CROSS-PATH POISONING ATTACKS ON STRUCTURED RAG

Anonymous authors
Paper under double-blind review

ABSTRACT

Retrieval-Augmented Generation (RAG) systems increasingly combine unstructured document retrieval with structured database access, allowing language models to synthesize vector-retrieved evidence and SQL results within a single pipeline. We show that this integration introduces a previously underexplored cross-path poisoning surface: relational content modified through legitimate database operations can be serialized, chunked, and indexed as retrievable text, allowing a single structured data source to influence both reasoning paths. To the best of our knowledge, prior work has not systematically studied poisoning that originates from relational data and propagates through both SQL and vector-retrieval paths in a hybrid RAG pipeline. We study three realizations of this threat—UPDATE, INSERT, and a higher-privilege VECTOR extension—while requiring the correct top-ranked SQL result to remain unchanged. Across Qwen, Llama, and Mistral, the frozen attack configurations induce incorrect answers in all 50 fixed-query runs for every attack–model pair and retain a 93.4% aggregate error rate over 40 paraphrased query variants. Our analysis shows that the vulnerability emerges from the composition of data serialization, relevance-based reranking, and final evidence synthesis: misleading database content can be promoted as highly relevant retrieved evidence and subsequently override independently computed SQL results. We further study retrieval-side anomaly signals, finding strong separation for vector-level poisoning but a substantial detection–false-positive trade-off for text-based attacks. Beyond the specific system evaluated here, our threat model applies to hybrid RAG architectures in which structured records are reused through both deterministic database queries and semantic retrieval, establishing cross-path consistency as a security requirement for this broader class of systems.

1 INTRODUCTION

Retrieval Augmented Generation (RAG) has become a common way to equip large language models (LLMs) with private and continuously updated knowledge without retraining the underlying model Lewis et al. (2020); Borgeaud et al. (2022). In enterprise settings, however, relevant knowledge rarely exists as text alone. Operational data are often split between unstructured documents and relational tables, motivating RAG systems that combine vector retrieval with SQL-based access to structured data Yu et al. (2025). Given a user query, such systems retrieve textual evidence from a vector store, execute SQL queries over relational tables, and ask an LLM to synthesize the two sources into a final answer.

This architecture introduces a cross-path security failure mode that is specific to systems combining structured and unstructured reasoning paths. Structured data are repeatedly transformed before they reach the model. Relational rows may be serialized into Markdown, divided into chunks, embedded, retrieved, and reranked as if they were ordinary documents. As a result, content that is semantically just a value inside a database cell can acquire a different interpretation after serialization. Line breaks, Markdown syntax, table delimiters, or JSON-like fragments may become standalone evidence in the retrieved context. More importantly, the relational and retrieval paths are not necessarily independent: the same database content can contribute to SQL execution while also being converted into documents that populate the vector index.

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.

Existing work has extensively studied retrieval poisoning and other attacks against RAG systems, but these attacks largely consider a single retrieval channel or assume that poisoned documents are injected directly into the unstructured corpus Zeng et al. (2024). In contrast, we study a *cross path poisoning* threat in which an attacker modifies only the structured data source. The malicious content then propagates through the system's normal ingestion pipeline and reappears as retrieved textual evidence, allowing a single database side modification to influence both the structured and unstructured reasoning paths. This setting is particularly concerning because the SQL computation itself can remain correct: the attack succeeds by changing which evidence the model trusts during retrieval and final synthesis, rather than by corrupting the SQL result.

We investigate three realizations of this attack surface. In the UPDATE attack, the adversary appends misleading text to existing free text cells while preserving the numerical values used by SQL aggregation. In the INSERT attack, the adversary inserts schema valid rows whose textual fields contain formatting tokens that are reinterpreted as headings, Markdown tables, or JSON blocks after serialization. We further consider a higher privilege VECTOR extension in which a relational table stores persistent retrieval vectors together with associated text. In this setting, the attacker can influence both whether a malicious record is retrieved and what the reranker and generator subsequently read.

Our experiments show that this attack surface is effective across Qwen, Llama, and Mistral models. Under the selected configurations, all three attacks induce incorrect outputs in 50 out of 50 repeated runs for the fixed query evaluation. The attacks remain effective under paraphrased queries, while exhibiting different failure patterns across models and injection mechanisms. Notably, the SQL path can continue to return the correct aggregate result even when the final answer follows the poisoned retrieved evidence. Our analysis attributes this behavior to two stages of the pipeline: reranking tends to promote attack text that directly resembles an answer, while the synthesis model often gives this retrieved text more weight than the independently computed SQL result. These observations show that the vulnerability is not confined to retrieval alone, but emerges from the interaction between data serialization, retrieval, reranking, and evidence synthesis.

Our contributions are as follows:

- We identify and formalize a cross path poisoning threat in dual path RAG systems, where an attacker modifies only relational data yet influences both SQL backed reasoning and the vector retrieval context. The attack is evaluated under the constraint that the correct SQL ranking and ground truth answer remain unchanged.
- We develop three attack realizations, UPDATE, INSERT, and a higher privilege VECTOR extension, and trace how malicious content propagates from database storage through serialization, chunking, retrieval, reranking, and final answer generation. Experiments across three open-source LLMs show that the resulting failures persist beyond a single fixed query formulation.
- We analyze the mechanisms that enable cross path poisoning and evaluate practical defenses. In addition to input sanitization and integrity checks, we study a lightweight retrieval anomaly detector based on the relationship between retrieval and reranking behavior, highlighting both its detection capability and its false-positive trade-offs.

2 SYSTEM AND THREAT MODEL

2.1 DUAL PATH RAG PIPELINE

We study the dual path RAG architecture illustrated in Figure 1. Given a user query, an LLM first decomposes it into one or more sub queries. Each sub query is processed through two parallel paths. The structured path generates and executes SQL queries over a relational database, while the retrieval path embeds the query and retrieves semantically relevant documents from a vector index. The retriever returns the top 30 candidates, which are reranked to retain the top 5; the top 3 documents are then provided to the answer generation model together with the SQL results.

The two paths are coupled through data ingestion. Relational tables used by the retrieval pipeline are serialized into Markdown, split into chunks of 1,000 characters with an overlap of 200, and embedded before indexing Yu et al. (2025). Consequently, the same underlying database content may appear in two different forms at inference time: as structured SQL output and as serialized text

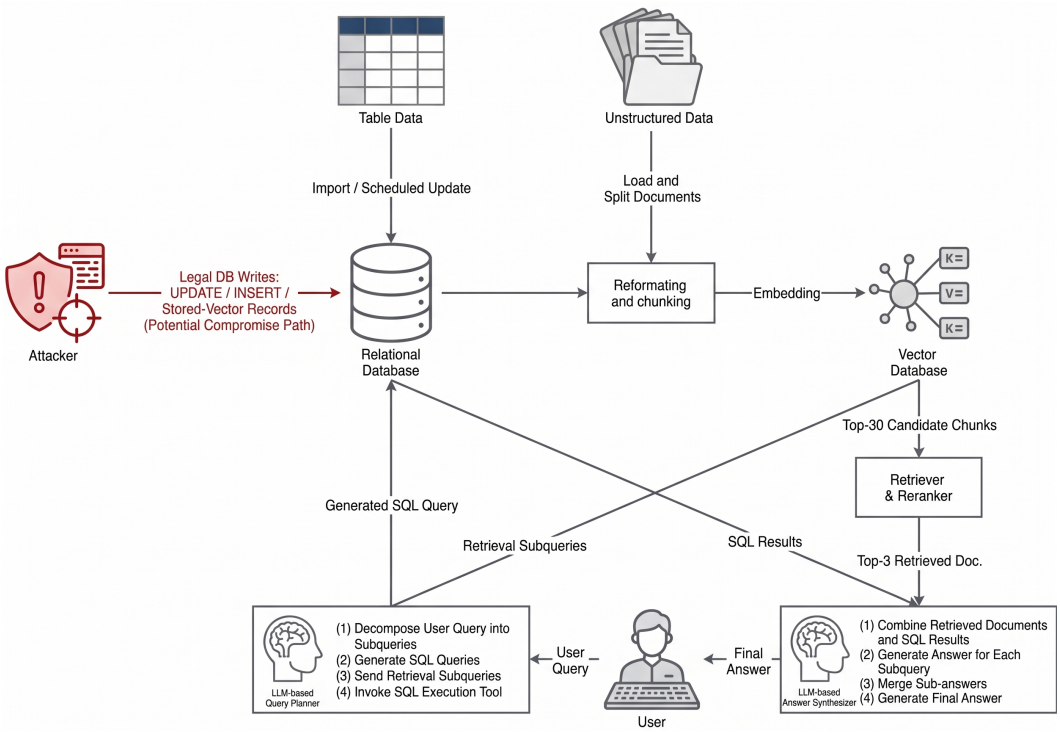


Figure 1: Overview of the dual path RAG pipeline and the cross path poisoning surface. A user query is decomposed into sub queries that access both the relational database and the vector retrieval pipeline. Database content can therefore contribute to the final answer through both SQL execution and its serialized document representation.

retrieved from the vector store. This representation change is the key attack surface considered in this work.

2.2 Experimental Setup

Question decomposition and final answer generation use `Qwen2.5-7B-Instruct`. Dense retrieval uses `BAAI/bge-m3`, followed by `BAAI/bge-reranker-v2-m3`. The relational backend is MySQL 8.0.44 and contains seven tables with 120 rows each. The clean retrieval corpus contains 50,160 chunks, represented by 1,024 dimensional embeddings.

For experiments involving persistent vectors, retrieval records are stored in the relational table `rag_vector_records`, with vectors encoded as `LONGBLOB`. At system startup, these vectors are loaded into an in memory `FAISS IndexFlatIP` index. All embeddings are L2 normalized, so inner product search corresponds to exact cosine similarity retrieval. Attack experiments are executed on isolated database replicas to prevent state from carrying across runs. Additional implementation and reproducibility details are provided in the appendix.

Evaluation metrics. Our primary metric, `wrong`, records whether the model's final prediction differs from the top item obtained from the clean aggregated result, or whether no unique item can be extracted from the answer. We additionally report `target_category`, which measures whether the model selects the attacker specified target, and `target_exact`, which additionally requires the claimed value to match the attack target. Finally, `delivery` measures whether the malicious content reaches the retrieval context, i.e., whether it appears among the top 30 retrieved candidates and the final top 3 reranked documents. Delivery and answer manipulation are evaluated separately. Groups with an invalid clean baseline are excluded according to the fixed evaluation protocol described in the appendix.

2.3 THREAT MODEL

We consider an attacker with legitimate write access to selected parts of the relational database. Under the base threat model, the attacker may modify free text cells in existing columns or insert new rows that satisfy the original table schema. The attacker cannot modify the database schema, embedding model, retriever, reranker, answer generation model, or the vector index directly. Any malicious content must therefore propagate through the standard ingestion pipeline, serialization, chunking, embedding, retrieval, and reranking, before it can influence the final context.

The attacker's objective is to induce an incorrect final answer, with targeted attacks aiming to force a specific incorrect item chosen in advance. To separate poisoning effects from direct corruption of numerical statistics, we impose two invariants. First, the SQL outputs used for evaluation must remain unchanged between the attack and corresponding control condition. Second, the top ranked item produced by the SQL path must remain identical to the clean baseline, and the attacker specified target must not become the top SQL result. Thus, a targeted error can only be attributed to the interaction between the poisoned document representation and the downstream retrieval and synthesis stages, rather than to a changed SQL computation.

We separately evaluate a higher privilege extension in the VECTOR attack. In this setting, the attacker is additionally allowed to insert new records into the relational vector storage table, thereby specifying both the stored text and its associated vector representation. This capability is not assumed by the base UPDATE and INSERT attacks and is reported separately throughout the paper.

3 CROSS PATH POISONING ATTACKS

We study three realizations of cross path poisoning under the threat model in Section 2. The first two attacks, UPDATE and INSERT, operate under ordinary write access to relational data. They modify only database content and rely on the standard serialization and indexing pipeline to propagate the poisoned content into the retrieval path. We additionally consider VECTOR, a higher privilege extension in which the attacker can insert records into a persistent vector storage table.

Across all three attacks, we separate answer manipulation from direct corruption of the SQL computation. The attacker aims to make the final model select an attacker specified target y_t, while the SQL path continues to identify the same correct top ranked item as in the clean database. Thus, attack success requires the poisoned retrieval evidence to override otherwise correct structured evidence during final synthesis.

3.1 UPDATE: POISONING EXISTING TEXT

The UPDATE attack considers an adversary with write access to an existing free text cell. Let D denote the clean database, c the modified cell, and p the appended attack text. The poisoned database is

$$D_{c,p} = D[c \leftarrow D[c] \parallel p], \tag{1}$$

where $\parallel$ denotes concatenation. We append rather than replace content so that the original value remains intact.

The base attack changes exactly one cell,

$$\delta(D, D_{c,p}) = 1, \tag{2}$$

while preserving the SQL results used by the task,

$$\mathrm{SQL}_q(D_{c,p}) = \mathrm{SQL}_q(D), \qquad \forall q \in Q_{\mathrm{dev}}. \tag{3}$$

This constraint is important: the attack does not succeed by changing the underlying aggregate. Instead, the appended text is later serialized with the relational table, indexed as retrievable content, and presented to the generator as an alternative interpretation of the same data.

We construct the payload as

$$p = T_U(\bar{q}, s, y_t, v_t; \rho), \tag{4}$$

where $\bar{q}$ represents the target query type, s specifies the relevant statistic, (y_t, v_t) is the attacker specified claim, and ρ controls its presentation. We use a small, pre defined set of candidate presentations and select the attack configuration on the development queries. The exact templates, candidate selection procedure, and payload examples are provided in Appendix **??**.

3.2 INSERT: EXPLOITING SERIALIZATION

The INSERT attack assumes that the adversary may add schema valid rows but cannot alter the table definition. Unlike UPDATE, its primary attack surface is the representation change that occurs when relational data are serialized into Markdown.

Let K denote the number of malicious inserted cells. For each attack insertion, we additionally introduce balancing content so that the evaluated SQL result and its top ranked item remain unchanged. For the resulting database D_θ, we require

$$\mathrm{rank}\big(\mathrm{SQL}_q(D_\theta)\big) = \mathrm{rank}\big(\mathrm{SQL}_q(D)\big), \qquad \forall q \in Q. \tag{5}$$

The malicious value itself remains a valid database string. However, it contains formatting tokens, such as line breaks, Markdown headings and tables, or JSON like blocks, that acquire structure only after serialization. A value that occupies a single relational cell can therefore appear to the retrieval pipeline as a standalone aggregation result.

We denote the formatted payload by

$$p = T_I(y_t, v_t). \tag{6}$$

Attack effectiveness also depends on where the inserted rows appear in the serialized table. Because rows are ordered by primary key before chunking, row placement determines whether the forged structure remains intact and how many chunks contain it. We therefore represent an INSERT configuration as

$$\theta = (K, p, \pi), \tag{7}$$

where π denotes the placement strategy. We consider tail placement, placement near target rows, distributed placement, and a per chunk strategy. These alternatives expose the interaction between relational ordering and document chunking: identical database values can yield different retrieval behavior solely because they are serialized into different chunk boundaries.

The search over K, π, and candidate payloads is performed on development queries and frozen before subsequent evaluation. Full search criteria, integrity checks, and formatted payload examples are deferred to Appendix **??**.

3.3 VECTOR: HIGHER PRIVILEGE PERSISTENT VECTOR POISONING

Finally, we consider a higher privilege setting in which the system persists retrieval vectors in a relational table and loads them directly when constructing the search index. The attacker may insert new vector records, but cannot modify existing application data, SQL results, retriever parameters, reranker parameters, or the answer generation model.

This setting exposes a second separation between retrieval and content. The stored vector determines whether an injected record is retrieved, whereas its associated text determines whether it survives reranking and influences synthesis. Because the system restores the stored vector directly rather than recomputing it from the associated text, the attacker can control these two components independently.

Let

$$U = \{u_1, \ldots, u_m\}$$

denote the sub queries produced by the system for representative questions. We embed these sub queries using the same retriever and cluster the normalized embeddings into $K_c = 5$ groups. If C_k denotes the k-th cluster center, the stored attack vector is

$$v_k = \frac{C_k}{\|C_k\|_2}, \qquad k = 1, \ldots, K_c. \tag{8}$$

Each vector is paired with text constructed from a representative sub query $\hat{u}_k$:

$$p_k = T_V(\hat{u}_k, s, y_t, v_t). \tag{9}$$

Here, y_t is selected from a non winning item in the clean SQL result, while v_t uses its corresponding clean statistical value. The payload is therefore misleading in its ranking claim rather than in the numerical value itself.

The two components serve different stages of the attack: v_k increases the probability that the record enters the initial retrieval set, while p_k is optimized to remain relevant under text reranking and to influence the final synthesis step. We evaluate several vector construction and text template alternatives on the development set and use the selected configuration unchanged for the subsequent evaluation. The complete candidate space and selection protocol are reported in Appendix **??**.

4 Experimental Results

4.1 Overall Attack Effectiveness

We first evaluate whether the three attacks remain effective once their configurations are selected on the development set and frozen for evaluation. For each attack, model pair, we conduct 50 independent runs using a fixed representative query. We then evaluate robustness to surface form variation using up to 40 semantically equivalent paraphrases of the same underlying question.

Table 1 reports the number of incorrect final answers. Under the fixed query setting, all frozen attack configurations induce an incorrect answer in all 50 runs across Qwen, Llama, and Mistral. The attacks also remain effective under query paraphrasing. Across the nine attack, model combinations, the aggregate error rate is 94.4% for the first 10 and 20 variants, 93.3% for 30 variants, and 93.4% for 40 variants.

Table 1: Matched control statistical analysis (Attack vs. Neutral, $n = 50$ per condition).

Attack	Model	ATK Err	NEU Err	Fisher p	Risk Diff.	OR	Wilson 95%	CP 95%
Update	Qwen	14	2	1.9×10^{-3}	+0.24	7.71	[0.175,0.417]	[0.162,0.425]
	Llama	39	7	1.1×10^{-10}	+0.64	19.92	[0.648,0.872]	[0.640,0.885]
	Mistral	26	40	5.7×10^{-3}	-0.28	0.28	[0.385,0.652]	[0.374,0.663]
Insert	Qwen	27	1	2.2×10^{-9}	+0.52	38.62	[0.404,0.670]	[0.393,0.682]
	Llama	25	7	2.1×10^{-4}	+0.36	5.80	[0.366,0.634]	[0.355,0.645]
	Mistral	48	7	4.0×10^{-18}	+0.82	112.52	[0.865,0.989]	[0.863,0.995]
Vector	Qwen	50	2	2.6×10^{-26}	+0.96	1959.4	[0.929,1.000]	[0.929,1.000]
	Llama	50	8	3.8×10^{-20}	+0.84	505.0	[0.929,1.000]	[0.929,1.000]
	Mistral	26	3	4.1×10^{-7}	+0.46	14.68	[0.385,0.652]	[0.374,0.663]

The degradation under paraphrasing is concentrated in two cases. For Insert, Qwen produces correct answers for three of the 30-query variants and three of the additional ten variants, whereas Llama and Mistral remain almost fully affected. For Vector, Mistral is substantially more sensitive to wording, with 21 errors among 40 variants, compared with 40/40 for Qwen and Llama. Thus, the attacks do not depend solely on repeatedly issuing one fixed prompt, although robustness to paraphrasing varies across attack and model families.

We also observe different response patterns after a successful attack. Under INSERT, Llama tends to reproduce the value stated in the injected content, whereas Qwen and Mistral often report values derived from the underlying data while still selecting the attacker specified category. A similar pattern appears under VECTOR: because this attack does not alter the application tables, Qwen and Mistral may report the correct numerical value but associate it with the wrong winner. This distinction suggests that numerical computation and final evidence selection can fail independently; we analyze this behavior further in Section 5.

4.2 SENSITIVITY TO INJECTION VOLUME

We next vary the number of modified or inserted cells while holding the remaining attack configuration fixed. The full results are reported in Appendix **??**.

The two attacks exhibit markedly different scaling behavior. For UPDATE, Qwen increases from 12/40 errors at $K = 1$ to 40/40 at $K = 3$, while Mistral moves from 0/40 to 40/40 over the same interval; Llama is already fully affected at $K = 1$. This produces a sharp threshold like transition once several cells contain a consistent claim.

In contrast, INSERT improves progressively with injection volume. For Qwen, errors increase from 9/40 at $K = 1$ to 26/40 at $K = 3$ and 37/40 at $K = 10$, with a similar trend for Llama. Thus, UPDATE behaves more like a consistency threshold, whereas INSERT benefits gradually from additional serialized attack blocks.

5 DISCUSSION

Our results suggest that cross path poisoning emerges from the composition of multiple individually reasonable stages. Serialization changes how relational content is represented, allowing formatting tokens and statistical looking statements to become standalone textual evidence. Reranking then promotes content based on query relevance rather than factual correctness, making direct looking attack statements disproportionately likely to reach the generator. Finally, the synthesis model must resolve conflicts between retrieved text and independently computed SQL results without an explicit trust hierarchy. The attack therefore does not require any single component to behave anomalously; the vulnerability appears when these components are composed.

The tested models also exhibit different failure modes. Llama more frequently reproduces both the attacker selected item and value, whereas Qwen and Mistral may preserve a data derived numerical value while still selecting the wrong item. This indicates that numerical consistency and final answer selection can fail independently. Attack robustness is likewise model dependent: Mistral is substantially more sensitive to query paraphrasing under VECTOR, while UPDATE becomes considerably more reliable once several cells repeat the same claim. Together, these observations suggest that defenses should not rely solely on payload filtering or numerical plausibility. More robust systems should preserve provenance across representation changes and explicitly resolve conflicts between retrieved claims and structured computation.

6 SECURITY FRAMEWORK

We finally ask whether cross path poisoning leaves observable signals before answer generation. As a lightweight monitoring prototype, we characterize each non initial retrieval step using two quantities: the rank disagreement between retrieval and reranking, and the prominence of the highest reranking score. A shallow classifier is fitted on benign and attack examples and frozen before evaluation; full feature definitions, thresholds, and training details are provided in Appendix **??**.

The resulting signals are most effective for VECTOR, where the conservative aggregation rule detects all 35 attack queries with one false positive on CLEAN and none on NEUTRAL. For UPDATE and INSERT, the same rule detects 40.0% and 48.6% of attacks, respectively. Triggering on any anomalous retrieval step increases detection to 74.3% and 91.4%, but also raises benign false positive rates substantially.

These results suggest that retrieval behavior can provide useful attack related signals, particularly when the adversary directly manipulates retrieval geometry. However, ranking anomalies alone are

insufficient for reliable detection of text based poisoning. We therefore view retrieval side monitoring as complementary to stronger defenses based on data integrity and consistency checks between retrieved claims and structured SQL results.

7 Related Work

7.1 Structured and Hybrid Retrieval Augmented Generation

Retrieval Augmented Generation (RAG) was originally introduced to ground language model generation in external knowledge sources, improving access to information that is absent from model parameters or changes over time Lewis et al. (2020). As RAG has been extended to enterprise and analytical settings, retrieval increasingly involves not only unstructured documents but also relational and tabular data. Recent systems therefore combine document retrieval with structured reasoning over tables or databases, enabling models to answer questions that require both textual evidence and numerical operations Wang et al. (2024); Yu et al. (2025). Much of this literature focuses on improving retrieval, table reasoning, and answer accuracy. Our work instead studies a security consequence of this integration: structured records may also be serialized, chunked, and indexed as text, allowing a modification to the relational source to influence both the SQL and retrieval paths.

7.2 RAG Security and Knowledge Poisoning

The growing use of external retrieval has introduced new attack surfaces, including indirect prompt injection and knowledge poisoning Chang et al. (2026); Zou et al. (2025). Existing poisoning attacks demonstrate that a small number of adversarial documents can manipulate retrieval and steer downstream generation, while related work has examined vulnerabilities in LLM driven analytical systems and structured data interfaces Wang et al. (2026). Topic oriented poisoning further shows that adversarial content can be crafted to remain retrievable across semantically related queries Gong et al. (2025).

These studies primarily consider attacks on an unstructured retrieval corpus, an LLM facing prompt, or a structured interface as separate attack surfaces. Our setting differs in that the attacker modifies the relational data source and relies on the system's normal ingestion procedure to propagate the same content into the vector retrieval path. The resulting attack is therefore *cross path*: the poisoned representation originates from structured data but is delivered to the model through a separate retrieval channel. Moreover, we evaluate the attack while preserving the correct top ranked SQL result, separating retrieval induced manipulation from direct corruption of the underlying computation.

7.3 Evidence Conflict and Context Reliance

A related line of work studies how language models behave when retrieved evidence conflicts with other available information. Recent results suggest that RAG models can place substantial weight on retrieved context even when the supplied evidence is unreliable, creating failures commonly associated with context over reliance Zhu et al. (2026). This issue becomes particularly important in hybrid systems, where an LLM may receive evidence from sources with different reliability properties, such as retrieved natural language passages and deterministic database computations.

Our work examines this problem under an adversarial setting. Rather than introducing an arbitrary contradiction at the final prompt, the attacker first exploits the data representation and retrieval pipeline to make the misleading evidence highly relevant to the query. The final generator must then reconcile this retrieved claim with independently produced SQL evidence. This setting connects retrieval poisoning with evidence conflict research and highlights that relevance based retrieval does not by itself provide a mechanism for establishing which evidence source should be trusted.

8 Conclusion

We study cross path poisoning in dual path RAG systems, where relational data contribute to both structured SQL reasoning and document retrieval. We introduce three attack realizations, UPDATE,

INSERT, and a higher privilege VECTOR extension, and evaluate them across three open source language models. Our results show that an attacker can manipulate the final answer even when the SQL path continues to return the correct top ranked result.

The experiments reveal that this vulnerability emerges from the interaction of multiple stages rather than from a single compromised component. Serialization can transform ordinary database values into salient textual structures, reranking can promote these query relevant but misleading representations, and the final generator may resolve conflicts in favor of the retrieved evidence over independently computed SQL results. We further observe distinct sensitivity to injection volume: UPDATE exhibits a sharp increase in effectiveness once several consistent modifications are present, whereas INSERT improves more gradually as additional poisoned rows are introduced.

We also investigate whether attack delivery can be detected from retrieval behavior. A lightweight monitoring prototype based on retrieval, reranking score dynamics provides a strong signal for VECTOR attacks, while UPDATE and INSERT expose a clear trade off between detection coverage and benign false positives. These results suggest that retrieval side monitoring can complement, but not replace, stronger integrity and cross path consistency mechanisms.

Overall, our findings highlight a security consequence of combining structured and unstructured retrieval: data transformations that improve accessibility can also create new paths through which trusted relational content is reinterpreted and propagated. Future work should evaluate these attacks on larger and more diverse RAG systems, study adaptive adversaries, and develop mechanisms that explicitly verify consistency between retrieved claims and structured computation before final answer generation.

9 LIMITATIONS

Our evaluation has several limitations.

- **Limited data scale and domain diversity.** Experiments are conducted on a single structured data scenario with relatively small relational tables. Although the attacks are evaluated across multiple query formulations, their effectiveness on larger databases, different schemas, and substantially different application domains remains to be established.
- **Limited model coverage.** We evaluate three open source language models and do not include larger proprietary or frontier models. The model specific behaviors observed in our experiments, including differences in numerical consistency, sensitivity to paraphrasing, and injection volume, should therefore not be assumed to generalize across model families or scales.
- **Dependence on the retrieval pipeline.** Our attacks are evaluated on a pipeline that serializes relational data into Markdown, performs dense retrieval followed by neural reranking, and combines retrieved documents with SQL results during final synthesis. Systems that use different serialization formats, retrieval components, rerankers, or evidence integration strategies may exhibit different attack surfaces and failure modes.
- **Restricted attacker and defense evaluation.** The study considers fixed attacker capabilities and non adaptive evaluation after attack configurations are selected on development queries. We do not study attackers that adapt specifically to the proposed monitoring signals, nor do we claim that the retrieval side detector constitutes a complete defense. Evaluating adaptive attacks and stronger cross path consistency mechanisms remains an important direction for future work.

AI USE STATEMENT

Generative AI tools, including ChatGPT, were used solely for language editing and proofreading to improve the grammar, clarity, and readability of the manuscript. They were not used to generate experimental results or scientific conclusions. The authors reviewed and verified the final manuscript and take full responsibility for its content.

REFERENCES

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pp. 2206–2240. PMLR, 2022.

Hongyan Chang, Ergute Bao, Xinjian Luo, and Ting Yu. Overcoming the retrieval barrier: Indirect prompt injection in the wild for llm systems. In *USENIX Security Symposium*. USENIX Association, 2026.

Yuyang Gong, Zhuo Chen, Jiawei Liu, Miaokun Chen, Fengchang Yu, Wei Lu, XiaoFeng Wang, and Xiaozhong Liu. {Topic-FlipRAG}:{Topic-Orientated} adversarial opinion manipulation attacks to {Retrieval-Augmented} generation models. In *34th USENIX Security Symposium (USENIX Security 25)*, pp. 3807–3826, 2025.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33: 9459–9474, 2020.

Kuncan Wang, Ziting Wang, Peizhuo Lv, Haoyang Li, Guoliang Li, Gao Cong, and Wei Dong. Data agents under attack: Vulnerabilities in llm-driven analytical systems. *arXiv preprint arXiv:2606.08661*, 2026.

Zilong (Ryan) Wang, Hao Zhang, Chun-Liang Li, Julian M Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, and Tomas Pfister. Chain-of-table: Evolving tables in the reasoning chain for table understanding. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 55587–55610, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/f53fd88a4340063ecd258c0ae9948b40-Paper-Conference.pdf.

Xiaohan Yu, Pu Jian, and Chong Chen. Tablerag: A retrieval augmented generation framework for heterogeneous document reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 14063–14082, 2025.

Shenglai Zeng, Jiankun Zhang, Pengfei He, Yiding Liu, Yue Xing, Han Xu, Jie Ren, Yi Chang, Shuaiqiang Wang, Dawei Yin, et al. The good and the bad: Exploring privacy issues in retrieval-augmented generation (rag). In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 4505–4524, 2024.

Xi Zhu, Ziqi Wang, Kai Mei, Wujiang Xu, Minghao Guo, Bangji Yang, Jiajun Fan, and Dimitris N Metaxas. Trust or abstain? a self-aware rag approach. *arXiv preprint arXiv:2605.18792*, 2026.

Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. {PoisonedRAG}: Knowledge corruption attacks to {Retrieval-Augmented} generation of large language models. In *34th USENIX Security Symposium (USENIX Security 25)*, pp. 3827–3844, 2025.