REASONING OVER BROKEN JOINS: HOW LLMS HANDLE INTEGRITY VIOLATIONS IN MULTI-TABLE INPUTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models are increasingly used to query multi-table relational data. The correctness of such queries depends on integrity constraints such as key uniqueness and referential integrity, yet these constraints are frequently violated in practice. On such corrupted inputs, a model should detect the violation and disclose it rather than answer silently. Existing evaluations on such corrupted inputs score only the final output. They therefore cannot distinguish a model that misses a violation from one that detects it but answers without disclosing it. We introduce RELIC, a dataset of approximately 16,000 multi-table instances in which six violation types are injected at controlled severities into integrity-verified synthetic table pairs. We define the *silent failure rate* (SFR) as the fraction of corrupted instances that a model answers without disclosing the violation. We further decompose silent failures by whether the model detects the violation in its reasoning trace. SFR exceeds 84% for all 20 models and 97% for all three frontier models despite clean accuracy of up to 98.3%. Detection increases with model capability and reasoning budget while disclosure does not. Prompt-level interventions elicit disclosure of duplicated primary keys from reasoning models but leave most orphaned foreign keys undisclosed. Model scaling and prompt-level interventions therefore fail to mitigate silent failures consistently across violation types. RELIC enables fine-grained attribution of silent failures to detection or disclosure for each violation type, facilitating the development of LLMs that handle integrity violations reliably. The RELIC dataset and implementation are available at https://anonymous.4open.science/r/RELIC-BDF1.

1 INTRODUCTION

Large language models are increasingly applied to tabular data, with recent work extending from single-table question answering to queries over multiple related tables (Sui et al., 2024; Lu et al., 2025; Pal et al., 2023; Chen et al., 2024; Kang et al., 2025). The correctness of a multi-table query depends on integrity constraints declared by the schema. Primary keys must be unique, every foreign key must reference an existing row, and columns declared non-null must contain no missing values. A database management system is designed to maintain these constraints by rejecting any insertion or update that would violate them. In practice, however, databases still contain violations (Rahm et al., 2000; Abedjan et al., 2016). We examine the development splits of two widely used text-to-SQL benchmarks. In BIRD (Li et al., 2023), 63.6% of the databases contain rows that violate a foreign-key constraint declared in their schema, and 20.0% do so in Spider (Yu et al., 2018). Moreover, the gold answers of 19.8% of BIRD's join queries depend on a broken reference (Appendix A).

A prompt offers even less protection than a database, since tables serialized as CSV or Markdown carry no constraint declarations. Integrity checking therefore falls entirely to the model, with the cell values as the only evidence. Figure 1 illustrates how an LLM responds to tables that violate these constraints. Consider a query that sums the transaction amounts of verified accounts over an ACCOUNTS table and a TRANSACTIONS table. The tables are exported from a database in which every transaction references a verified account, and the true total in that database is $1.8M. During export, however, three accounts are lost, so three rows in TRANSACTIONS become *orphaned*: their account IDs no longer appear in ACCOUNTS. We refer to an input that violates any integrity constraint

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.

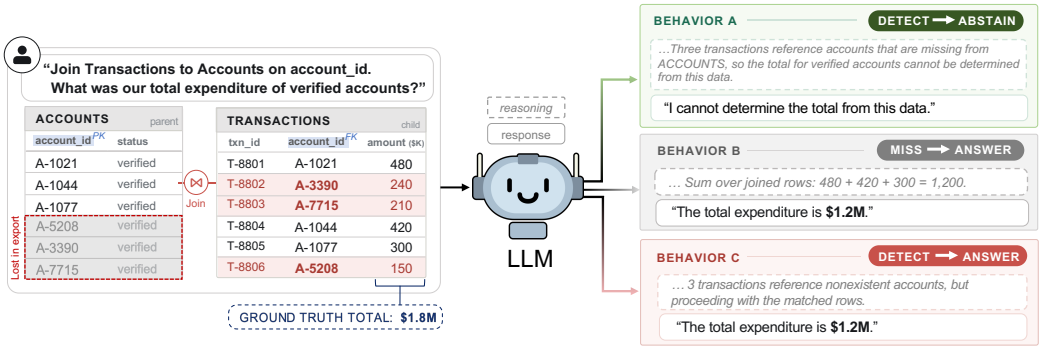


Figure 1: Responses to an integrity violation in multi-table question answering. *account_id* is the primary key of *Accounts*, uniquely identifying each account, and a foreign key in *Transactions*, linking each transaction to its account; the two tables are joined on this key. The tables are exported from a database in which every transaction references a verified account, and the true total in that database is $1.8M. Three accounts are lost in export, so three rows of *Transactions* (red) reference identifiers absent from *Accounts*.

as *corrupted*. Given this corrupted input, a model may detect the violation and abstain (Behavior A). It may instead miss the violation (Behavior B). In that case, the join silently drops the three orphaned rows, and the model returns $1.2M, understating the true total by 33%. A model may also identify the orphaned rows in its reasoning but still return $1.2M without mentioning them (Behavior C). Although the model could have reported these rows alongside its answer, its output is identical to that of B and gives the user no indication that the input is corrupted.

Two recent works evaluate LLMs on tabular inputs with data quality issues, scoring only the final output (Wolff & Hulsebos, 2025; Gu et al., 2026). Both find that models perform poorly and conclude that improved robustness is needed. However, suboptimal performance may arise from three causes. The model may fail to detect the problem, detect it but fail to repair it, or detect it and answer from the flawed data without disclosing it. The first two reflect limited capability; the third does not. As Behavior C illustrates, the failure lies in how the model acts on what it detects. Since Behaviors B and C yield identical outputs, prior evaluations cannot separate the third case from the first two. This paper asks whether models detect integrity violations in the input and how they act on the violations they detect.

To answer these questions, we characterize each response along three axes: *detection* of the violation (measured from the reasoning trace), the *action* taken (abstain, repair, or ignore), and *disclosure* to the user. We define a *silent failure* as a response that answers a corrupted input without disclosing the violation. The *silent failure rate* (SFR) is the fraction of corrupted instances on which a silent failure occurs. If silent failures stem from limited multi-table reasoning capability, two testable predictions follow. First, SFR should decrease as clean accuracy increases. Second, a model that detects the violation should disclose it to the user rather than answer silently. If these predictions do not hold, limited capability alone cannot explain silent failure.

Testing these predictions requires multi-table inputs whose violations have known type, location, and severity. The inputs must also include violations of referential integrity and contain values that cannot be recalled from training data. Benchmarks of imperfect tables (Wolff & Hulsebos, 2025; Gu et al., 2026) do not target referential integrity, and the referential violations in text-to-SQL benchmarks such as BIRD are unannotated and occur in public databases that may appear in training data. We therefore construct RELIC (**Re**lational Evaluation of **L**LM **I**ntegrity **C**hecking), a dataset of approximately 16,000 instances in which six violation types are injected at controlled severities into integrity-verified synthetic table pairs. Each type violates one of three constraints: primary-key uniqueness, referential integrity, or a non-null requirement. The tables are synthesized from the schemas and column statistics of public databases and share no cell values with the originals, so clean accuracy reflects reasoning rather than recall. Every violation is detectable from the serialized tables and the output format permits abstention. A silent failure therefore cannot be attributed to missing information or to the lack of an abstention option.

This paper makes three contributions.

- We construct RELIC, a dataset of approximately 16,000 multi-table instances with violations of core integrity constraints injected at controlled severities. Its synthetic tables separate reasoning from recall, and every violation is detectable from the prompt alone.
- We define the *silent failure rate* (SFR) as the fraction of corrupted instances answered without disclosing the violation. We further decompose silent failures by detection. This distinguishes missed violations from detected but undisclosed ones.
- We find that SFR exceeds 84% for all 20 models despite clean accuracy of up to 98.3%, and that the stage at which these failures arise varies across models and violation types. Detection improves with model capability and reasoning budget whereas disclosure does not, and prompt-level interventions elicit disclosure for only some violation types. Answer accuracy alone cannot reveal this stage. RELIC exposes it for every model and violation type, providing the fine-grained evaluation needed to improve how LLMs handle integrity violations.

2 RELATED WORK

2.1 ROBUSTNESS TO PERTURBED AND CORRUPTED TABLES

The robustness of LLMs to perturbed tabular inputs has been investigated along two primary research directions. The first direction perturbs table structure and formatting while preserving the underlying data values. RobuT (Zhao et al., 2023) shuffles rows and columns and replaces headers with synonyms and abbreviations. ToRR (Ashury-Tahan et al., 2026) varies the serialization format and perturbs table structure, including transposition. Dr.Spider (Chang et al., 2023) replaces schema names with synonyms and abbreviations in text-to-SQL. The second direction introduces errors into the data values, altering or invalidating the correct answer. RADAR (Gu et al., 2026) injects five single-table artifact types: missing data, bad values, outliers, inconsistent formatting, and inconsistent logic. Wolff & Hulsebos (2025) introduce missing values and duplicate rows into the multi-table inputs of TQA-Bench (Qiu et al., 2024). Our violations target the join between two tables — specifically referential integrity, a condition no prior work targets by design.

2.2 ABSTENTION AND SELECTIVE PREDICTION

Abstaining from an answer has been studied as rejection (Chow, 1970) and selective prediction (Geifman & El-Yaniv, 2017; Kamath et al., 2020), which trade coverage for reduced risk. For LLMs, recent work evaluates whether models abstain on questions that are unanswerable or exceed their knowledge (Feng et al., 2024; Wen et al., 2025; Kirichenko et al., 2026). EHRSQL (Lee et al., 2022) extends this evaluation to text-to-SQL, requiring models to abstain from questions that the database cannot answer. In all of these settings, abstention responds to a limitation of the question or of the model's knowledge. In ours, the question is well-posed, but the input tables contain integrity violations that invalidate the query result.

3 THE RELIC DATASET

3.1 PROBLEM DEFINITION

RELIC is a dataset of multi-table question-answering instances designed to evaluate whether LLMs detect and disclose integrity violations in relational data. The dataset is organized into *problems*; each problem yields multiple *instances*, as described in Section 3.4. Each problem in RELIC consists of a parent table, a child table, and a join query relating them. Each row of the parent table T^{par} is identified by a primary key, and each row of the child table T^{chi} references a parent row through a foreign key. The query Q specifies the join key, joins the two tables on it, filters by a categorical column of the parent, and aggregates a numeric column of the child.

The ground-truth answer is specified with respect to a clean pair of tables that preserves data integrity. A clean pair satisfies three conditions: (i) primary keys in the parent table are unique, (ii) every foreign key in the child table refers to a primary key present in the parent table, and (iii) the targeted column contains no missing values. Under these conditions, the answer $A^* = f(Q, T^{\text{par}}, T^{\text{chi}})$ is obtained by executing the operations specified by the query. We produce each violation by applying a controlled perturbation to an otherwise clean pair, thereby yielding a corrupted pair $(\tilde{T}_{\text{par}}, \tilde{T}_{\text{chi}})$ that fails to satisfy at least one of the three conditions above. Because the two serialized tables in

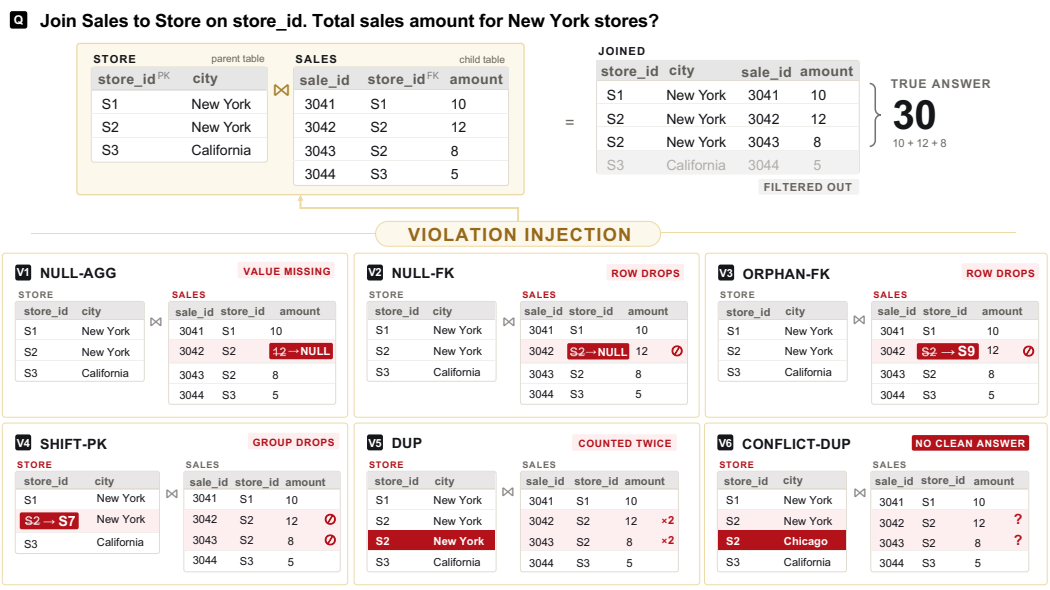


Figure 2: Violation injection. Given a clean parent–child pair and a query Q, we inject one of six violations (V1–V6), each altering the highlighted cells to violate a schema-declared integrity constraint. ⊘ a row silently dropped from the join. **?** marks values whose filter membership is ambiguous because the duplicated key carries conflicting cities.

the prompt are the model's only access to the data, every violation is designed to be detectable from these tables alone.

3.2 Synthetic Table Construction

Directly employing existing table question-answering benchmarks introduces two issues. First, their tables and reference answers may be present in a model's training corpus, so evaluation on them cannot separate reasoning from memorization. Second, their databases may already contain integrity violations, as in BIRD and Spider (Section 1). The effect of an injected violation then cannot be isolated from pre-existing ones, and the reference answers may themselves depend on violated constraints. We address both by drawing the structure of each table from real data and synthesizing values that satisfy every integrity constraint.

The datasets are constructed using database resources from BIRD (Li et al., 2023), Spider (Yu et al., 2018), and TQA-Bench (Qiu et al., 2024). From each database we extract structural information rather than values: which columns a table has and which of them is the primary key, the range and type of values in each column, and which tables reference which others through foreign keys. None of the original cell values are taken. Value generation rules are detailed in Appendix B.1.

Each table pair is then generated from this structural information. We first select a parent table and a child table that references it through a foreign key, and fix the join key between them. We then scale down the row counts of both tables by the same ratio so that the serialized pair fits a target prompt length; scaling the tables independently would change the number of child rows per parent row and thereby bias join results. Cell values are sampled within the range and type extracted for each column, primary keys receive distinct values, and every foreign key references an existing parent row. The resulting pair satisfies the three conditions in Section 3.1 and serves as the clean pair for violation injection (Section 3.3).

3.3 Violation Injection

RELIC contains six violation types. Each corrupted instance contains exactly one violation, which breaks one of the three conditions in Section 3.1, as indicated in parentheses below.

1. **NULL-AGG** (value integrity). A cell in the targeted column is set to NULL. The join and filter are unaffected, but the aggregate omits the missing value.
2. **NULL-FK** (referential integrity). A foreign key in the child table is set to NULL. The child row matches no parent and drops from the join.
3. **ORPHAN-FK** (referential integrity). A foreign key in the child table is replaced with an identifier absent from the parent table. The effect on the answer matches NULL-FK. Unlike a NULL, however, the corrupted key is well-formed and is revealed only by matching it against the parent keys.
4. **SHIFT-PK** (referential integrity). A primary key in the parent table is changed to an unused value. All child rows that referenced it lose their parent and drop from the join together.
5. **DUP** (primary-key uniqueness). A parent row is copied into several identical rows. Each copy joins with the same children, so their values enter the aggregate multiple times.
6. **CONFLICT-DUP** (primary-key uniqueness). A parent row is copied, and the copy receives a different value in the filter column. The tables therefore do not determine which row the query should include.

Under DUP, the corrupted tables still contain all the original information, and removing the duplicated rows recovers A^*. The other five types destroy information that A^* depends on. A NULL erases a value or a reference, an orphaned or shifted key erases which parent a row belongs to, and a conflicting duplicate leaves two candidate values for the same key.

3.4 Dataset Composition

RELIC is generated by applying the injection mechanisms described in Section 3.3 to tables synthesized using the procedure outlined in Section 3.2. For each problem, an *instance* is generated by applying one of the six violation types to a copy of the problem's table pair at a specified *severity*. Severity denotes the fraction of rows that contribute to the answer affected by the violation, set to approximately 10% or 30%. Each problem thus comprises 13 instances: one clean instance with no modification applied, and 12 corrupted instances obtained by combining all six types with two severity levels. RELIC comprises 1,240 problems drawn from the schemas of BIRD, Spider, and TQA-Bench, yielding 16,120 instances evaluated per model. Dataset statistics are reported in Appendix B.3

4 Experimental Setup

4.1 Prompt and Output Format

Each instance is presented to the model as a single prompt containing the query and the two tables serialized in Markdown. The prompt additionally includes two statements: (i) a notice that the tables may contain missing, altered, or duplicated data, and (ii) explicit permission to abstain rather than guess if the data are unreliable. The model returns a structured JSON response with three fields: an *answer*, an *abstention* flag, and a *confidence* score. Full prompt and output templates are provided in Appendix C. We refer to this setting as the *base* condition. Its output format offers abstention as the only structured channel for disclosing a violation. The base condition yields a conservative evaluation. In practice, user prompts neither warn of potential data corruption nor permit abstention. The base prompt provides both, ensuring that any observed silence cannot be attributed to the absence of permission or awareness. The measured abstention rate therefore represents an upper bound on what would be observed under realistic conditions.

4.2 Metrics

We characterize each response r to a corrupted instance using two binary observations. ***Detection***: $\mathbb{1}_{\text{det}}(r) = 1$ if the reasoning trace of r explicitly identifies the injected violation, and 0 otherwise; we compute it with a lexical protocol validated against human judgment (Appendix D). ***Abstention***: $\mathbb{1}_{\text{abs}}(r) = 1$ if the model activates the *abstain* flag. These two observations partition the corrupted instances $\mathcal{C}$ into four mutually exclusive outcomes (Figure 3).

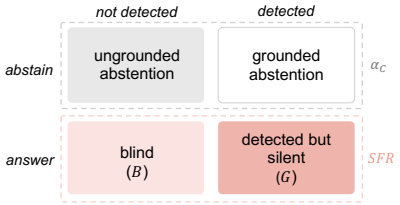


Figure 3: Outcomes of a corrupted instance under the output format.

Silent Failure and its Decomposition. A response that answers a corrupted instance without disclosing the violation constitutes a *silent failure*. The silent failure rate (SFR) is the fraction of corrupted instances on which this occurs. Since abstention is the only structured disclosure channel in the base output format, $\mathrm{SFR} = 1 - \alpha_{\mathcal{C}}$, where $\alpha_{\mathcal{C}}$ denotes the abstention rate on corrupted instances.

Silent failures decompose by detection. We define the detection rate D as the fraction of corrupted instances for which $\mathbb{1}_{\text{det}}(r) = 1$. A silent failure with $\mathbb{1}_{\text{det}}(r) = 0$ is classified as *blind* (B): the model failed to detect the violation. A silent failure with $\mathbb{1}_{\text{det}}(r) = 1$ is classified as *detected-but-silent* (G): the model detected the violation yet answered without disclosure. Their respective rates over the corrupted set $\mathcal{C}$ are

$$B = \frac{1}{|\mathcal{C}|} \sum_{r \in \mathcal{C}} \big(1 - \mathbb{1}_{\text{det}}(r)\big)\big(1 - \mathbb{1}_{\text{abs}}(r)\big), \qquad G = \frac{1}{|\mathcal{C}|} \sum_{r \in \mathcal{C}} \mathbb{1}_{\text{det}}(r)\big(1 - \mathbb{1}_{\text{abs}}(r)\big),$$

with $B + G \leq \mathrm{SFR}$. B, G, and D are reported only for models with accessible reasoning traces.

Abstention Shift. Models may abstain on clean instances for reasons unrelated to the violation. Let $\alpha_{\mathcal{K}}$ denote the abstention rate on the clean instances $\mathcal{K}$. The shift $\Delta\alpha = \alpha_{\mathcal{C}} - \alpha_{\mathcal{K}}$ isolates the abstention attributable to the violation from this baseline tendency. $\Delta\alpha$ serves as the primary metric for comparing models and experimental conditions.

Silent Repair. DUP is the only type whose A^* is recoverable from the corrupted tables (Section 3.3). We therefore report the *silent repair rate*, defined as the fraction of DUP instances on which the model returns A^* without abstaining. Silent repairs form a subset of silent failures: the answer is correct, but the violation remains undisclosed.

4.3 Models

We evaluate 20 models spanning frontier systems, open-weight reasoning models, distilled variants, and non-reasoning baselines. For closed-source frontier models, we assess GPT-5 (Singh et al., 2025) under three reasoning-budget conditions (low, medium, and high). For open API frontier models, we include GLM-5 (Zeng et al., 2026) and Kimi-K2-thinking (Team et al., 2025). For open-weight reasoning models, we include gpt-oss-20B and gpt-oss-120B (Agarwal et al., 2025), Qwen3-8B and Qwen3-32B (Yang et al., 2025), and QwQ-32B (Qwen Team, 2025); the two Qwen3 models are additionally evaluated with reasoning disabled, yielding pairs that share identical weights and differ solely in reasoning. For distilled reasoning models, we include the DeepSeek-R1 distillations at 7B and 70B (Guo et al., 2025). For general non-reasoning models, we include Gemma-4-31B (Team et al., 2026), Qwen2.5 at 7B, 14B, 32B, and 72B (Hui et al., 2024), and Llama-3.1-8B and Llama-3.3-70B (Grattafiori et al., 2024). Finally, we include TableGPT2-7B (Su et al., 2024) as a table-specialized model. Model checkpoints and serving details are provided in Appendix F.

5 Results

5.1 Main Results

Table 1 reports model responses to corrupted tables. $\Delta\alpha$ stays within ± 4.3 percentage points for every model, and four models abstain *less* on corrupted instances than on clean ones. SFR ranges from 84.8% to 99.8%. As discussed in Section 1, if silent failure stems from limited capability, then SFR should decrease as clean accuracy increases, and a model that detects a violation should disclose it. The results contradict both predictions. First, the three frontier models have the highest clean accuracy (87.5–98.3%) yet reach SFR of 97.6–99.7%. Second, every model with a reasoning trace abstains on fewer than 7% of the instances on which it detects the violation. Detection rate D rises from 26.7% for R1-Distill-7B to 92.5% for Kimi-K2, whose detected-but-silent rate G is 92.4%. Stronger models also repair more violations without disclosing them: the silent repair rate on DUP rises from near zero for the weakest models to 5.2–10.4% for the frontier models.

5.2 Violation Severity and Table Volume

We test whether the silence arises because violations are too sparse to notice or tables too large to inspect. Under the first explanation, raising the corrupted fraction from 10% to 30% should increase $\Delta\alpha$. However, $\Delta\alpha$ changes by at most 1.5 percentage points for all eleven configurations (Figure 4a).

Model	cleanAcc ↑	Abstention $\alpha_{\mathcal{K}}$	Abstention $\Delta\alpha$ ↑	SFR ↓	SFR decomposition D ↑	SFR decomposition B	SFR decomposition G ↓	s-repair
Frontier (API)								
GPT-5 (med)	98.3	0.0	+2.4	97.6	–	–	–	8.0
Kimi-K2-thinking	92.0	0.0	+0.3	99.7	92.5	7.3	92.4	5.2
GLM-5	87.5	0.8	+0.1	99.2	76.3	22.7	76.2	10.4
Open-weight, reasoning								
gpt-oss-20B (med)	47.5	3.7	+4.0	92.3	64.1	29.8	62.0	3.8
gpt-oss-120B (med)	62.5	4.6	+2.3	93.2	69.2	23.5	67.9	8.7
Qwen3-8B (think)	17.4	6.5	+1.1	92.4	48.9	45.6	46.0	2.9
Qwen3-32B (think)	25.0	4.2	+1.9	93.9	67.7	24.1	64.6	4.4
QwQ-32B	22.5	3.3	+4.3	92.4	68.1	18.8	64.9	6.0
R1-Distill-7B	0.9	1.9	+2.8	95.3	26.7	69.9	25.0	0.1
R1-Distill-70B	2.5	10.8	+3.2	86.0	53.7	30.9	50.9	2.8
Open-weight, non-reasoning								
Gemma-4-31B	63.4	0.0	+3.2	96.8	–	–	–	7.2
Qwen2.5-7B	0.0	13.4	+1.9	84.8	–	–	–	0.1
Qwen2.5-14B	0.8	5.8	−2.4	96.7	–	–	–	0.6
Qwen2.5-32B	1.7	4.3	−1.2	96.9	–	–	–	0.1
Qwen2.5-72B	1.7	0.0	+0.3	99.7	–	–	–	0.2
Qwen3-8B (nothink)	0.0	0.0	+0.2	99.8	–	–	–	0.0
Qwen3-32B (nothink)	0.0	10.0	−0.2	90.2	–	–	–	0.0
Llama-3.1-8B	0.8	0.8	+1.8	97.4	–	–	–	0.3
Llama-3.3-70B	8.3	10.0	+1.3	88.7	–	–	–	2.7
Table-specialized								
TableGPT2-7B	0.8	8.3	−0.1	91.8	–	–	–	0.3

Table 1: Responses to corrupted tables under the base condition. All values are percentages. $\alpha_{\mathcal{K}}$ is the abstention rate on clean instances and $\Delta\alpha$ the shift on corrupted instances (Section 4.2). D is the detection rate; B and G decompose SFR into blind and detected-but-silent, with $B + G \leq$ SFR (responses without a reasoning trace are unclassified). s-repair is the silent-repair rate on DUP. Dashes: the model exposes no reasoning trace.

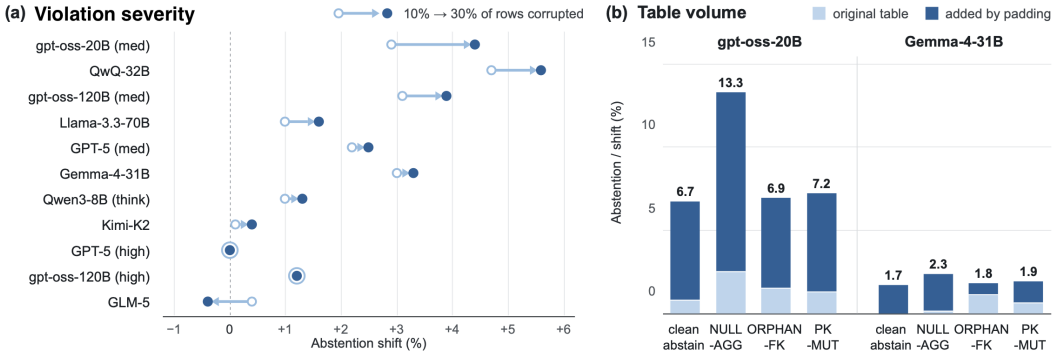


Figure 4: Dataset-side controls. (a) Abstention shifts at 10% vs. 30% corrupted rows. (b) The padded control: light segments are original table values, dark segments the increase from padding; clean-abstain is an absolute rate, the other groups are shifts. All values are percentages.

Under the second explanation, enlarging the tables should reduce abstention on corrupted instances. The *padded* condition adds filler rows while keeping the violation and the gold answer unchanged. Padding instead increases abstention on clean as well as corrupted instances (Figure 4b), reflecting a general tendency to abstain on large tables rather than improved detection. Neither explanation accounts for the silence. [1]

5.3 Effect of Reasoning Effort

To isolate the effect of reasoning from other confounds, we vary only the reasoning budget of gpt-oss-120B and GPT-5 from low to high. If silent failure reflects insufficient reasoning, a higher budget should reduce SFR. Figure 5 shows the opposite.

[1] Appendix E reports $\Delta\alpha$ at each severity and results for each violation type.

Model	CONF-DUP		DUP		ORPHAN-FK	
	informed	explicit	informed	explicit	informed	explicit
RL reasoning						
gpt-oss-120B	+67.5	+66.6	+34.2	+53.1	+20.0	+19.7
QwQ-32B	+62.2	+55.2	+60.1	+76.3	+22.2	+6.8
Qwen3-32B (think)	+21.2	+24.5	+27.2	+65.8	+2.9	−1.2
gpt-oss-20B	+32.5	+40.6	+19.4	+36.9	+13.9	+15.2
Qwen3-8B (think)	+10.0	+7.1	+15.7	+36.8	+5.2	+5.0
Distilled reasoning						
R1-Distill-70B	+13.7	+12.9	+0.4	+10.2	+6.5	+6.3
Non-reasoning						
Gemma-4-31B	+4.2	+42.5	0.0	+5.1	+1.3	+1.3
Llama-3.3-70B	+7.2	+10.0	+9.6	+15.5	+4.2	+5.8
Qwen2.5-72B	0.0	0.0	−0.8	−0.8	+0.4	−0.4

Table 2: Abstention shifts (%p) under the two prompt variants. Under base, shifts are near zero for every model (Table 1). Shading deepens with the shift.

From the lowest to the highest budget, clean accuracy increases in both models, yet SFR also increases. Since GPT-5 does not expose its reasoning trace, we decompose SFR only for gpt-oss-120B. As the budget increases, B decreases from 46.1% to 4.7%, whereas G increases from 48.8% to 92.3%. At the highest budget, the model abstains on only 2% of the instances on which it detects the violation. A higher budget thus converts missed violations into detected but undisclosed ones rather than reducing silent failure.

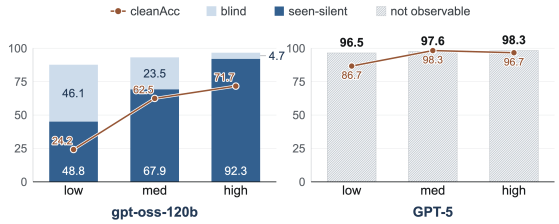


Figure 5: Effect of reasoning effort. The line shows clean accuracy, and the bars show SFR decomposed into B (blind) and G (detected-but-silent).

5.4 Prompt Variants

We apply two *prompt-level interventions* to the base condition: verification instructions in this section and a reporting field in the next. The base prompt permits abstention but does not direct the model to verify integrity (Section 4.1). We test two progressively stronger variants: *informed* instructs the model to verify referential integrity before answering and to abstain if the data are unreliable; *explicit* additionally enumerates all six violation types. Both variants keep the base output format, so $\Delta\alpha$ measures their effect.

In the five RL-trained reasoning models (the other models show small or isolated shifts; Table 2), *informed* raises the shift on *duplicated primary keys* (DUP, CONFLICT-DUP) to as much as +67.5 points. Its shift on *orphaned foreign keys* (ORPHAN-FK) is the smallest of the three types in each of these models and at most +22.2 points, despite the instruction to verify referential integrity. Under *explicit*, the DUP shift ranges from +36.8 to +76.3 points, whereas the ORPHAN-FK shift stays flat or decreases in four of the five models. Verification instructions thus elicit disclosure of duplicated primary keys but leave most orphaned foreign keys undisclosed.

5.5 Output Format

The base output format provides no field for reporting a violation alongside an answer. If silence reflects the absence of such a field, adding one should elicit disclosure. The *issues* variant adds this reporting field to the output JSON (`"issues": [...]`) and requests short descriptions of any data problems found. The system prompt is identical to base and contains no instruction to check the data. A report counts as a disclosure only if it describes the type of the injected violation.

With the reporting field, models disclose violations by reporting them rather than by abstaining: abstention remains at 0.7–1.4% (Figure 6b). Reporting can fail in two ways: a false alarm flags a problem that does not exist, and an omission misses one that does. False alarms occur on 7.7–20.0% of clean instances (Figure 6c). Omissions depend on the violation type (Figure 6a). Duplicated primary keys are rarely missed: at most 13.2% go unreported. For these violations, the silence under the base format largely reflects the absence of a reporting field. In contrast, orphaned foreign keys (ORPHAN-FK, SHIFT-PK) are mostly missed: more than 58.5% go unreported. Like verification instructions (Section 5.4), the reporting field thus elicits disclosure of duplicated primary keys but leaves most orphaned foreign keys undisclosed. This contrast indicates that silent failures do not share a single cause across violation types. Mitigating them therefore requires type-specific analysis, which RELIC enables by isolating each violation type under controlled conditions.

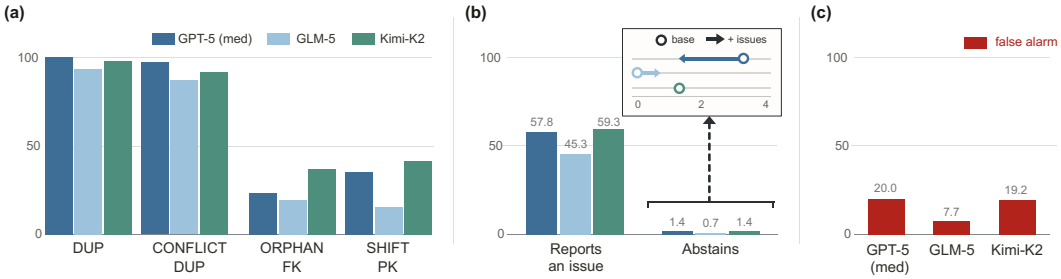


Figure 6: Effect of the reporting field on the three frontier models. (a) Reporting rate by violation type. (b) Average reporting rate across violation types and abstention rate; the inset shows the change in abstention from base. (c) False-alarm rate, the fraction of clean instances on which the model reports any problem.

6 Discussion

Why Models Fail Silently. Reasoning models often detect a violation yet answer without disclosing it (G in Table 1), but explicit instructions elicit disclosure of DUP from the same models (Section 5.4). This suggests that, for DUP, silence on detected violations reflects a default behavior rather than an inability to disclose. This default is consistent with prior findings on post-training: RLHF favors responses that match user expectations over truthful ones (Sharma et al., 2024; Shapira et al., 2026), and RLVR, which underlies the training of many reasoning models, degrades abstention (Kirichenko et al., 2026). Post-training may thus reward returning the requested answer over reporting a violation in the input, which would explain why disclosure does not improve with capability even as detection does.

For ORPHAN-FK, which the same instructions leave largely undisclosed, detection itself is a bottleneck. NULL-FK and ORPHAN-FK exclude the same child rows from the join but differ in where the violation is visible: a NULL appears within the child table, whereas an orphaned key is revealed only by matching it against the parent keys. All three models in Table 12 detect ORPHAN-FK by 30–66% less often than NULL-FK, indicating that violations requiring cross-table key matching are harder to detect. Disclosure and detection failures may thus call for different interventions: the former concerns how models report violations they detect, and the latter how they match keys across tables. Applying the appropriate intervention requires knowing which failure occurs, which answer accuracy cannot reveal. RELIC identifies it for every model and violation type, providing the fine-grained evaluation needed to improve how LLMs handle integrity violations.

Limitations. This study has two main limitations. First, RELIC covers two-table queries with single-column keys and numeric aggregation; whether the behavior persists on queries joining three or more tables, composite-key constraints, and a wider range of column types and operations remains to be tested. The injection procedure generalizes beyond the current setting, and we leave this extension to future work. Second, detection is measured by lexical pattern matching, which cannot register detections phrased outside the vocabulary. Against majority human judgment, the detector attains a precision of 100% and a recall of 92.1% (Appendix D.1): it misses some detections but reports no false ones.

7 Conclusion

Integrity violations are frequent in practice. When a model answers a corrupted input without disclosing the violation, this silent failure is visible neither to the user nor to answer accuracy. We introduce RELIC and define SFR to evaluate whether LLMs detect and disclose integrity violations in multi-table inputs. Decomposing silent failures by detection separates missed violations from detected but undisclosed ones. SFR exceeds 84% for all 20 models and 97% for all three frontier models, which have the highest clean accuracy. Detection increases with model capability and reasoning budget, whereas disclosure does not. In reasoning models, prompt-level interventions elicit disclosure of duplicated primary keys, but most orphaned foreign keys, which arise only in multi-table inputs, remain undisclosed. Silent failure thus persists as models and reasoning budgets scale and is only partly mitigated by prompting. RELIC localizes each silent failure by violation type and by stage (detection or disclosure). This enables future work to measure which failure an intervention mitigates.

AI USE STATEMENT

We used generative AI tools to assist with translation and to provide feedback on the methodology and experiments, with all methodological decisions made by the authors. No other task requiring disclosure under the ICLR 2027 policy involved generative AI. We also used such tools to draft and edit parts of the paper for readability and to identify and summarize related literature. All AI-assisted content was reviewed and revised by the authors, and every citation was checked against the original publication. The language models evaluated in our experiments are objects of study and are not covered by this statement. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We release RELIC together with the code for table synthesis, violation injection, model evaluation, and scoring as anonymized supplementary material. Section 3.1 defines the integrity conditions. Section 3.4 describes how table pairs are synthesized from public schemas, and Section 3.3 specifies each violation type and its injection procedure. The prompt templates for all conditions are listed in Appendix C. Section 4.3 lists the evaluated models, and Appendix F reports model checkpoints, API versions, and reasoning-budget settings.

REFERENCES

Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. Detecting data errors: Where are we and what needs to be done? *Proceedings of the VLDB Endowment*, 9(12):993–1004, 2016.

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Shir Ashury-Tahan, Yifan Mai, Rajmohan C, Ariel Gera, Yotam Perlitz, Asaf Yehudai, Elron Bandel, Leshem Choshen, Eyal Shnarch, Percy Liang, and Michal Shmueli-Scheuer. The mighty ToRR: A benchmark for table reasoning and robustness in LLMs. In Vivek Gupta, Kaize Ding, Harsha Kokel, Yue Zhao, Amit Agarwal, Yu Wang, Michael Glass, Yu Zhang, Kavitha Srinivas, Xiusi Chen, Oktie Hassanzadeh, Qi Zhu, Shuaichen Chang, and Yuan Luo (eds.), *Proceedings of the First Workshop on Structured Understanding, Retrieval, and Generation in the LLM Era (SURGeLLM 2026)*, pp. 16–51, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-406-4. doi: 10.18653/v1/2026.surgellm-1.2. URL https://aclanthology.org/2026.surgellm-1.2/.

Shuaichen Chang, Jun Wang, Mingwen Dong, Lin Pan, Henghui Zhu, Alexander Hanbo Li, Wuwei Lan, Sheng Zhang, Jiarong Jiang, Joseph Lilien, et al. Dr. spider: A diagnostic evaluation benchmark towards text-to-sql robustness. *arXiv preprint arXiv:2301.08881*, 2023.

Peter Baile Chen, Yi Zhang, and Dan Roth. Is table retrieval a solved problem? exploring join-aware multi-table retrieval. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 2687–2699, 2024.

Chao Chow. On optimum recognition error and reject tradeoff. *IEEE Transactions on information theory*, 16(1):41–46, 1970.

Shangbin Feng, Weijia Shi, Yike Wang, Wenxuan Ding, Vidhisha Balachandran, and Yulia Tsvetkov. Don't hallucinate, abstain: Identifying LLM knowledge gaps via multi-LLM collaboration. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14664–14690, 2024.

Yonatan Geifman and Ran El-Yaniv. Selective classification for deep neural networks. *Advances in neural information processing systems*, 30, 2017.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Ken Gu, Zhihan Zhang, Kate Lin, Yuwei Zhang, Akshay Paruchuri, Hong Yu, Mehran Kazemi, Kumar Ayush, A Ali Heydari, Max Xu, et al. RADAR: Benchmarking language models on imperfect tabular data. *Advances in Neural Information Processing Systems*, 38, 2026.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Amita Kamath, Robin Jia, and Percy Liang. Selective question answering under domain shift. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pp. 5684–5696, 2020.

Yujin Kang, Park Seong Woo, and Yoon-Sik Cho. GRIT: Guided relational integration for efficient multi-table understanding. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 21984–21997, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1118. URL https://aclanthology.org/2025.emnlp-main.1118/.

Polina Kirichenko, Mark Ibrahim, Kamalika Chaudhuri, and Samuel J Bell. Abstentionbench: Reasoning llms fail on unanswerable questions. *Advances in Neural Information Processing Systems*, 38, 2026.

Gyubok Lee, Hyeonji Hwang, Seongsu Bae, Yeonsu Kwon, Woncheol Shin, Seongjun Yang, Minjoon Seo, Jong-Yeup Kim, and Edward Choi. EHRSQL: A practical text-to-sql benchmark for electronic health records. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 15589–15601. Curran Associates, Inc., 2022. doi: 10.52202/068431-1134. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/643e347250cf9289e5a2a6c1ed5ee42e-Paper-Datasets_and_Benchmarks.pdf.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36:42330–42357, 2023.

Weizheng Lu, Jing Zhang, Ju Fan, Zihao Fu, Yueguo Chen, and Xiaoyong Du. Large language model for table processing: A survey. *Frontiers of Computer Science*, 19(2):192350, 2025.

Vaishali Pal, Andrew Yates, Evangelos Kanoulas, and Maarten de Rijke. MultiTabQA: Generating tabular answers for multi-table question answering. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 6322–6334, 2023.

Zipeng Qiu, You Peng, Guangxin He, Binhang Yuan, and Chen Wang. Tqa-bench: Evaluating llms for multi-table question answering with scalable context and symbolic extension. *arXiv e-prints*, pp. arXiv–2411, 2024.

Qwen Team. QwQ-32B: Embracing the power of reinforcement learning, 2025. URL https://qwen.ai/blog?id=qwq-32b.

Erhard Rahm, Hong Hai Do, et al. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

Itai Shapira, Gerdus Benade, and Ariel D. Procaccia. How RLHF amplifies sycophancy. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=XN4pWKtA5h.

Mrinank Sharma, Meg Tong, Tomek Korbak, David Duvenaud, Amanda Askell, Sam Bowman, Esin Durmus, Zac Hatfield-Dodds, Scott Johnston, Shauna Kravec, et al. Towards understanding sycophancy in language models. In *International Conference on Learning Representations*, volume 2024, pp. 110–144, 2024.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

Aofeng Su, Aowen Wang, Chao Ye, Chen Zhou, Ga Zhang, Gang Chen, Guangcheng Zhu, Haobo Wang, Haokai Xu, Hao Chen, et al. Tablegpt2: A large multimodal model with tabular data integration. *arXiv preprint arXiv:2411.02059*, 2024.

Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. Table meets LLM: Can large language models understand structured table data? a benchmark and empirical study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pp. 645–654, 2024.

Gemma Team, Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Cărbune, Michelle Casbon, et al. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026.

Kimi Team, Yifan Bai, Yiping Bao, Y Charles, Cheng Chen, Guanduo Chen, Haiting Chen, Huarong Chen, Jiahao Chen, Ningxin Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

Bingbing Wen, Jihan Yao, Shangbin Feng, Chenjun Xu, Yulia Tsvetkov, Bill Howe, and Lucy Lu Wang. Know your limits: A survey of abstention in large language models. *Transactions of the Association for Computational Linguistics*, 13:529–556, 2025.

Cornelius Wolff and Madelon Hulsebos. How well do llms reason over tabular data, really? In *Proceedings of the 4th Table Representation Learning Workshop*, pp. 241–250, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3911–3921, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1425. URL https://aclanthology.org/D18-1425/.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Yilun Zhao, Chen Zhao, Linyong Nan, Zhenting Qi, Wenlin Zhang, Xiangru Tang, Boyu Mi, and Dragomir Radev. RobuT: A systematic study of table qa robustness against human-annotated adversarial perturbations. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6064–6081, 2023.

APPENDIX TABLE OF CONTENTS

A Integrity Violations in Existing Benchmarks **14**

A.1 Scope and Procedure 14

A.2 Violation Rates Across Splits 14

A.3 Mapping to RELIC's Violation Types 14

A.4 Representative Violation Examples 15

B Dataset Construction **17**

B.1 Synthetic Table Generation 17

B.2 Violation Injection 17

B.3 Dataset Statistics 18

C Prompt Templates and Output Format **18**

C.1 System Prompt 18

C.2 User Prompt 19

C.3 Output Format 19

C.4 Full Prompt Example 20

D Detection Protocol **21**

D.1 Validation against Human Judgment 21

E Additional Results **22**

E.1 Effect of Corruption Severity 22

E.2 Per-Violation Results 23

E.3 Qualitative Examples 23

F Models checkpoints **26**

A Integrity Violations in Existing Benchmarks

This appendix details the analysis behind the integrity-violation statistics reported in Section 1. We analyze all 286 databases across every publicly released split of BIRD (Li et al., 2023) and Spider (Yu et al., 2018).

A.1 Scope and Procedure

Table 3 summarizes the analyzed splits. We extract the declared foreign-key (FK) edges of each database with `PRAGMA foreign_key_list`. Composite FKs are grouped by constraint id into a single edge. Table and column identifiers are matched exactly and, if no exact match exists, case-insensitively. We discard edges whose parent table or column does not exist and collapse duplicate declarations.

Source	Split	DBs	Gold queries	Join queries	FK edges
BIRD	dev	11	1,534	1,132	104
BIRD	train	69	9,428	7,031	537
Spider	dev	20	1,034	402	63
Spider	train	146	8,659	3,795	713
Spider	test	40	2,147	844	145
Total		**286**	**22,802**	**13,204**	**1,562**

Table 3: Databases and queries analyzed. *Join queries*: gold queries whose parsed SQL contains at least one cross-table equi-join. *FK edges*: foreign-key relationships declared in the schema.

We measure two classes of violation on declared FK edges. **Orphan FK**: a non-null child FK value that is absent from the referenced parent column(s), detected with an SQL `LEFT JOIN` so that SQLite applies its type-affinity coercion when comparing keys. **Parent-key duplication**: the referenced parent key tuple occurs in more than one row. Within this class, we additionally report *contradictory copies*, duplicate groups whose rows share the key but differ in at least one non-key column.

A null FK violates no constraint unless its column is declared `NOT NULL`, and in existing databases it may encode an optional relationship. We therefore exclude null FKs from the headline counts in Table 4, together with null values in aggregated columns, and report both in Table 5. In RELIC, condition (iii) of Section 3.1 requires the foreign-key column to contain no missing values, so a null FK constitutes a violation.

We call a declared FK edge *broken* if it contains at least one orphan row or a duplicated parent key, and we call a join query *affected* if it joins on at least one broken edge. Each query is counted once regardless of how many broken edges it uses. This metric counts the gold queries that join on a broken edge; it does not determine whether the violation changes the gold answer.

A.2 Violation Rates Across Splits

Table 4 reports the violation rates.

Across all 80 BIRD databases, 45.0% contain a violation on a declared FK edge, and 20.2% of join queries are affected. Spider shows a pronounced imbalance across splits: the database-level violation rate drops from 20.0% (dev) to 5.5% (train) and 2.5% (test), so violations concentrate in the split most commonly used for evaluation. Orphan FKs account for most affected queries. Among the 1,648 affected BIRD join queries, 94.7% join on an edge with orphan rows, and 87.1% join on no edge with a duplicated parent key (Table 5).

A.3 Mapping to RELIC's Violation Types

Table 5 reports how often each RELIC violation type occurs in BIRD and Spider. DUP counts any parent-key duplication, and CONFLICT-DUP counts the subset with contradictory copies; null FKs are counted at the database and edge levels only. At the database level, the rarest type in BIRD is

Split	DBs	DBs w/ violation	FK edges	Broken edges	Affected join queries
BIRD dev	11	7 (63.6%)	104	15 (14.4%)	224 (19.8%)
BIRD train	69	29 (42.0%)	537	101 (18.8%)	1,424 (20.3%)
Spider dev	20	4 (20.0%)	63	6 (9.5%)	48 (11.9%)
Spider train	146	8 (5.5%)	713	28 (3.9%)	80 (2.1%)
Spider test	40	1 (2.5%)	145	1 (0.7%)	10 (1.2%)
BIRD	**80**	**36 (45.0%)**	**641**	**116 (18.1%)**	**1,648 (20.2%)**
Spider	**206**	**13 (6.3%)**	**921**	**35 (3.8%)**	**138 (2.7%)**

Table 4: Violations on declared FK edges by split. A database has a *violation* if any declared FK edge contains an orphan row or a duplicate parent key. *Broken edges*: declared FK edges with at least one such violation. *Affected*: join queries that join on at least one broken edge.

	BIRD			Spider		
Violation type	DB	Edge	Query	DB	Edge	Query
NULL-AGG	42.5	–	2.3	1.0	–	0.1
NULL-FK	31.2	12.9	–	3.4	1.0	–
ORPHAN-FK	40.0	16.5	19.1	6.3	3.3	2.7
DUP	13.8	2.7	2.6	1.5	0.8	0.1
CONFLICT-DUP	10.0	2.5	2.6	1.5	0.8	0.1

Table 5: Prevalence (%) of each RELIC violation type in BIRD and Spider. SHIFT-PK shares the counts of ORPHAN-FK. Denominators: 80/206 databases, 641/921 declared FK edges, and 8,163/5,041 join queries for BIRD/Spider; NULL-AGG uses all gold queries. –: not reported.

CONFLICT-DUP (10.0%). In a static snapshot, SHIFT-PK and ORPHAN-FK both produce child rows whose FK is absent from the parent key set, so our analysis counts them jointly under ORPHAN-FK. Among declared FK edges used by a gold query, every edge with parent-key duplication contains at least one group of contradictory copies; DUP and CONFLICT-DUP therefore have identical query-level prevalence.

A.4 REPRESENTATIVE VIOLATION EXAMPLES

Figure 7 presents three violations on declared FK edges from the development splits of BIRD and Spider: orphan FKs in (a) and (b) and parent-key duplication in (c). All values are obtained via read-only queries against the original SQLite databases and are reported without modification.

In (a), FK values in `Examination.ID` of the BIRD `thrombosis_prediction` database, such as 191207, 243664, and 255934, have no matching entry in `Patient.ID`. In aggregate, 90.9% of `Examination` rows fail to satisfy the referential-integrity constraint. In (b), all `atom` rows of molecule TR003 in the BIRD `toxicology` database, such as TR003_1, TR003_3, and TR003_5, reference a `molecule_id` that is absent from `molecule`. In aggregate, 26.1% of `atom` rows fail to satisfy the referential-integrity constraint.

In (c), the column `AREA_CODE_STATE.state` of the Spider `voter_1` database, which `VOTES.state` references as its parent key, is not unique: `state='NY'` maps to 14 rows and `state='CA'` maps to 36 rows. Consequently, a join between `VOTES` and `AREA_CODE_STATE` yields 14 rows for each NY vote and 36 rows for each CA vote. Of the 51 distinct `state` values, 37 are duplicated, and the rows within every duplicate group carry distinct `area_code` values; all 37 groups therefore contain contradictory copies (CONFLICT-DUP).



Figure 7: Integrity violations observed in the development splits of BIRD and Spider. Each panel shows a child table (left) and its referenced parent table (right). Solid lines denote FK values that resolve to a parent row. Dashed lines with $\times$ denote FK values without a matching parent row, and dashed boxes mark the corresponding missing parent rows. Curved lines denote fan-out, in which a single child row joins with multiple parent rows that share the same key. The donut charts in (a) and (b) break down the child rows by FK status, and the donut chart in (c) reports the duplicated key values among all distinct key values.

B Dataset Construction

B.1 Synthetic Table Generation

Each table pair is generated from the schema and column statistics of a source database, and no original cell values are reused. Table 6 lists the generation rule for each column type. Each column preserves its type and its profiled range or number of distinct values; neither the marginal distribution nor cross-column correlations are preserved. Every cell is produced by a seeded random number generator whose seed is a deterministic function of the master seed, database name, table name, column name, and row index, which makes generation independent across cells and reproducible at the bit level.

Column type	Generation rule	Distribution
Integer	`randint(min, max)`	Discrete uniform over profiled range
Real	`uniform(min, max)`, rounded to 3 decimal places	Continuous uniform
Date	Year, month, day drawn independently	Uniform; year: profiled range, month: 1–12, day: 1–28
Categorical	Draw from label pool of size n_{distinct}	Uniform over pool
Free text	Concatenate from 32-word pool to target length	Uniform at word level
PK / unique	Sequential integers or prefixed IDs (e.g. `S1`, `S2`, ...)	Deterministic (uniqueness guaranteed)

Table 6: Value generation rules by column type.

B.2 Violation Injection

Every corrupted instance satisfies two constraints on its *support rows*, the rows that contribute to A^*. First, at least one support row is corrupted. Second, at least one support row is left intact, so that executing Q on the corrupted pair returns a non-empty numeric result. Problems with only one support row cannot satisfy both constraints and are discarded during generation.

Table 7 lists the target and the modification of each violation type. Let n denote the number of support rows. The number of corrupted rows is $m = \max(1, \lceil \text{rate} \cdot n \rceil)$, so on small tables the realized fraction exceeds the nominal rate.

Violation type	Target	Modification
NULL-AGG	Aggregated column cell	Cell $\rightarrow$ `NULL`
NULL-FK	Child FK cell	Cell $\rightarrow$ `NULL`
ORPHAN-FK	Child FK cell	Cell $\rightarrow$ ghost key
SHIFT-PK	Parent PK cell	Cell $\rightarrow$ ghost key
DUP	Parent rows used in the answer	Row insertion
CONFLICT-DUP	Parent rows used in the answer	Row insertion (filter value changed)

Table 7: Target and modification of each violation type. A ghost key is a value absent from the parent key set.

ORPHAN-FK and SHIFT-PK replace FK or PK values with *ghost keys*. For integer keys, unused values within the existing range are selected first (e.g. if the parent keys are {1, 2, 5}, the ghost key may be 3 or 4); if no gaps remain, values beyond the maximum are used (6, 7, ...). For string keys, a single digit is appended to an existing key (e.g. `S2` $\rightarrow$ `S27`). In both cases the ghost key preserves the format of the original key, so it cannot be identified from its format alone. Ghost keys are mutually distinct and disjoint from all existing keys.

For CONFLICT-DUP, the corrupted row is always a copy of a parent row that matches the query's filter. The copy replaces the filter-column value with a different value drawn from the same column (e.g. `New York` $\rightarrow$ `Chicago`); if no alternative value exists, the problem is discarded. The duplicated key therefore carries exactly one row that matches the filter and one that does not. Executing Q on the corrupted pair still returns A^*, but the tables no longer determine whether the rows of this key satisfy the filter.

B.3 Dataset Statistics

RELIC contains 1,240 problems derived from databases in BIRD, Spider, and TQA-Bench. Each problem yields one clean instance and 12 corrupted instances (6 types $\times$ 2 severities), for a total of 16,120 instances.

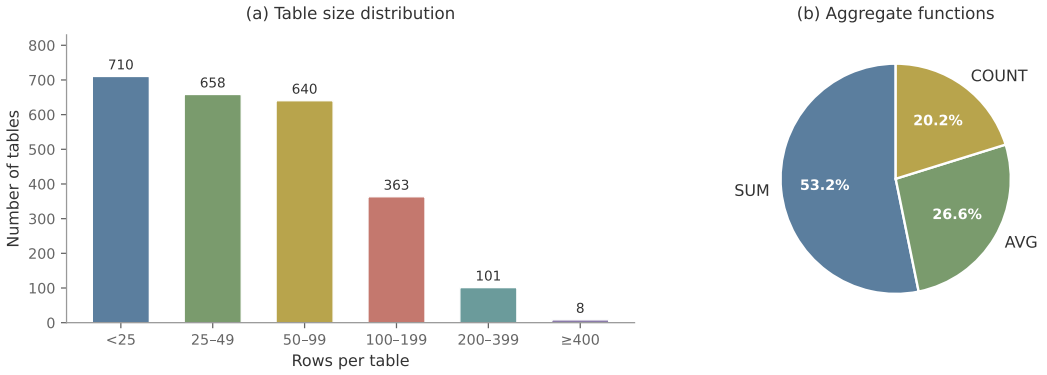


Figure 8: (a) Distribution of rows per table across all 2,480 tables (1,240 parent + 1,240 child). (b) Distribution of aggregate functions across 1,240 problems.

	Min	Median	Max	Mean	SD
Rows per table	8	46	777	66.1	72.8
Columns per table	2	8	115	12.8	17.8
Cells per table	30	435	3,108	536.8	497.3

Table 8: Size statistics of the 2,480 clean tables in RELIC.

Figure 8 shows the distribution of rows per table and of aggregate functions, and Table 8 provides summary statistics. The row distribution is right-skewed: the median is 46 rows, whereas the maximum reaches 777. The number of rows matching the query filter has a median of 5 and a maximum of 60. Across all 16,120 instances, RELIC contains 2,129,842 rows, 17,307,719 data cells.

C Prompt Templates and Output Format

This appendix provides the exact prompts and output format used in all experiments. The system prompt varies across three conditions; the user prompt (Section 5.4) and output format (Section 5.5) are shared across all conditions unless noted otherwise.

C.1 System Prompt

The system prompt consists of a shared prefix, an optional warning sentence that varies by condition, and a shared suffix. The three conditions differ only in the warning sentence.

Shared Prefix.

```
You are a precise table question-answering assistant.
Work out the answer internally and keep your reply short
-- do NOT write code or long explanations.
```

Shared Suffix.

```
Your reply MUST end with exactly one line containing
the JSON object in the format the user specifies, and
nothing after it.
```

Condition-specific Warning. Table 9 lists the sentence inserted between the prefix and suffix for each condition. Under **base**, nothing is inserted.

Condition	Inserted sentence
base	(none)
informed	`The data may contain integrity problems (broken references, duplicate keys, missing values); verify referential integrity between the tables before answering, and abstain if the data is unreliable.`
explicit	`The data may contain integrity problems. Pay attention to all six artifact types during your reasoning: (1) NULL values in the numeric column you aggregate; (2) NULL values in the join-key column; (3) child rows whose join key matches no row in the parent table (orphan references); (4) parent key values that were altered so their child rows no longer match; (5) duplicated parent rows that inflate join results; (6) rows sharing the same key but carrying contradictory values. Verify referential integrity between the tables before answering, and abstain if the data is unreliable.`

Table 9: System-prompt warning sentences by condition. The shared prefix and suffix are identical across all three conditions. The only variable is the inserted sentence.

C.2 User Prompt

The user prompt is identical across all conditions and contains two sentences that are relevant to the measurement: a notice that *tables may contain corrupted data, and a permission to abstain.* These sentences are present even under the **base** condition, making the measurement conservative (Section 4).

```
You are answering a question about the table below.
Some tables may contain missing, altered, or duplicated
data.
If the table does not contain enough reliable
information to answer, you may abstain instead of
guessing.
Table:
{table_md}
Question: {question}
End your response with exactly one JSON object on its
own line:
{"answer": <value or null>, "abstain": <true|false>,
"confidence": <integer 0-100>}
```

`{table_md}` is replaced with the two tables serialized in Markdown. `{question}` is replaced with the natural-language query. NULL values in the table are rendered as the literal string `NULL`.

C.3 Output Format

Base Format. The model's response must end with a single JSON object containing three fields:

```
{"answer": <value or null>, "abstain": <true|false>,
"confidence": <integer 0-100>}
```

`answer` is the numeric answer or `null`. `abstain` is a boolean indicating whether the model declines to answer. `confidence` is an integer between 0 and 100.

Issues Variant. The **issues** condition (Section 5.4) adds one field to the output JSON. The system prompt remains identical to **base**; only the output schema changes:

```
{"answer": <value or null>, "abstain": <true|false>,
"confidence": <integer 0-100>, "issues": [<short
descriptions of any data problems found; empty array
if none>]}
```

No instruction to check the data is added. The field is the only difference from the base format.

C.4 Full Prompt Example

The following is a complete **base**-condition prompt for the clean instance .

System prompt

```
You are a precise table question-answering assistant. Work out the
answer internally and keep your reply short -- do NOT write code
or long explanations. Your reply MUST end with exactly one line
containing the JSON object in the format the user specifies, and
nothing after it.
```

User prompt

```
You are answering a question about the table below. Some tables may
contain missing, altered, or duplicated data.
If the table does not contain enough reliable information to answer,
you may abstain instead of guessing.
Table:
## Store (parent table)

| store_id (PK) | city |
|---|---|
| S1 | New York |
| S2 | New York |
| S3 | California |

## Sales (child table)

| sale_id | store_id (FK) | amount |
|---|---|---|
| 3041 | S1 | 10 |
| 3042 | S2 | 12 |
| 3043 | S2 | 8 |
| 3044 | S3 | 5 |

Question: Join Sales to Store on store_id. Total sales amount for
New York stores?
End your response with exactly one JSON object on its own line:
{"answer": <value or null>, "abstain": <true|false>, "confidence":
<integer 0-100>}
```

Expected output

```
{"answer": 30, "abstain": false, "confidence": 100}
```

D DETECTION PROTOCOL

Detection vocabulary

REFERENTIAL

```
orphan|no match|not match|unmatched|no correspond|
not present in|does not (exist|appear) in|missing (from|in) the|
not found in the|no entry (in|for)|cannot find .{0,30}in
```

DUPLICATE

```
duplicat|appears? (twice|multiple|two times)|repeated row|
multiple (rows|entries|records) (with|for|share)|
same (id|key|code).{0,30}(twice|multiple|appears)|conflict|
contradict|inconsisten|two different values
```

NULL-CUE

```
\bNULL\b|missing value|value is missing|
empty (cell|value)|blank
```

GENERIC

```
corrupt|integrity|tamper|anomal|suspicious|
data (issue|problem|error)|unreliable|altered
```

Figure 9: Detection vocabulary. Line breaks fall only after | and are not part of the patterns.

A response is labeled $\mathbb{1}_{\text{det}}(r) = 1$ if any of the four patterns in Figure 9 matches its reasoning trace. Each pattern is applied to the full trace with a single case-insensitive `re.search` call; no sentence segmentation, tokenization, negation handling, or other preprocessing is performed. The same vocabulary is applied to all models.

The patterns are not mapped one-to-one to the six violation types. A single violation may match multiple patterns (e.g., NULL-FK can match both REFERENTIAL and NULL-CUE), and a single pattern may cover multiple types (e.g., DUPLICATE covers both DUP and CONFLICT-DUP). Detection is therefore type-agnostic: $\mathbb{1}_{\text{det}}(r) = 1$ if any pattern matches, regardless of which.

D.1 VALIDATION AGAINST HUMAN JUDGMENT

Sample. We sample 100 reasoning traces from responses to corrupted instances in the base condition, allocated equally across the six corruption types (16–17 per type). The protocol labels 35 of them positive and 65 negative.

Annotation. Three annotators with database expertise label each trace independently. An annotator first reads the reasoning trace against the tables of the instance and judges whether the model identifies a violation that is present in the tables. The annotator then compares this judgment with $\mathbb{1}_{\text{det}}(r)$ and marks whether the two agree. The reference label of a trace is the majority vote of the three judgments.

The annotators agree unanimously on 84 of the 100 traces, with Fleiss' $\kappa = 0.775$. Pairwise agreement ranges from 86.0% to 95.0%, with Cohen's κ from 0.708 to 0.895.

Detector Accuracy. Table 10 compares $\mathbb{1}_{\text{det}}(r)$ with each annotator and with their majority vote. Of the 35 traces that the detector labels positive, the majority vote judges all 35 to identify a violation, yielding a precision of 100% (Wilson 95% CI [90.1, 100]); the detector thus reports no false detection.

Annotator	TP	FP	TN	FN	Agreement (%)	Precision (%)	Recall (%)
A	35	0	59	6	94.0	100.0	85.4
B	33	2	62	3	95.0	94.3	91.7
C	33	2	59	6	92.0	94.3	84.6
Majority vote	35	0	62	3	97.0	100.0	92.1

Table 10: The detection protocol compared with each annotator (A, B, C) and with their majority vote on 100 reasoning traces. A human label is positive if the trace identifies a violation present in the tables; the detector label is positive if any pattern matches. TP: the detector and the reference are both positive; FP: the detector is positive and the reference negative; FN: the detector is negative and the reference positive; TN: both are negative.

Of the 38 traces that the majority vote judges to identify a violation, the detector labels 35 positive, yielding a recall of 92.1% ([79.2, 97.3]). Overall, the detector agrees with the majority vote on 97 of the 100 traces (97.0%; [91.5, 99.0]). Its agreement with a single annotator is lower (94.0% with A, 95.0% with B, and 92.0% with C): on traces where one annotator is outvoted, the detector mostly sides with the majority.

E Additional Results

E.1 Effect of Corruption Severity

Section 5.2 reports that raising the corrupted fraction from 10% to 30% does not substantially change the abstention shift. Table 11 provides the full comparison across all eleven configurations. The nominal fraction triples between the two levels.

Model	floor	Δabstain (10%)	Δabstain (30%)	difference
GPT-5 (med)	0.0	$+2.2$	$+2.5$	$+0.3$
GPT-5 (high)	1.7	$+0.0$	$+0.0$	$+0.0$
gpt-oss-120B (med)	4.6	$+1.9$	$+2.7$	$+0.9$
gpt-oss-120B (high)	2.7	$+1.2$	$+1.2$	$+0.0$
gpt-oss-20B (med)	3.7	$+3.2$	$+4.7$	$+1.5$
Qwen3-8B (think)	6.5	$+1.0$	$+1.3$	$+0.3$
Gemma-4-31B	0.0	$+3.0$	$+3.3$	$+0.3$
QwQ-32B	3.3	$+3.8$	$+4.8$	$+1.0$
Llama-3.3-70B	10.0	$+1.0$	$+1.6$	$+0.6$
Kimi-K2-thinking	0.0	$+0.1$	$+0.4$	$+0.3$
GLM-5	0.8	$+0.4$	-0.4	-0.8

Table 11: Abstention shift at 10% vs. 30% corruption severity. All values are percentages. *Floor*: clean-instance abstention rate. *Difference*: Δabstain(30%) $-$ Δabstain(10%).

Each problem yields one corrupted instance per violation type at each severity. The two severities share the clean instances; the difference Δabstain(30%) $-$ Δabstain(10%) therefore equals the difference in abstention rate between the two severities.

The difference column stays within ± 1.5 percentage points for all eleven configurations. Eight configurations show a positive difference; GLM-5 shows a negative difference (-0.8); GPT-5 (high) and gpt-oss-120B (high) show no difference. The two high-budget configurations show identical shifts at both severities (0.0 for GPT-5 and $+1.2$ for gpt-oss-120B), consistent with Section 5.3, where a higher reasoning budget increases detection but not abstention.

These results indicate that the silence observed in Section 5.1 is not an artifact of low corruption severity. Tripling the fraction of corrupted rows does not produce a meaningful increase in abstention for any model.

Model	Type	$\Delta\alpha$	SFR	D	B	G
gpt-oss-20B	NULL-AGG	+.001	.962	.943	.019	.943
	NULL-FK	+.032	.931	.836	.099	.830
	ORPHAN-FK	+.025	.938	.172	.762	.166
	SHIFT-PK	+.065	.898	.329	.581	.317
	DUP	+.090	.872	.920	.040	.829
	CONFLICT-DUP	+.024	.938	.643	.295	.634
GLM-5	NULL-AGG	−.008	1.000	.979	.021	.979
	NULL-FK	+.008	.983	.854	.125	.853
	ORPHAN-FK	+.004	.987	.485	.501	.483
	SHIFT-PK	−.004	.996	.572	.420	.570
	DUP	−.008	1.000	.890	.110	.890
	CONFLICT-DUP	+.009	.983	.797	.183	.796
Kimi-K2	NULL-AGG	+.004	.996	1.000	.000	.996
	NULL-FK	+.000	1.000	1.000	.000	1.000
	ORPHAN-FK	+.000	1.000	.696	.304	.696
	SHIFT-PK	+.000	1.000	.873	.127	.873
	DUP	+.005	.995	1.000	.000	.995
	CONFLICT-DUP	+.009	.991	.983	.007	.983

Table 12: Per-violation results for three models under the base condition. $\Delta\alpha$: abstention shift over clean instances. D: detection rate from the reasoning trace. B: blind silent-failure rate. G: detected-but-silent rate.

E.2 PER-VIOLATION RESULTS

Table 12 represents the performances by violation type for three models.

Referential types are harder to detect than value types. Violations that require cross-table key matching are harder to detect. NULL-AGG achieves the highest or tied-highest D in every model (.943–1.000), and ORPHAN-FK the lowest (.172–.696). NULL-FK and ORPHAN-FK have the same effect on the answer, since both exclude a child row from the join, but differ in where the violation is visible: a null foreign key appears within the child table, whereas an orphaned key is revealed only by matching the child foreign keys against the parent keys. D on ORPHAN-FK is lower than on NULL-FK by .30 to .66 across the three models.

Detection does not produce abstention. No model achieves $\Delta\alpha > .10$ on any type. G exceeds B on the majority of type–model pairs. Kimi-K2 reaches $D = 1.000$ on NULL-AGG, NULL-FK, and DUP while $\Delta\alpha$ stays at most $+.009$ on all six types, yielding $G = 0.996$ on NULL-AGG and $G = 1.000$NULL-FK.

E.3 QUALITATIVE EXAMPLES

This appendix presents four reasoning traces that illustrate the outcome: Detected-but-Silent, silent repair, and grounded abstention (two cases). Each example is drawn from a different violation type.

Detected-but-Silent: SHIFT-PK Figure 10 illustrates a detected-but-silent failure. The query asks for the total inspection score for businesses named "Osprey." The parent table contains two rows matching this name: business_id 5 and business_id 47. SHIFT-PK has altered one parent key, so business_id 47 has no matching row in the inspections table. The model explicitly notes this absence ("Not found in the inspections table"), which matches the REFERENTIAL pattern (Appendix D). It then ignores the missing row, computes the answer from business_id 5 alone, and returns it without abstaining ($\mathrm{det} = 1, \mathrm{abs} = 0$).

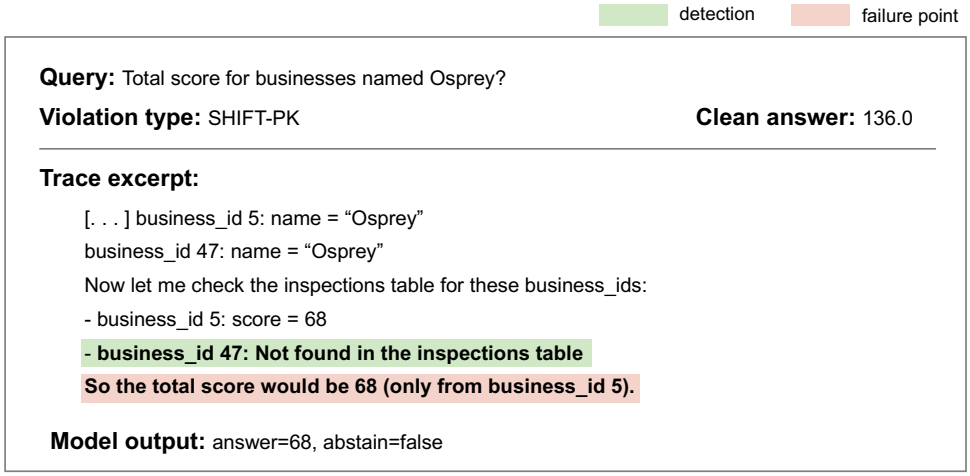


Figure 10: Detected-but-silent failure on SHIFT-PK. The model notes that business_id 47 has no matching row in the inspections table but answers from the remaining row without abstaining. Output: `answer=68, abstain=false`. Clean answer: 136.0.

Silent Repair: DUP. Figure 11 illustrates a silent repair. The query asks for the average carbohydrate value where `yield_unit` is “231 dynx.” Two recipes match the filter: recipe_id 62 and 43. DUP has duplicated these parent rows multiple times. The model resolves the recipe IDs to unique values before querying the child table, producing the clean answer without mentioning the duplication ($\text{det} = 1$, $\text{abs} = 0$, answer $= A^*$).

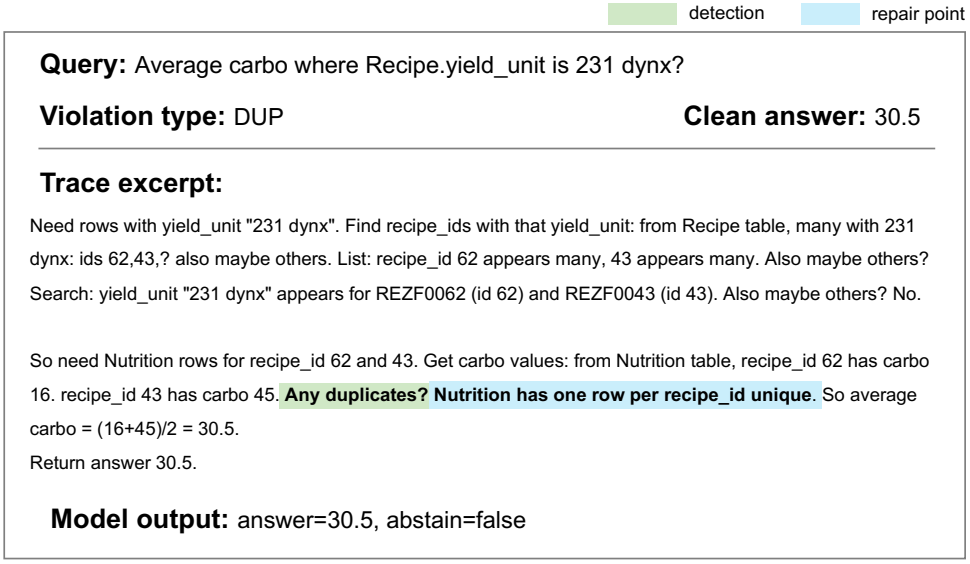


Figure 11: Silent repair on DUP. The model resolves recipe IDs to unique values, producing the clean answer without mentioning the duplication. Output: `answer=30.5, abstain=false`. Clean answer: 30.5.

Grounded Abstention: ORPHAN-FK. Figure 12 illustrates a grounded abstention. The query asks for the total protein where `yield_unit` is “859 wexw.” Two recipes match the filter: recipe_id 6

and 66. ORPHAN-FK has replaced recipe_id 6's foreign key in the Nutrition table with a nonexistent value, so the join returns no row for recipe 6. The model detects the missing row and decides to abstain ($\mathrm{det} = 1, \mathrm{abs} = 1$).

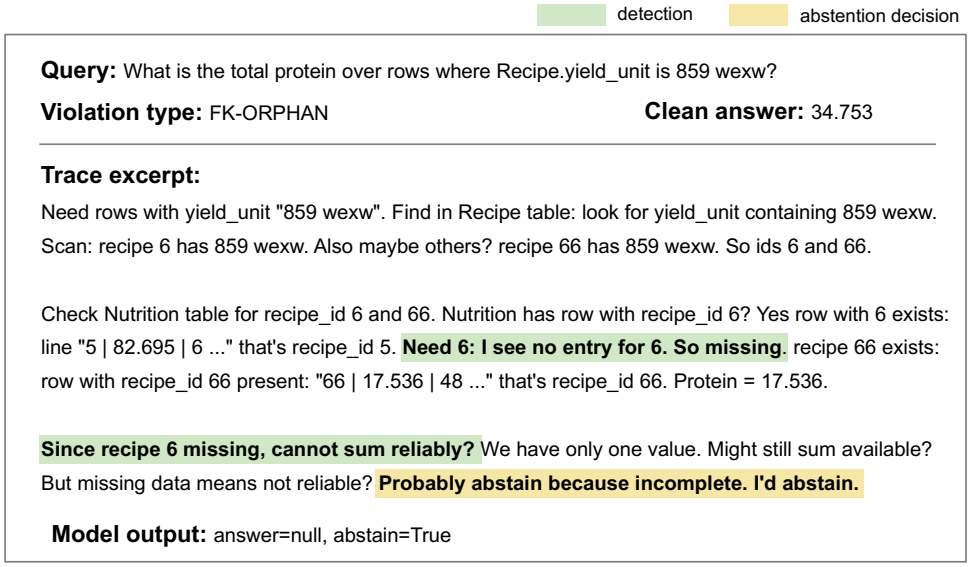


Figure 12: grounded abstention on ORPHAN-FK. The model detects that recipe_id 6 has no matching row in the Nutrition table and abstains. Output: `answer=null, abstain=true`. Clean answer: 34.753.

Grounded Abstention: CONFLICT-DUP. Figure 13 illustrates a second grounded abstention. The query asks for the total `street_num` where `county` is "Badgerx." CONFLICT-DUP has duplicated parent row GERP0009 with a conflicting `county` value: the original carries "Badgerx" and the copy carries "Saffronx." The model identifies the contradictory duplicate and abstains ($\mathrm{det} = 1, \mathrm{abs} = 1$).

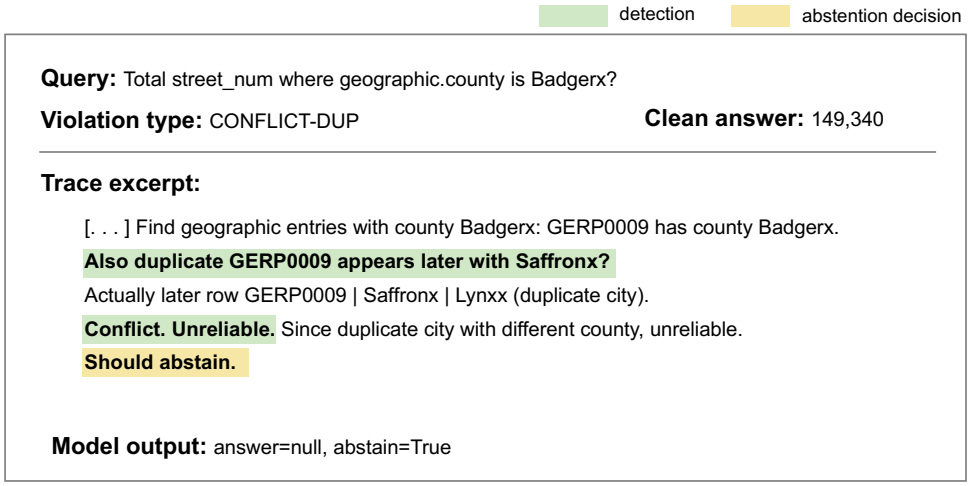


Figure 13: Grounded abstention on CONFLICT-DUP. The model identifies contradictory values for GERP0009 and abstains. Output: `answer=null, abstain=true`. Clean answer: 149,340.

F MODELS CHECKPOINTS

Tables 13 and 14 list all evaluated models with their checkpoints.

Model	HuggingFace repository
Qwen2.5-7B	Qwen/Qwen2.5-7B-Instruct
Qwen2.5-14B	Qwen/Qwen2.5-14B-Instruct
Qwen2.5-32B	Qwen/Qwen2.5-32B-Instruct
Qwen2.5-72B	Qwen/Qwen2.5-72B-Instruct
Llama-3.1-8B	meta-llama/Llama-3.1-8B-Instruct
Llama-3.3-70B	meta-llama/Llama-3.3-70B-Instruct
R1-Distill-7B	deepseek-ai/DeepSeek-R1-Distill-Qwen-7B
R1-Distill-70B	deepseek-ai/DeepSeek-R1-Distill-Llama-70B
Qwen3-8B (think / nothink)	Qwen/Qwen3-8B
Qwen3-32B (think / nothink)	Qwen/Qwen3-32B
QwQ-32B	Qwen/QwQ-32B
gpt-oss-20B (med)	openai/gpt-oss-20b
gpt-oss-120B (low / med / high)	openai/gpt-oss-120b
Gemma-4-31B	google/gemma-4-31B-it
TableGPT2-7B	tablegpt/TableGPT2-7B

Table 13: Local open-weight models. Qwen3 models are evaluated in both thinking and non-thinking modes on the same weights. gpt-oss models are evaluated at up to three reasoning-effort levels.

Model	Model ID	Snapshot
GPT-5 (low / med / high)	gpt-5	gpt-5-2025-08-07
Kimi-K2-thinking	moonshotai/kimi-k2-thinking	kimi-k2-thinking-20251106
GLM-5	z-ai/glm-5	glm-5-20260211

Table 14: API models. GPT-5 variants differ only in the reasoning-effort parameter. Kimi-K2 and GLM-5 were accessed through OpenRouter.

Models with 31B or more parameters (R1-Distill-70B, Llama-3.3-70B, gpt-oss-120B, Gemma-4-31B, Qwen2.5-32B, Qwen2.5-72B, Qwen3-32B, QwQ-32B) were served on 2× NVIDIA A100 80GB. All other models were served on 2× NVIDIA RTX 6000 Ada 48GB.