); on items that do not enumerate, 69.2%

[4]QUDT could be loaded only after we fixed its upstream SHACL (QUDT): 57 of its 500 declared property shapes lacked `sh:path`, and the validator rejected the whole file. We filled all 57 with three rules and kept the original file unchanged, so the QUDT results reflect our patched copy rather than the SHACL distributed upstream.

Table 15: Denominators behind Table 4. Abstained: the model returned no fix or chose to defer; abstentions are not attempts. Only the full loop offers a defer option and routes abstentions and gate hand-offs to the human queue, so the Deferral column of Table 4 is nonzero only there ($92 + 3$ of 244).

Method	Responded	Attempted	Abstained	Auto-applied	Accuracy denom.
Re-prompt	244	142	102	101	45
+ Violation explanation	244	166	78	128	48
+ Allowed-value list	244	187	57	179	45
Full loop	244	152	92	149	40

Table 16: The full loop of Table 4 sliced by KG $\times$ constraint kind (not a new run).

KG	Constraint	#Shapes	#Retro. viol.	Validity ↑ (%)	Accuracy ↑ (%)	n (unique)
LUBM	`sh:class`	3	99	100.0	—	0
LUBM	`sh:datatype`	1	33	100.0	—	0
QUDT	`sh:class`	23	52	100.0	92.9	14
QUDT	`sh:datatype`	21	43	94.7	66.7	18
QUDT	`sh:maxCount`	9	9	100.0	50.0	6
QUDT	`sh:pattern`	4	8	50.0	100.0	2

(n=26). A non-enumerable value domain gives the method no list to supply, which accounts for the smaller gain on those items.

By constraint kind. Table 16 slices the full loop of Table 4 by KG and constraint kind; it is not a new run.

The accuracy cells of the two LUBM rows are empty in principle: several values can fill each deleted slot, so no item has a unique gold answer. Many rows have single-digit unique counts, so we do not compare accuracy across rows directly.

D.3 Question Answering: Prediction Distribution, Report Length, Configurations

Conditions without rules can derive nothing and fall back to `unknown`: C1 and C3 do so on 685–688 of 997 items, twice the gold rate, while C4 does so on 402, close to the gold distribution (Table 17, Figure 7). The main effect of the validator is thus to make an answer possible at all. Over an open-world graph the only safe answer is *unknown*, and executing the rules to derive the conclusion makes it possible to answer *no*. From C3 to C4, 238 items flip from wrong to right and 27 from right to wrong, a ratio of 8.8 to 1.

Restricting to the 227 retrieval-failure items and splitting by the number of conclusions the validator reports gives Table 18 and Figure 8; we split only within failures, since over all items the report length correlates with retrieval failure.

As reports grow longer, C2 falls from 0.714 to 0.053 while C4 rises from 0.429 to 0.789. Stating all rules in natural language helps less as more rules apply, whereas supplying executed conclusions helps more. In the zero-line bucket C4 is below C2: when the validator reports nothing, C4's prompt reduces to C3's.

Table 19 gives the per-configuration numbers behind Figure 4. The conservative score is computed over all 1,000 items and counts every item without an answer as wrong.

The success and failure columns show where the difference comes from. C4 gains over C2 on retrieval failures in four of five configurations and on retrieval successes in the same four; on KnowledgeHeavy it loses on both; at reach 0.265, three quarters of its prompts carry no validator output and differ from C3's prompt only in the vocabulary block.

Table 17: Prediction distribution by condition (997 items; C4 left 26 unanswered).

Condition	proved	disproved	unknown
C1 Vector RAG	153	156	688
C2 Natural-language rules	206	187	604
C3 Loose schema	157	155	685
C4 Tightened schema	295	274	402
Gold	333	331	333

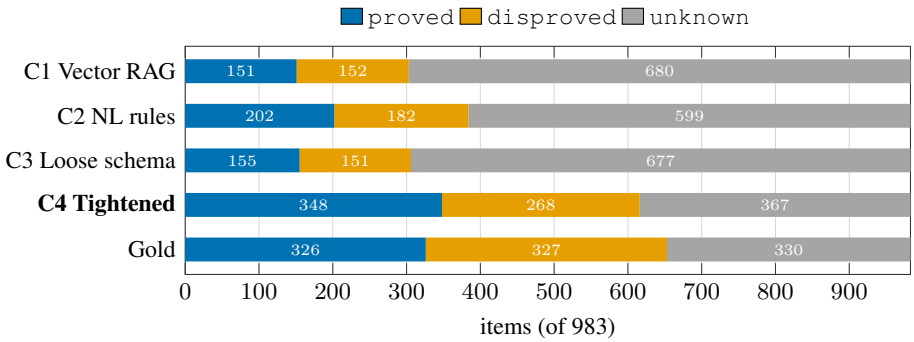

Figure 7: C4 moves the prediction distribution toward gold by reducing `unknown`.

The filter of Section 3.3 discards rules whose body contains an attribute condition, and the share of items involving attribute conditions differs by config (KnowledgeLight 32.7%, Main 65.6%, KnowledgeHeavy 84.8%). The higher that share, the fewer rules the validator holds and the lower the reach. Over the three points, the correlation between attribute-condition share and Δ(C4−C2) is $r = -0.957$. The negative difference on KnowledgeHeavy quantifies the cost of trading reach for fewer false alarms.

Table 20 gives the agentic conditions per configuration. Both agents read most of each item's units through repeated search, so the comparison with the frozen-retrieval conditions C1–C4 is not a comparison at equal retrieval; the comparison that isolates the validator is C6 against C5.

D.4 Question Answering: Chain Length, Attribute Conditions, Rule Conflicts

Varying chain length. Table 18 splits by violation line count, not by reasoning-chain length. Table 21 varies chain length directly via depths 1, 2, and 3 of the Main family.

Neither difference rises with chain length (Figure 9). Δ(C4−C3) is near zero at depth 1 and positive at depths 2 and 3, while Δ(C4−C2) is negative at both ends and positive only in the middle, under both scores. The trend of Table 18 therefore holds only along the validator-report line-count axis and does not generalize to chain length. Depth 3 also has high reach (0.750) yet a negative Δ(C4−C2), so reach alone does not explain this table. Including all seven configs, the rank correlation between reach and Δ(C4−C2) drops to $\rho = 0.61$.

The validator supplies conclusions derived in one step from the base ABox (Section 3.3). On a three-step chain such a conclusion fills only one link, and on items where natural-language rules alone suffice to build the whole chain it may interfere. Table 22 is consistent with this explanation: at depth 3, C4 falls below C2 both where retrieval fetched every gold rule and where it did not, and at depth 1, where retrieval never misses, C4 is below C2 as well.

Filling attribute conditions with embeddings. The negative value on KnowledgeHeavy arises because the validator cannot judge attribute conditions: deciding whether *has a red card* satisfies *has a primary-colored card* requires arithmetic or world knowledge. We cast this judgment as link prediction on a single relation `satisfies(fact, condition)` and solve it with two knowledge graph embeddings: MuRP in hyperbolic space (Balažević et al., 2019) and BoxE, which

Table 18: The 227 retrieval-failure items split by validator-report line count.

Report lines	n	C2 NL rules	C3 Loose schema	C4 Tightened schema
0	7	0.714	0.286	0.429
1	53	0.453	0.113	0.509
2–3	148	0.345	0.041	0.743
4+	19	0.053	0.000	0.789

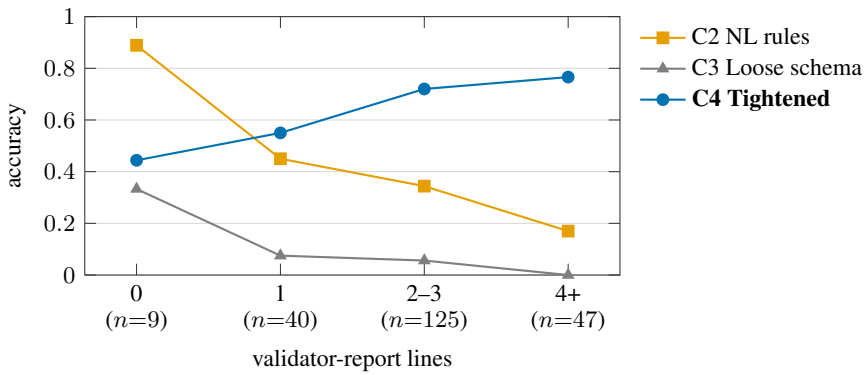

Figure 8: Executing rules helps more as more rules apply.

models relations as boxes (Abboud et al., 2020). The embeddings never emit violations: conditions judged `satisfies`-true are inserted into the ABox as base facts, and the validator makes the judgment.

The training data for the judge was drawn from gold proofs of all items except the test items of the evaluated config, because BoardgameQA's test split differs lexically from its train split: only 109 of 1,322 test conditions appear in train. The threshold was set on a held-out 15% to achieve precision ≥ 0.95.

Lookup only checks whether the pair appeared in training, so its new-pair recall is zero by construction. The embeddings contribute in the new-pair columns, where BoxE recovers a quarter of new pairs at precision 0.976.

Reach rises from 0.265 to 0.743–0.769. The recovered rules, however, include both sides of conflicts that are resolved by priority; shapes carry no priority, so both conclusions are reported, and the number of items with contradictory conclusion pairs grows from 3 to 41–45.

Every judge turns the difference positive, from -0.046 to $+0.102$–$+0.120$ (conservative $+0.101$–$+0.108$). The embeddings recover slightly more reach than lookup, but lookup gives the largest gain in accuracy, so the gain comes from filling the conditions at all and not from the geometry of the judge. The n column differs because C4's prompt, and hence its unanswered set, differs by judge.

When rules conflict. BoardgameQA specifies which of two rules with opposite conclusions wins, via priorities. Items whose gold proof uses a priority cannot have the label `unknown`, so each prediction falls into one of three cases: the correct conclusion; `unknown`, meaning the model never reached the conflict; or the losing rule's conclusion, meaning it saw the conflict but applied the priority backwards. We define two metrics on this basis (Table 26).

C4 reaches the conflict on twice as many items as C2 (46.3% against 22.0%) but applies the priority correctly least often (71.1%), because the validator reports both conclusions of a conflict. Writing the priority into the shapes, so that the losing rule fires only when the winning rule's body fails on the base facts, does not help: it must drop the 350 rules whose winning body the validator cannot evaluate, reach falls from 0.635 to 0.545, and accuracy from 0.639 to 0.602.

Table 19: Reach and difference by config. Varied: config. Fixed: pipeline, evaluated LLM, prompt skeleton. Succ./Fail.: accuracy of C4 / C2 on retrieval success / failure.

Config	Reach	Δ(C4−C2)	Cons.	Δ(C4−C2)	Δ(C4−C3)	Succ. C4 / C2	Fail. C4 / C2
ManyDistractors-depth2	0.852	+0.132		+0.120	+0.217	0.733 / 0.631	0.398 / 0.256
KnowledgeLight-depth2	0.801	+0.144		+0.144	+0.212	0.894 / 0.803	0.683 / 0.357
HighConflict-depth2	0.635	+0.097		+0.078	+0.177	0.815 / 0.767	0.355 / 0.238
Main-depth2	0.615	+0.073		+0.067	+0.137	0.829 / 0.766	0.386 / 0.288
KnowledgeHeavy-depth2	0.265	−0.046		−0.062	+0.026	0.756 / 0.771	0.144 / 0.277

Table 20: Agentic conditions, first 200 items per configuration; unanswered items count as wrong. C5: `search` only; C6: `search` and `validate()`. Gold hit: share of items on which C5's searches returned every gold rule. b/c: items only C6 / only C5 answers correctly (exact McNemar p). Pooled b/c: 44/69 ($p = 0.024$) over the five depth-2 configs, 69/88 ($p = 0.15$) over all seven.

Config	C2	C4	C5	C6	Gold hit (C5)	`validate()` called	b/c (p)
ManyDistractors-depth2	0.490	0.622	0.837	0.832	0.934	79%	11/12 (1.0)
KnowledgeLight-depth2	0.710	0.840	0.905	0.875	0.960	94%	8/14 (0.29)
HighConflict-depth2	0.513	0.614	0.843	0.807	0.919	88%	6/13 (0.17)
Main-depth2	0.620	0.680	0.870	0.835	0.975	92%	7/14 (0.19)
KnowledgeHeavy-depth2	0.615	0.535	0.800	0.780	0.945	91%	12/16 (0.57)
Main-depth1	0.895	0.895	0.900	0.895	1.000	99%	5/6 (1.0)
Main-depth3	0.487	0.452	0.739	0.774	0.729	74%	20/13 (0.30)

D.5 Negative Result: Judging New Concepts with Hyperbolic Embeddings

Stage 1 of Section 3.2 decides, through the policy gate of Appendix C.1, whether a proposed patch may change the schema. We measured an alternative gate that reads hyperbolic embeddings (Nickel & Kiela, 2017; Kasarla et al., 2026). That method rests on two claims: in hyperbolic space the norm of an embedding represents the generality of a concept, and flat space has no such axis.

Neither claim held. On the largest real ontology (envo, 6,472 nodes), Poincaré's ρ(norm, depth) is $+0.089$, and $+0.026$ after controlling for degree. The norm correlates with the log of degree at -0.34 to -0.77, so it tracks how often a node appeared in training rather than how general the concept is. A Euclidean embedding trained with the same loss also has a stronger depth axis on all seven taxonomies.

Hyperbolic space does reconstruct the hierarchy itself better on all seven (MAP), so the hierarchy information is present, but not on the norm axis that the gate was meant to read. We therefore keep the policy gate of Appendix C.1. The embeddings were trained at a smaller scale than in the original paper (dimension 10, 10 negative samples), which limits this conclusion.

D.6 Approval Gate and Producer Tracing

Of the four stages in Section 3.2, Sections 4.1 and 4.2 evaluate Stages 2 and 4. Stage 1 (proposal and approval) and Stage 3 (producer tracing) are designed (Appendices C.1 and C.4) but not measured in this paper; their evaluation requires a stream of proposed patches and a multi-behavior producer log, which we leave to future work.

E Additional Material

E.1 Preliminary Validation of the Hyperbolic-Embedding Gate

Table 28 in Section D.5 reports medians, and Table 29 gives the per-taxonomy values behind them. The candidate gate for Stage 1 (Section 3.2) would read the norm of a Poincaré embedding as the generality of a concept (Nickel & Kiela, 2017; Kasarla et al., 2026), so the premise to test is that the norm tracks depth in the hierarchy. We trained Poincaré embeddings on seven public taxonomies (dimension 10, 300 epochs, 10 negative samples, seed 0) using the direct parent–child

Table 21: Difference by reasoning-chain length. Varied: depth. Fixed: config family (Main), pipeline. Cons.: conservative score.

Depth	Reach	C1	C2	C3	C4	Δ(C4−C2)	Cons. Δ(C4−C2)	Δ(C4−C3)	Cons. Δ(C4−C3)
1	0.366	0.930	0.938	0.913	0.914	−0.024	−0.026	+0.001	+0.008
2	0.615	0.579	0.637	0.573	0.710	+0.073	+0.067	+0.137	+0.135
3	0.750	0.383	0.508	0.381	0.455	−0.053	−0.050	+0.074	+0.074

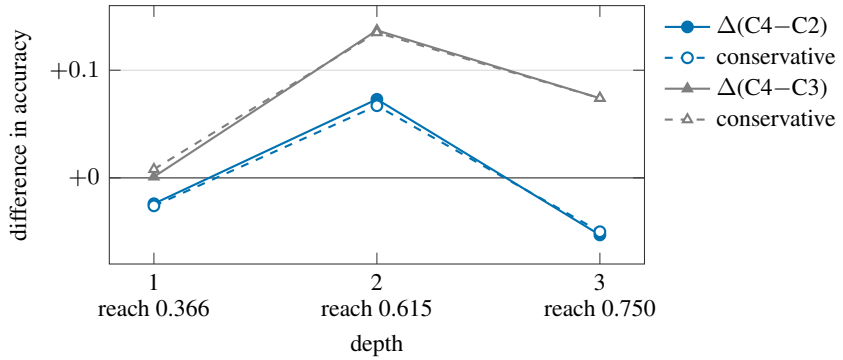

Figure 9: Chain length does not reproduce the line-count trend.

edges. Shallow nodes have many children and therefore appear in many training pairs, so depth and training frequency are correlated by construction; the partial correlation controls for degree to separate the two. As a control we trained a Euclidean embedding with the same loss, dimension and data, changing only the distance function. Distortion is the mean relative error of pairwise distances after fitting one scale factor.

Section D.5 interprets these values. Our setting is smaller than the original work, which trained on the transitive closure of the WordNet noun hierarchy with dimensions 5–200 and 50 negative samples, so the table does not show that Poincaré embeddings fail in general, only that the norm cannot serve as this gate's decision signal. We did not evaluate entailment cones or OOD scores, since placing a new concept in the space would require a text-to-hyperbolic mapping that we do not have.

E.2 Four Deferral-Like Rates

Four quantities in this paper share the idea of handing something over instead of completing it automatically. They differ in who hands over, what is handed over, and the denominator, so they are not interchangeable (Table 30).

Repair deferral is the paired metric of repair accuracy; reporting accuracy without it would hide accuracy gained by automating less. A QA non-answer is not the same as BoardgameQA's `unknown` label, which is a valid answer that can be correct; a non-answer is the absence of any label. Approval deferral is decided before a patch is applied, from the impact preview $|\Delta R(G, \delta)|$, whereas repair deferral is decided per violation after the patch is applied. Human pickup would measure whether a deferral that cites a shape actually helps the person who receives it. It requires human sessions, which we did not run, so we make no claim about it; this is a limit of resources rather than of the design.

Table 22: Retrieval success and failure for Table 21. Items: depth 1, 961 success and none failed; depth 2, 714 / 264; depth 3, 381 / 604.

Depth	Success C2	Success C4	Failure C2	Failure C4
1	0.938	0.914	—	—
2	0.766	0.829	0.288	0.386
3	0.945	0.911	0.232	0.167

Table 23: The `satisfies` judge. 919 pairs (365 positive) from gold proofs of KnowledgeHeavy-depth2 test. New pairs: 581 pairs absent from training (159 positive).

Judge	Precision ↑	Recall ↑	F1 ↑	AUROC ↑	New-pair precision ↑	New-pair recall ↑
Lookup (baseline)	1.000	0.564	0.722	0.884	—	0
MuRP	0.978	0.603	0.746	0.953	0.969	0.195
BoxE	0.983	**0.627**	**0.766**	0.946	0.976	**0.258**

Table 24: Effect of judge-filled conditions on reach (KnowledgeHeavy-depth2).

	No judge	Lookup	MuRP	BoxE
Condition triples inserted into ABox	0	832	855	865
Rules recovered at the attribute filter	0	1,415	1,398	1,398
Reach	0.265	0.743	0.757	**0.769**
Items reporting contradictory conclusion pairs	3	41	45	41

Table 25: QA by judge. Prompts of C1–C3 do not depend on the validator and equal the no-judge row.

Judge	n	C2 NL rules	C4 Tightened schema	Δ(C4−C2)
No judge	968	0.632	0.586	−0.046
Lookup	945	0.640	**0.760**	**+0.120**
MuRP	969	0.633	0.735	+0.102
BoxE	969	0.633	0.740	+0.107

Table 26: Conflict metrics.

Metric	Denominator	Numerator
Conflict detection	Items whose proof uses a priority	Prediction is not `unknown`
Source-pref. adherence	Numerator of conflict detection	Prediction is correct

Table 27: Rule-conflict items. HighConflict-depth2, 164 items whose proof uses a priority; accuracy over all 953 scored items.

Condition	Acc ↑ (all)	Conflict detection ↑ (%)	Source-pref. adherence ↑ (%)
C1 Vector RAG	0.468	9.1	80.0
C2 Natural-language rules	0.542	22.0	75.0
C3 Loose schema	0.462	9.1	80.0
C4 Tightened schema	**0.639**	**46.3**	71.1
C4 + priorities in shapes	0.602	44.2	68.8

Table 28: Correlation between norm and hierarchy depth. Seven public taxonomies, dimension 10, same loss. Partial correlation controls for degree.

Space	Median ρ(norm, depth)	Median partial corr. ↑	MAP range ↑
Poincaré	$+0.317$	$+0.222$	0.62–0.91
Euclidean	$+0.582$	$+0.483$	0.28–0.79

Table 29: Preliminary validation of the hyperbolic-embedding gate. Poincaré embeddings trained on direct edges; the last two rows are medians over the seven taxonomies. †: same loss, dimension and data as the Poincaré rows, with Euclidean distance.

Hierarchy	Nodes	Depth	Avg. branching	Multi-parent (%)	ρ(norm, depth) ↑	ρ(norm, depth | deg) ↑	MAP ↑	Distortion ↓
envo	6,472	34	2.89	19.53	$+0.089$	$+0.026$	0.621	0.330
science_en	452	4	8.61	2.88	$+0.581$	$+0.326$	0.909	0.220
science_wordnet	429	7	3.77	2.80	$+0.317$	$+0.162$	0.848	0.283
science_eurovoc	125	4	4.00	0.00	$+0.633$	$+0.480$	0.888	0.204
food_en	1,556	5	22.67	2.06	$+0.178$	$+0.132$	0.797	0.253
food_wordnet	1,486	8	5.08	3.03	$+0.301$	$+0.222$	0.813	0.301
environment_eurovoc	261	5	4.35	0.38	$+0.573$	$+0.451$	0.890	0.183
Poincaré (median)	—	—	—	—	$+0.317$	$+0.222$	0.848	0.253
Euclidean control (median)†	—	—	—	—	$+0.582$	$+0.483$	0.593	0.199

Table 30: The four deferral-like rates. Only the first two are measured in this paper.

Rate	Definition	Denominator	Where
Repair deferral	The repair loop sends a retroactive violation to the human queue instead of applying a fix (full-loop abstention or gate hand-off)	All repair tasks with a response	Tables 4 and 11
QA non-answer	The QA system returns no label for an item (empty or truncated output); counted as wrong in the conservative count	All items	Section 4.3
Approval deferral	The policy gate sends a proposed patch to the approval queue instead of applying or rejecting it	All proposed patches	Not measured
Human pickup	Among handed-over items, the share that a human actually takes up and resolves	All handed-over items	Not measured