Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Don't Show What They Can't See: Authorization-First Schema Context for Secure Text-to-SQL

Jianan Jiang
Guangzhou University
Zhenxin Zhang
zzx@stu.xidian.edu.cn

Xidian University
Yucheng Long
Guangzhou University
Zhihao Li
Guangzhou University
Yaqi Wu
Guangzhou University
Yu Wang
Guangzhou University

Research Article

Keywords: Text-to-SQL, large language models, schema security, access control, prompt injection, schema retrieval

Posted Date: September 28th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10905100/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Don't Show What They Can't See: Authorization-First Schema Context for Secure Text-to-SQL

Jianan Jiang[1,2], Zhenxin Zhang[3*†], Yucheng Long[1,2†], Zhihao Li[1,2†], Yaqi Wu[1,2†], Yu Wang[1,2†]

[1]Guangdong Key Laboratory of Blockchain Security, Guangzhou University, No. 72 Zhiming Road, Guangzhou, 510725, Guangdong, China.

[2]Institute of Artificial Intelligence, Guangzhou University, No. 230 Waihuan West Road, University Town, Guangzhou, 511436, Guangdong, China.

[3*]School of Cyber Engineering, Xidian University, No. 2 South Taibai Road, Xi'an, 710071, Shaanxi Province, China.

*Corresponding author(s). E-mail(s): zzx@stu.xidian.edu.cn;
Contributing authors: jiangjn1993@gmail.com;
yuchenglong@e.gzhu.edu.cn; zhihaoli@e.gzhu.edu.cn;
1550322596@qq.com; yuwang@gzhu.edu.cn;
†These authors contributed equally to this work.

Abstract

Large language model (LLM)-based text-to-SQL agents rely on database schemas to identify relevant tables, columns, and join relations. In enterprise databases with fine-grained access control, schema metadata can itself be sensitive. If a complete schema is exposed to components vulnerable to prompt injection or processed by services outside the enterprise trust boundary, unauthorized identifiers and structural relations may be disclosed. Post-generation validation of structured query language (SQL) restricts execution but cannot revoke schema information already observed by a lower-trust component. We propose an authorization-first framework for constructing compact, safe schema context. Table-, column-, and relation-level authorization is enforced before relevance retrieval, so unauthorized objects never enter lower-trust retrieval, reranking, or generation components. Provenance-aware permission inheritance

extends this protection to derived artifacts, including semantic descriptions, join hints, and SQL demonstrations. Within the authorized subspace, permission-safe semantic enrichment improves the discoverability of abbreviated enterprise identifiers, while a satisfiability gate enables non-revealing abstention. On an access-control-augmented Spider benchmark containing 240 question-role pairs over 20 databases and four permission tiers, the framework reduces process-level schema exposure from 35.5 to 0 unauthorized objects per request at identical retrieval utility and reduces unauthorized SQL references from 0.546 to 0.115. A budget-matched retrieve-then-mask baseline produces byte-identical final contexts in 87.1% of requests but exposes the full schema during retrieval. Leakage-provenance analysis further shows that all residual unauthorized SQL references under our framework originate from model hallucination rather than supplied context.

Keywords: Text-to-SQL, large language models, schema security, access control, prompt injection, schema retrieval

1 Introduction

Text-to-SQL translates natural-language questions into structured query language (SQL), lowering the barrier to database querying. With the development of large language models (LLMs), retrieval-augmented generation (RAG), and agent-based techniques, text-to-SQL systems increasingly combine database schemas, data dictionaries, and external tools to perform schema linking, query planning, and SQL generation. Schema information—table names, column names, data types, and foreign-key relations—provides essential evidence for understanding user intent and generating correct SQL [1–3].

Existing methods commonly provide the complete database schema to the model, or retrieve relevant objects from the complete schema space before adding them to the model context. These methods primarily optimize schema recall, execution accuracy, and context efficiency, but rarely treat access authorization as a prerequisite for a schema object to *participate in retrieval*. In enterprise databases with table- and column-level permissions, this design exposes sensitive metadata. Identifiers such as `employee_compensation`, `customer_risk_score`, and `fraud_investigation_result` reveal internal data assets and business processes even without disclosing a single record. Database schemas should therefore not be treated as universally visible auxiliary information, but should instead be governed by access policies consistent with the underlying data objects.

Processing a complete schema *within a fully trusted component* does not by itself constitute information leakage. This paper targets the exposure surface created when unauthorized schema information enters a lower-trust component or crosses an organizational trust boundary. As illustrated in Fig. 1(a), an attacker may use prompt injection, role-playing, or multi-turn probing to induce an agent to reproduce schema information contained in its context. Indirect instructions may be embedded in compromised data dictionaries, schema comments, business documents, or tool responses.

Moreover, when a system invokes external embedding, reranking, or LLM services, the processed schema enters third-party requests, caches, and logs, creating compliance and supply-chain risks [4–6]. We make this assumption explicit because it determines when our contribution is meaningful: **(A1)** The retrieval, reranking, and generation components lie *outside* the trusted boundary. Under A1, any mechanism that filters *after* retrieval cannot achieve process-level confidentiality because the unauthorized metadata has already been transmitted.

Existing defenses are applied mainly after schema retrieval or SQL generation. Post-generation SQL validation and DBMS access control block unauthorized queries from executing, but cannot revoke schema information already observed by the model. A retrieve-then-mask pipeline removes unauthorized candidates from the final prompt, yet those objects have already participated in relevance computation and may have entered external retrieval services or intermediate logs. Pre-generation schema confidentiality and post-generation SQL authorization are therefore *distinct* objectives. Although schema disclosure does not directly cause data exfiltration, it reduces attacker uncertainty, facilitates identification of sensitive objects, and enables targeted follow-up queries. We refer to this as a *schema-reconnaissance-assisted targeted attack*. Based on this observation, we adopt the design principle that *authorization must precede retrieval*, consistent with least privilege and role-based access-control design [7, 8]. As shown in Fig. 1(b), the system first removes unauthorized tables, columns, and structural relations within a trusted boundary, and then performs retrieval, ranking, and context construction exclusively over the authorized schema subspace. Consequently, even if the downstream agent is manipulated, the schema directly available for disclosure is bounded by the current user's authorized scope. SQL validation and DBMS access control are retained as execution-time defenses, forming a defense-in-depth architecture.

Our contributions are as follows:

1. **Pre-generation schema confidentiality framework.** We isolate pre-generation schema confidentiality as a distinct objective from post-generation SQL authorization. We define a threat model for cross-boundary schema processing and introduce a process-level exposure metric. Based on this, we propose an authorization-first context construction mechanism with provenance-aware inheritance, ensuring unauthorized objects and their derived artifacts are removed before any query-dependent retrieval computation.
2. **Empirical evidence for ordering and defense in depth.** Through controlled experiments yielding byte-identical final contexts, we prove that eliminating process-level leakage is attributable to authorization ordering alone at zero utility cost. Furthermore, leakage-provenance analysis demonstrates that context minimization and post-generation validation mitigate *orthogonal* risks: full-schema contexts leak via direct injection, while residual leaks under our framework originate solely from model hallucination, establishing a robust defense-in-depth architecture.
3. **Conditional result on semantic enrichment.** We establish identifier opacity as the precondition for the benefit of semantic enrichment, showing that it yields no measurable gain on standard datasets but significantly improves schema recall and downstream accuracy on systematically abbreviated enterprise-style schemas.

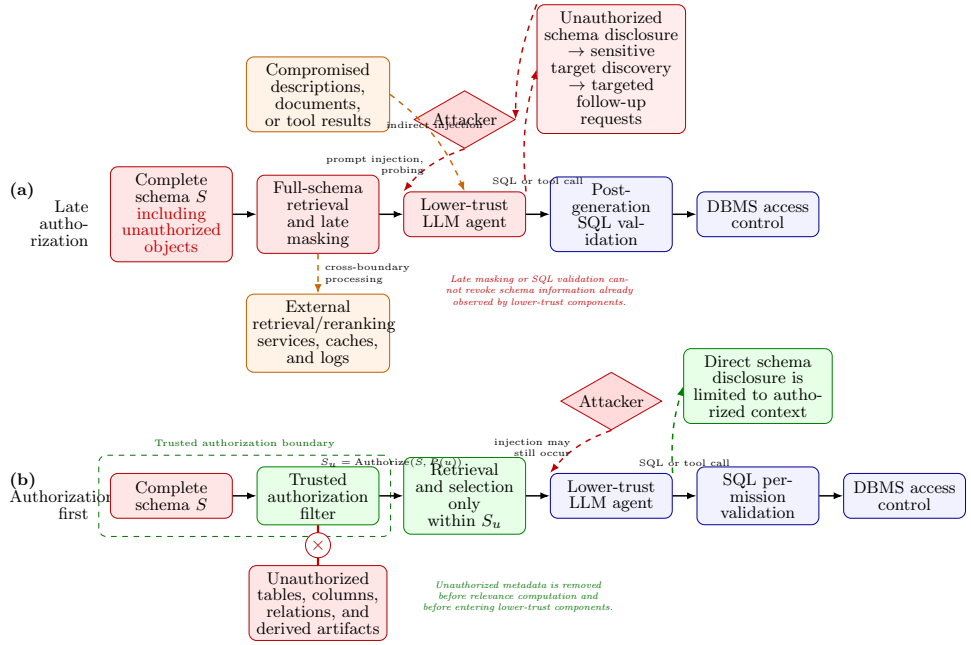


Fig. 1 Threat model and pipeline comparison. (a) In late-authorization pipelines, unauthorized schema objects participate in full-space retrieval and may reach the LLM before masking or SQL validation. Direct or indirect prompt injection can expose this information, and external retrieval services, caches, and logs create an additional cross-boundary exposure surface. (b) The proposed pipeline applies authorization within a trusted boundary before relevance retrieval. Prompt injection may still manipulate the downstream agent, but the schema directly available for disclosure is restricted to the authorized subspace S_u. SQL validation and DBMS access control remain as execution-time defenses.

2 Related Work

2.1 Text-to-SQL and Schema Retrieval

Spider, BIRD, and Spider 2.0 have advanced cross-domain text-to-SQL, complex SQL generation, and schema generalization [1, 2, 9, 10]. Recent surveys and real-world evaluations document the rapid development of LLM-based text-to-SQL systems and the continuing difficulty of grounding queries in large enterprise schemas [3, 11, 12].

As schemas grow, providing all tables and columns increases context length and introduces irrelevant information. Existing systems therefore use schema linking, semantic retrieval, or coarse-to-fine pruning to select objects related to the input question [13–20].

These methods determine which schema objects are *relevant*, optimizing schema recall, execution accuracy, and context size. In an access-controlled setting, however, a highly relevant object may still be ineligible for the current user. Our framework first determines which objects may be *exposed*, and performs relevance-based pruning only within that authorized subspace. Schema pruning is thus a utility and compression component of our framework, not its security boundary.

2.2 Secure Text-to-SQL and Access Control

Classical database access control prevents unauthorized queries at SQL analysis or execution time [21–24]. Existing secure text-to-SQL systems similarly use role conditioning, output validation, refusal, policy synthesis, or schema abstraction to constrain generated SQL [25–30]. Such controls cannot protect schema information already included in the model context. Schema inference, data-leakage, backdoor, and SQL-injection studies further show that hidden tables and columns may be recovered or targeted through adversarial interactions [4, 31–36].

Our abstention design follows the *non-Truman* model of fine-grained access control: rather than silently rewriting a query into an authorized approximation, which may return misleading results, the system refuses. Crucially, the refusal is identical regardless of *why* the request cannot be served, so that refusal behavior cannot be used as a schema-probing oracle.

Authorization in general retrieval-augmented generation is typically applied at the document level. Database schemas additionally contain table–column dependencies, foreign-key relations, and column-level permissions. Descriptions, join hints, and SQL demonstrations may depend on multiple objects simultaneously. We therefore extend authorization from base schema objects to provenance-tracked derived artifacts.

2.3 Secure LLM Agents and Tool-Use Governance

The broader literature on LLM-agent security has shown that tool use changes the risk profile of LLM systems. Tool-augmented agents may be vulnerable to prompt injection, indirect prompt injection, malicious tools, untrusted tool feedback, poisoned external knowledge, time-of-check-to-time-of-use failures, and unsafe multi-tool orchestration [5, 37–42]. Surveys of LLM-agent security emphasize that agentic systems combine model reasoning with delegated authority, persistent state, and external actions, creating risks that cannot be mitigated solely through text filtering [5, 37, 38].

Several recent defenses therefore advocate runtime enforcement, tool-call mediation, provenance tracking, capability restriction, and policy-based access control [39, 43–46]. For example, runtime guardrails enforce constraints at tool-call boundaries; attribute-based access-control frameworks for tool-use agents regulate whether tools can be invoked under given contextual conditions; and verifiable tool-use approaches argue for formalized constraints on data flows and tool sequences [39, 43, 44]. Studies on Model Context Protocol security and secure tool ecosystems further emphasize scoped authorization, provenance, sandboxing, and auditable governance [47].

Our framework follows the same general principle: security should be enforced at the boundary where the LLM's proposed action becomes an external effect. In our case, the relevant tool action is SQL execution. The framework therefore intercepts generated SQL before execution, adjudicates it against user permissions, and permits execution only after deterministic validation. This places secure text-to-SQL within a broader systems-security view of LLM agents.

3 Problem Formulation and Threat Model

3.1 Schema Objects and Permissions

Let q denote a natural-language question and u the current user or role. The set of base schema objects is

$$\mathcal{B} = \mathcal{T} \cup \mathcal{C} \cup \mathcal{R}, \tag{1}$$

where $\mathcal{T}$, $\mathcal{C}$, and $\mathcal{R}$ denote tables, columns, and structural (join) relations. The authorization scope of user u is

$$P(u) = P_{\mathcal{T}}(u) \cup P_{\mathcal{C}}(u) \cup P_{\mathcal{R}}(u) \subseteq \mathcal{B}. \tag{2}$$

Table authorization does not imply access to every column. We require policies to be *well-formed*:

$$c \in P_{\mathcal{C}}(u) \Rightarrow \mathrm{tbl}(c) \in P_{\mathcal{T}}(u), \tag{3}$$

$$r = (t_1, c_1, t_2, c_2) \in P_{\mathcal{R}}(u) \Rightarrow \{t_1, c_1, t_2, c_2\} \subseteq P(u). \tag{4}$$

Equation (4) states that a join relation may be exposed only when both participating tables and both join columns are authorized. In our implementation, $P_{\mathcal{R}}(u)$ is *derived* as the subset of foreign-key constraints satisfying this condition, rather than specified independently.

3.2 Trust Boundary and Attacker

We treat identity management, policy storage, the authorization filter, the intent parser, the satisfiability gate, the SQL validator, and DBMS access control as trusted. Following assumption A1, the user-facing LLM, external embedding and reranking services, agent planning modules, and prompt-related logs are lower-trust components that must not receive information beyond the current user's scope.

The attacker holds a legitimate but restricted account and may submit arbitrary questions, observe responses and errors, apply direct prompt injection or multi-turn probing, and place indirect instructions in content consumed by the agent. The objective is to recover unauthorized table names, columns, types, or relations, and to use that knowledge for sensitive target discovery or targeted SQL and tool-call construction.

3.3 Provenance-Aware Safety Objective

A model context may contain both base objects and derived artifacts. Let $\mathrm{src}(a) \subseteq \mathcal{B}$ be the base-schema provenance of artifact a. For base objects,

$$\mathrm{src}(t) = \{t\}, \quad \mathrm{src}(c) = \{c, \mathrm{tbl}(c)\}, \tag{5}$$

and

$$\mathrm{src}(r) = \{r, t_1, c_1, t_2, c_2\}. \tag{6}$$

Note that exposing a column necessarily exposes its qualifying table name; omitting this dependency would leave a gap in the safety argument.

A context C is safe for user u if every artifact depends exclusively on authorized base objects:

$$\mathrm{Safe}(C, u) \iff \forall a \in C : \ \mathrm{src}(a) \subseteq P(u). \tag{7}$$

We distinguish context confidentiality from SQL authorization:

$$\mathrm{Safe}(C, u) = \mathrm{true}, \tag{8}$$
$$\mathrm{Ref}(\mathrm{SQL}) \subseteq P(u), \tag{9}$$

where Ref(SQL) denotes the base objects referenced by generated SQL. Equation (8) protects what lower-trust components can observe *before* generation; Eq. (9) protects query execution. Section 6.3 shows experimentally that these two conditions are independent: enforcing Eq. (8) perfectly does not imply Eq. (9).

Subject to context safety, the system constructs a compact context covering the schema functions required by the question:

$$\begin{aligned} \min_{C} \quad & \mathrm{Cost}(C) \\ \text{s.t.} \quad & \mathrm{Safe}(C, u), \\ & \mathrm{Cover}(q, C) \geq \eta, \end{aligned} \tag{10}$$

where $\mathrm{Cost}(C)$ is the serialized context length and $\mathrm{Cover}(q, C)$ measures coverage of the required projection, filtering, aggregation, and join functions. Our coarse-to-fine procedure is a practical approximation, not a claim of global optimality.

We also define a *process-level* confidentiality requirement that is strictly stronger than Eq. (8). Let $\mathcal{E} = \{\mathrm{index}, \mathrm{retrieval}, \mathrm{reranking}, \mathrm{prompting}, \mathrm{logging}\}$ be boundary-crossing events and $\mathrm{Obs}(e, u)$ the set of base objects observable by lower-trust components at event e. Then

$$\forall e \in \mathcal{E} : \ \mathrm{Obs}(e, u) \subseteq P(u). \tag{11}$$

Proposition 1 (Necessity of authorization ordering) *Under assumption A1, no mechanism that filters schema objects after relevance computation can satisfy Eq. (11), because* $\mathrm{Obs}(\mathit{retrieval}, u)$ *already includes the complete schema. Enforcing authorization before relevance computation is therefore necessary.*

The proposition is immediate from the definition, but it makes the measurement target explicit: our evaluation reports Eq. (11) (metrics CtxLeak and LeakCount) in addition to the final-prompt condition.

The framework does not address inference from public knowledge, database error messages, or query results, nor statistical disclosure through authorized aggregations.

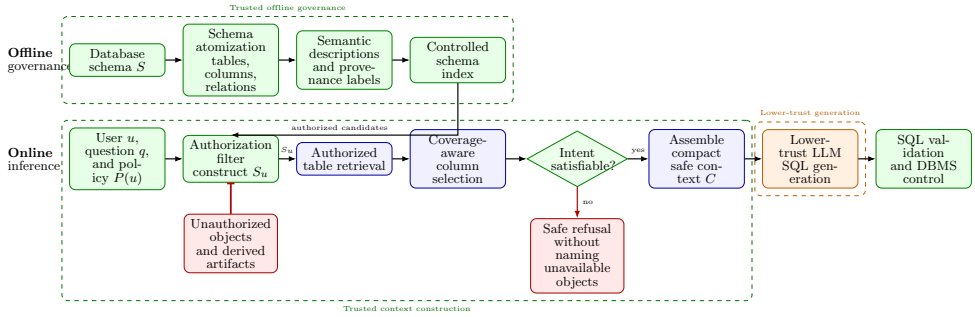


Fig. 2 Authorization-first safe schema context construction. Offline governance creates semantic descriptions and records their base-schema provenance. Online processing constructs the authorized subspace S_u before relevance retrieval. Only authorized objects and derived artifacts can reach the LLM. Requests that cannot be supported within S_u follow a non-revealing refusal path. Retrieval, ranking, and satisfiability analysis all execute inside the trusted boundary.

Algorithm 1 Online safe context construction

Require: question q, user u, schema S, policy $P(u)$, index I, descriptions D, budget K, threshold η

Ensure: safe context C or non-revealing refusal

$S_u \leftarrow \text{Authorize}(S, P(u))$ ▷ trusted; before any use of q

$I_u \leftarrow \{a \in I : \text{src}(a) \subseteq P(u)\}$ ▷ provenance filter

$\widehat{\mathcal{T}}_q \leftarrow \text{TopK}_{t \in \mathcal{T}_u}\, s(q, t; I_u, D)$

$\widehat{\mathcal{C}}_q \leftarrow \text{CoverSelect}(q, \widehat{\mathcal{T}}_q, P_{\mathcal{C}}(u))$

$\widehat{\mathcal{R}}_q \leftarrow \text{JoinSelect}(q, \widehat{\mathcal{T}}_q, \widehat{\mathcal{C}}_q, P_{\mathcal{R}}(u))$

$A_q \leftarrow \{a \in I_u : \text{src}(a) \subseteq \widehat{\mathcal{T}}_q \cup \widehat{\mathcal{C}}_q \cup \widehat{\mathcal{R}}_q\}$

if $\text{Cover}(q, S_u) < \eta$ **then** ▷ evaluated over S_u, not only retrieved objects

  return generic refusal ▷ identical message in all cases

end if

return $C \leftarrow \widehat{\mathcal{T}}_q \cup \widehat{\mathcal{C}}_q \cup \widehat{\mathcal{R}}_q \cup A_q$

4 Authorization-First Safe Context Construction

4.1 Framework Overview

As shown in Fig. 2, the framework consists of offline schema governance and online context construction. Offline governance atomizes the schema, generates semantic descriptions, records provenance, and builds a controlled index. At inference time, the system first constructs a user-specific authorized subspace; table retrieval, column selection, relation selection, and satisfiability analysis are then performed exclusively within that subspace. Algorithm 1 summarizes the online procedure.

4.2 Authorized Schema Subspace

Before any query-dependent relevance computation, the system constructs

$$S_u = \text{Authorize}(S, P(u)). \tag{12}$$

Only objects in S_u are eligible for retrieval, reranking, and context assembly. Unauthorized objects are not assigned low relevance scores; they are absent from the candidate space.

Possible implementations include role-isolated indices, trusted server-side metadata pre-filtering, or request-specific candidate sets generated inside the trusted boundary. Role-isolated indices cost $O(|\text{roles}| \cdot |S|)$ storage and are impractical for large organizations; our implementation therefore uses a single index with pre-filtering applied inside the trusted boundary before any similarity computation.

4.3 Permission Inheritance for Derived Artifacts

For a derived artifact a, visibility is determined by its complete provenance:

$$\text{Visible}(a, u) \iff \text{src}(a) \subseteq P(u). \tag{13}$$

A join hint depending on two tables, two join columns, and a foreign-key relation is visible only if all of these are authorized. An SQL demonstration is checked against every schema object it references.

The same rule applies to semantic descriptions, which is where the risk is most subtle. If the description of an authorized column mentions an unauthorized table—for example, "`order_amt`: transaction value, used together with `customer_risk_score` for fraud triage"—then object-level masking would still leak the hidden identifier through natural-language metadata. We therefore (i) generate each description from the target object and its own table alone, never from the global schema, and (ii) run a post-generation extraction pass that scans the description text for any identifier or alias belonging to another object and adds that object to $\text{src}(a)$.

4.4 Permission-Safe Semantic Enrichment

Enterprise schemas frequently contain abbreviations such as `cust_rsk_scr` and `txn_dt_utc`. Retrieval based only on raw identifiers may fail to associate them with expressions such as "customer risk score" or "transaction time."

For each schema object e, we construct a provenance-tracked description

$$D(e) = \big\langle \text{name}(e), \text{type}(e), \text{meaning}(e), \text{role}(e), \text{aliases}(e) \big\rangle, \tag{14}$$

where meaning is the business meaning, role is the typical projection, filtering, aggregation, or join function, and aliases contains abbreviation expansions and synonyms.

Rather than concatenating $D(e)$ with the identifier text—which may dilute the weight of exact identifier matches—we compute two relevance signals and fuse their

normalized scores:

$$s(q, e) = \alpha\, \tilde{s}_{\mathrm{id}}(q, e) + (1 - \alpha)\, \tilde{s}_{\mathrm{desc}}\big(q, D(e)\big), \tag{15}$$

where $\tilde{s}$ denotes min–max normalization over candidates and α controls the identifier weight. Descriptions therefore act as a fallback when identifier matching fails, without displacing exact matches.

Semantic enrichment improves the discoverability of *authorized* objects. It never reconstructs or substitutes for unauthorized objects: a request requiring an unauthorized object is refused rather than approximated with an unrelated authorized field.

4.5 Coarse-to-Fine Authorized Selection

Table retrieval is performed only over the authorized table set $\mathcal{T}_u$:

$$\widehat{\mathcal{T}}_q = \mathrm{TopK}_{t \in \mathcal{T}_u}\, s(q, t). \tag{16}$$

For multi-table queries, only authorized bridge tables and relations in $P_{\mathcal{R}}(u)$ may be added.

For a selected table t, the column candidate set is

$$\mathcal{C}_u(t) = \mathcal{C}(t) \cap P_{\mathcal{C}}(u). \tag{17}$$

Column relevance may be refined as

$$\begin{aligned} s_{\mathcal{C}}(q, c) = {} & w_1 s_{\mathrm{sem}}(q, c) + w_2 s_{\mathcal{T}}(q, t_c) \\ & + w_3 s_{\mathrm{op}}(q, c) + w_4 s_{\mathrm{key}}(c), \end{aligned} \tag{18}$$

where s_{op} measures compatibility with the implied SQL operation and s_{key} measures structural importance. Because simple top-k selection may omit a filter, grouping field, or join key, columns are selected to cover projection, filtering, aggregation, grouping, ordering, and required joins; structural keys are included only when required by an authorized join path. The evaluated implementation applies selection at the table level and retains all authorized columns of selected tables, which already yields a 3.2× context reduction relative to the full schema. Column-level refinement is an orthogonal compression axis.

4.6 Satisfiability and Safe Abstention

After authorization, a request may fail because the retriever missed an authorized object (a retrieval error) or because a required object is not authorized (which requires abstention).

The satisfiability gate evaluates whether the *authorized subspace* S_u—not merely the selected top-K tables—covers the core entities, output attributes, conditions,

Table 1 Dataset and access-policy statistics.

Role	Description	Avg. visible cols.	Answerable
User_1	Full read access (admin)	72.8	60 / 60
User_2	Department-level tables	37.6	15 / 60
User_3	All tables, masked columns	33.6	16 / 60
User_4	Least privilege	17.6	8 / 60
Total: 20 databases, 240 pairs		—	99 / 240

aggregation inputs, and authorized join paths implied by q. Evaluating over S_u is essential: evaluating over $\widehat{\mathcal{T}}_q$ would conflate retrieval failure with permission insufficiency. When coverage falls below η, the system returns

The request cannot be completed within the current access scope.

The message is byte-identical regardless of the cause, and no object name is revealed, preventing refusal behavior from serving as a schema-probing oracle.

The final context is

$$C = \widehat{\mathcal{T}}_q \cup \widehat{\mathcal{C}}_q \cup \widehat{\mathcal{R}}_q \cup A_q, \tag{19}$$

where A_q contains only artifacts satisfying Eq. (13). Under correct policies and complete provenance labels, $\mathrm{Safe}(C, u)$ holds by construction; we state the assumptions explicitly in Section 7.

5 Experimental Setup

5.1 Dataset and Access Policies

We build on the access-control-augmented Spider corpus of Klisura et al. [25, 26], which extends Spider [9] with role-based `GRANT` policies and records, for each (query, role) pair, the PostgreSQL outcome of executing the reference SQL under that role.

From 153 databases, we select the 20 largest and most permission-sensitive ones, scoring each database by table count, column count, and the presence of potentially sensitive column names (e.g., `salary`, `score`, `address`, and `budget`). For each database, we instantiate four roles reflecting realistic organizational tiers and sample three questions per (database, role) pair, yielding **240 (question, role) pairs**. Table 1 summarizes the resulting policy structure; the average number of visible columns decreases monotonically across tiers.

Relation-level permissions $P_{\mathcal{R}}(u)$ are not provided by `GRANT` statements and are derived as described in Eq. (4).

5.2 Answerability Labels and Validation

A (question, role) pair is *answerable* if all schema objects referenced by the reference SQL are authorized. We parse each reference query with an SQL AST parser and resolve every column to its owning table via schema-aware qualification; `SELECT *` is expanded to all columns of the referenced table, while `COUNT(*)` is correctly treated as referencing no specific column. This yields 99 answerable and 141 unanswerable pairs.

To validate these labels, we compare them against the execution outcomes recorded in the source corpus, treating a successful execution as PERMIT and a `42501 permission denied` error as DENY. The two labelings agree on **240/240 pairs (100.0%)**. This is a conservative approximation with respect to the reference query: an alternative authorized formulation might exist for some pairs labeled unanswerable, a limitation we discuss in Section 7.

5.3 Baselines

All systems share the same generator, prompt template, retriever, and retrieval budget $K = 5$; they differ only in how the schema context is constructed.

- **Full**: the complete schema is supplied.
- **Ret-then-mask**: retrieval over the complete schema, top-K selection, and subsequent removal of unauthorized objects. Part of the retrieval budget is consumed by objects that are later discarded.
- **Ret-then-mask-bm**: a *budget-matched* variant that retrieves over the complete schema but skips unauthorized objects while filling the budget, so that exactly K authorized tables are retained. This is the controlled counterpart to our method: its final context is designed to match ours, isolating authorization ordering as the only difference.
- **Ours-no-desc**: authorization-first retrieval without semantic enrichment.
- **Ours-no-gate**: authorization-first retrieval with semantic enrichment but without the satisfiability gate.
- **Ours**: the complete framework.

5.4 Metrics

Process-level security. CTXLEAK is the fraction of requests in which at least one unauthorized base object crosses a lower-trust boundary—entering retrieval scoring, reranking, or the prompt—rather than merely appearing in the final prompt. LEAKCOUNT is the average number of such objects per request. These metrics operationalize Eq. (11).

Output-level security. SQLLEAK is the fraction of generated queries referencing at least one unauthorized object. For systems with a satisfiability gate, it is computed over non-abstained requests; abstentions are evaluated separately. LKCTX is the fraction of leaking queries whose unauthorized references *appeared in the supplied context*, distinguishing context-derived leakage from hallucinated identifiers that coincidentally match real unauthorized columns.

Utility. RECALL is the recall of authorized schema objects required by the reference query, computed on answerable pairs. SCHEMAF1 is the F1 score between the schema objects referenced by the generated query and those of the reference query. For gated systems, the main table reports SCHEMAF1 over non-abstained requests; a strict end-to-end score that counts abstentions as zero is reported separately. SQLSIM is a token-level Jaccard similarity. CTXRATIO is the number of schema-context tokens divided by the tokens of the *complete authorized* schema, so that a value of 1.0 corresponds to supplying all authorized objects.

Table 2 Main results on access-control-augmented Spider (20 databases, 240 (question, role) pairs, Qwen3.5-35B-A3B, $K = 5$). CtxLeak, LeakCount, Recall, and CtxRatio are deterministic context-level measurements; the remaining columns depend on generation.

	Security			Utility			Abstention	
Method	CtxLeak↓	LeakCount↓	SQLLeak↓	SchemaF1↑	Recall↑	CtxRatio↓	DCR↑	FRR↓
Full	0.75	35.5	0.546	0.831	1.000	2.208	0.00	0.00
Ret-then-mask	0.75	35.5	0.175	0.783	0.881	**0.484**	0.00	0.00
Ret-then-mask-bm	0.75	35.5	0.154	0.809	0.913	0.688	0.00	0.00
Ours-no-desc	0.00	**0.0**	0.145	0.809	0.913	0.686	0.00	0.00
Ours-no-gate	0.00	**0.0**	0.154	0.801	0.907	0.687	0.00	0.00
Ours	0.00	**0.0**	0.115	0.815	0.907	0.687	0.58	0.25

Abstention. DCR is the abstention rate on unanswerable pairs and FRR is the abstention rate on answerable pairs. We attribute abstention *solely* to the satisfiability gate. The generator is instructed always to produce a query; its spontaneous refusal rate is reported separately as a confounder and is 0.00 for the main model.

5.5 Implementation

Retrieval uses BM25 over tokenized identifiers, with snake-case splitting and dictionary-based abbreviation expansion, fused with a description-based BM25 signal according to Eq. (15) with $\alpha = 0.75$. Semantic table descriptions are generated offline for all 236 tables of the selected databases, each conditioned only on its own table to preserve provenance. The satisfiability gate uses keyword coverage over S_u, with η selected by Youden's J on the development sweep. The main generator is Qwen3.5-35B-A3B at temperature 0; Qwen2.5-7B-Instruct is used for a cross-model check. All prompts are cached, and every run records the model identifier, decoding parameters, and random seed.

6 Results

6.1 Main Results

Table 2 reports the main comparison. Three observations stand out.

Process-level exposure is eliminated. All full-schema-retrieval baselines expose an average of 35.5 unauthorized objects per request during retrieval, whereas all authorization-first variants expose none. The aggregate CtxLeak of 0.75 rather than 1.00 reflects the composition of the benchmark: User_1 is a full-access administrator for whom no object is unauthorized. Restricted to the three restricted roles, CtxLeak is exactly 1.00 for every full-schema baseline and 0.00 for every authorization-first variant.

Context minimization—not authorization ordering—drives SQLLeak. SQLLeak drops from 0.546 (Full) to approximately 0.15 for *both* pre- and post-retrieval filtering because their final contexts are equivalent. Comparing Ret-then-mask-bm (0.154) with Ours-no-gate (0.154) confirms that authorization ordering has no effect on output-level leakage. The further reduction to 0.115 under the full framework is attributable to the satisfiability gate, which withholds requests that cannot

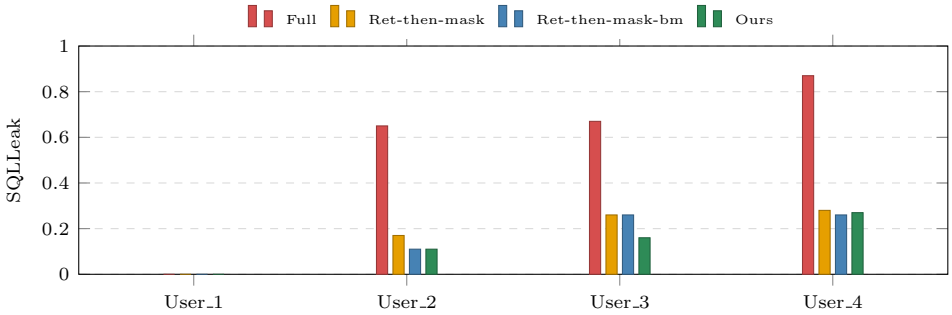


Fig. 3 Unauthorized SQL references by permission tier. Under a full-schema context, leakage grows as privileges shrink, peaking at 0.87 for the least-privileged role. User_1 is a full-access administrator, so its SQLLeak is zero by definition.

be served. We state this explicitly to avoid overclaiming: the benefit of authorization ordering is confined to, but complete within, the process-level objective of Eq. (11).

Full retains the highest generation quality, as expected. Providing the complete schema yields the best SchemaF1 (0.831) and perfect Recall, since no relevant object can be pruned. This comes at a 2.208× context size, complete process-level exposure, and a 4.7× higher SQLLeak. Within a matched retrieval budget, our framework incurs *no* utility loss relative to post-retrieval masking.

Fig. 3 shows the per-role breakdown. Under Full, unauthorized SQL references increase as privileges decrease, reaching 0.87 for the least-privileged role—precisely the population that access control is meant to protect. Our framework reduces this value to 0.27.

6.2 Isolating the Effect of Authorization Ordering

The comparison between Ret-then-mask-bm and Ours-no-desc is the central controlled experiment of this paper. Both retrieve $K = 5$ authorized tables using the same scoring function and produce contexts that are *byte-identical in 209 of 240 requests* (87.1%). The residual 12.9% differ only because restricting the BM25 index to S_u changes the inverse document frequency statistics; aggregate utility is unaffected (Recall 0.913 vs. 0.913; SchemaF1 0.809 vs. 0.809; CtxRatio 0.688 vs. 0.686). We further verified that the two retrieval candidate spaces differ for every restricted-role request; the process-level divergence is therefore universal rather than incidental.

Consequently, the entire gap in process-level exposure—CtxLeak 0.75 → 0.00 and LeakCount 35.5 → 0.0—is attributable to *when* authorization is applied, at zero utility cost. Notably, restricting the retrieval index to the authorized subspace does not degrade ranking quality despite the smaller IDF corpus.

Table 3 shows that this equivalence holds across the entire retrieval-budget range: Ret-then-mask-bm and Ours-no-desc coincide or remain nearly identical at every $K \in \{3, \ldots, 10\}$, while unbudgeted Ret-then-mask is consistently worse because part of its budget is wasted on objects that are subsequently discarded.

Table 3 Retrieval-budget sensitivity, reported as Recall / CtxRatio. Ret-then-mask-bm and Ours-no-desc remain nearly identical across budgets, confirming that authorization ordering is utility-neutral.

K	Full	Ret-then-mask	Ret-then-mask-bm	Ours-no-desc	Ours
3	1.000 / 2.21	0.831 / 0.30	0.831 / 0.45	0.821 / 0.45	0.821 / 0.45
4	1.000 / 2.21	0.852 / 0.39	0.852 / 0.58	0.855 / 0.58	0.855 / 0.59
5	1.000 / 2.21	0.881 / 0.48	0.913 / 0.69	0.913 / 0.69	0.907 / 0.69
6	1.000 / 2.21	0.905 / 0.56	0.931 / 0.76	0.931 / 0.76	0.921 / 0.76
8	1.000 / 2.21	0.918 / 0.74	0.939 / 0.86	0.939 / 0.86	0.936 / 0.86
10	1.000 / 2.21	0.958 / 0.84	0.979 / 0.92	0.979 / 0.92	0.979 / 0.92

Table 4 Provenance of unauthorized SQL references. Under a full-schema context, every leak is copied from the context; under authorization-first construction, every residual leak is a hallucinated identifier that coincidentally matches a real unauthorized column.

Method	Leaking queries	From context	Hallucinated
Full	131	131 (100%)	0 (0%)
Ret-then-mask-bm	35	0 (0%)	35 (100%)
Ours-no-desc	33	0 (0%)	33 (100%)
Ours	15	0 (0%)	15 (100%)

6.3 Where Does Residual SQL Leakage Come From?

A natural question is why SQLLeak is nonzero (0.115) for our framework when no unauthorized object ever enters the context. We answer it by tracing the provenance of every leaking reference (Table 4).

Under Full, all 131 leaking queries copy identifiers directly from the supplied schema. Under every context-minimizing method, the corresponding figure is exactly zero: the model invents plausible identifiers—for instance, producing `zip_postcode` when the context contains only `address_id`—and these occasionally coincide with genuine unauthorized columns.

This constitutes direct empirical evidence that pre-generation context minimization and post-generation SQL validation address *orthogonal* risks. Context minimization eliminates leakage of *knowledge the model was given*; it cannot eliminate leakage of *knowledge the model guesses*. Neither mechanism subsumes the other, which is precisely the defense-in-depth argument of Section 3.3, here validated rather than merely asserted.

6.4 Satisfiability Gate: Detection–Precision Trade-off

The gate exposes a single threshold η that trades correct refusals against false refusals. Fig. 4 plots the full sweep. At the operating point selected by Youden's J ($\eta = 0.5$), the gate detects 58% of unanswerable requests at a 25% false-refusal rate. A more conservative setting ($\eta = 0.30$) yields DCR 0.40 at FRR 0.12.

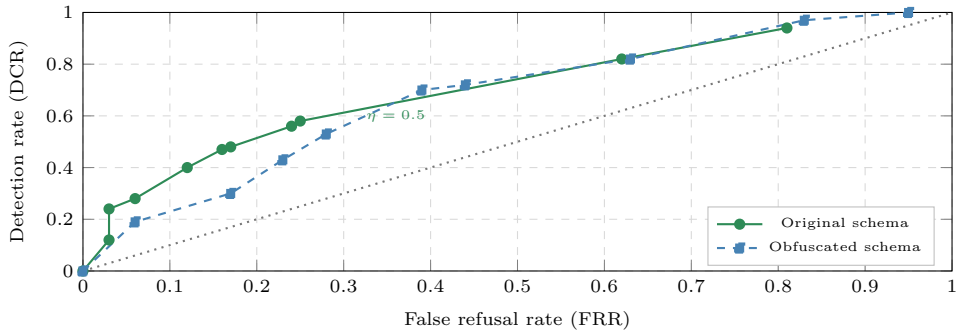


Fig. 4 Satisfiability-gate operating characteristic. Each point is a threshold $\eta \in [0, 0.7]$. The dotted diagonal marks chance. The selected operating point achieves DCR 0.58 at FRR 0.25 on the original schema.

The gate is deliberately simple—keyword coverage over S_u—and its absolute performance is modest. Its value in this work is architectural: it demonstrates that abstention can be decided *inside the trusted boundary*, before the schema reaches the generator, and with a refusal message that is invariant to the cause. Under strict scoring in which gated abstentions are counted as zero, our full system's SchemaF1 falls from 0.801 (Ours-no-gate) to 0.576; this is the explicit price of the detection rate, and the threshold provides a tunable operating point.

6.5 Semantic Enrichment Is Conditional on Identifier Opacity

On the original Spider schema, semantic enrichment produces no measurable benefit: Recall changes from 0.913 (Ours-no-desc) to 0.907 (Ours-no-gate), well within run-to-run variation. This is a negative result, and we attribute it to a property of the benchmark rather than the mechanism: Spider identifiers are already natural-language words such as `asset_model` and `monthly_rental`, so lexical matching against the question already succeeds and description text may dilute the signal.

To test this attribution, we construct an *obfuscated* variant of the same 20 databases by systematically abbreviating table and column names (`customer`→`cust`, `maintenance`→`mnt`, `staff`→`stf`, etc.), retaining the original name whenever an abbreviation would create a collision. Policies, answerability labels, and reference-query object sets are mapped consistently, so the two settings differ *only* in identifier surface form.

The result is unambiguous (Table 5 and Fig. 5). Obfuscation reduces retrieval recall for the enrichment-free variant from 0.913 to 0.770, confirming that opaque identifiers genuinely break lexical matching. Semantic enrichment recovers most of this loss, raising recall to 0.885 (+11.5 pp), and the gain propagates downstream to SchemaF1 (0.741 → 0.807, +6.6 pp). The improvement is stable across the entire retrieval-budget range, from +13.4 pp at $K = 4$ to +4.3 pp at $K = 10$, where the budget is large enough that retrieval is easy for both variants.

Table 5 Effect of semantic enrichment under natural versus abbreviated identifiers ($K = 5$). Enrichment is beneficial only when identifiers are opaque.

Schema	Method	Recall↑	SchemaF1↑	SQLLeak↓
Original	Ours-no-desc	0.913	0.809	0.145
	Ours-no-gate	0.907	0.801	0.154
	Δ	−0.006	−0.008	—
Obfuscated	Ours-no-desc	0.770	0.741	0.055
	Ours-no-gate	0.885	**0.807**	0.065
	Δ	+0.115	+0.066	—

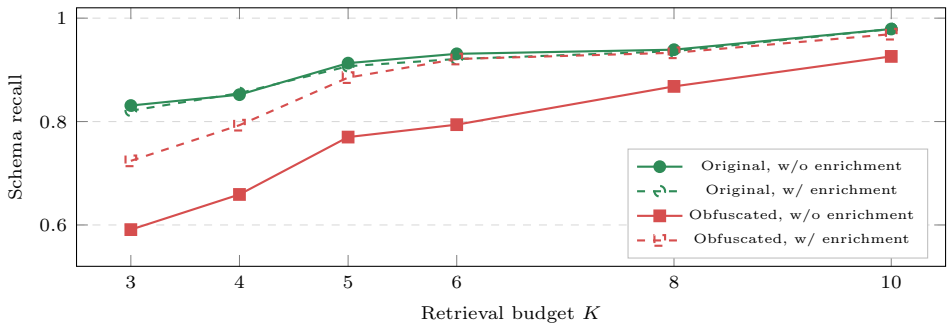


Fig. 5 Semantic enrichment yields no gain when identifiers are already natural-language words (the two upper curves overlap), but recovers most of the recall lost to abbreviation (the two lower curves), with gains of +4.3 to +13.4 pp across the budget range.

6.6 Prompt-Injection Case Study

We evaluate whether an adversary can extract unauthorized schema information from a manipulated generator. Twenty (database, role) cases spanning seven databases and three restricted roles are attacked with two strategies: a *direct* request to enumerate all tables and columns, and a *role-play* attack asserting that access control has been disabled for maintenance. Each case is run under three context configurations: our framework, a full schema, and a *no-context* control in which the schema placeholder is empty. The no-context control is essential because Spider is a public benchmark whose schemas the model may have memorized; it establishes how much of the recovered information comes from the context rather than from parametric knowledge.

Table 6 reports the results. Under a full schema, 40–55% of attacks cause the model to enumerate unauthorized tables, recovering up to 23.9% of the hidden schema; the per-database breakdown shows leakage in six of seven databases, so the effect is not driven by a single outlier. Under our framework, extraction is near zero.

Two measurement caveats deserve emphasis. First, on the original schema the no-context control is also nonzero (0.10), and manual inspection shows that both this result and the residual 0.10 under our framework stem from the tokens `addresses` and `course` appearing in explanatory prose rather than in a schema listing—string

Table 6 Prompt-injection case study (20 cases across 7 databases and 3 restricted roles). EXTRACT is the fraction of attacks revealing at least one unauthorized table; RECOVERY is the fraction of unauthorized tables recovered.

		Original schema		Obfuscated schema	
Attack	Context	Extract↓	Recovery↓	Extract↓	Recovery↓
Direct	No-context	0.10†	0.017	0.00	0.000
	Full	0.40	0.089	0.55	0.239
	Ours	0.10†	0.017	0.00	0.000
Role-play	No-context	0.10†	0.017	0.00	0.000
	Full	0.40	0.089	0.40	0.089
	Ours	0.00	0.000	0.00	0.000

† Manual inspection attributes these matches to common English words (`addresses`, `course`) appearing in prose rather than schema enumeration.

Table 7 Cross-model check on the original schema. Security ordering is preserved under a substantially weaker generator.

Model	Method	CtxLeak↓	LeakCount↓	SQLLeak↓	SchemaF1↑
Qwen3.5-35B	Full	0.75	35.5	0.546	0.831
	Ret-then-mask-bm	0.75	35.5	0.154	0.809
	Ours	0.00	0.0	0.115	0.815
Qwen2.5-7B	Full	0.75	35.5	0.523	0.488
	Ret-then-mask-bm	0.75	35.5	0.135	0.473
	Ours	0.00	0.0	0.069	0.454

matching on common English words therefore overcounts. Second, the obfuscated setting is immune to this artifact because abbreviated identifiers such as `stu_addr` and `mnt_engi` are not English words, and there both the control and our framework are exactly 0.00, while Full rises to 0.55. We therefore regard the obfuscated result as the more reliable measurement and report both for transparency.

6.7 Cross-Model Robustness

To check that the conclusions are not specific to one generator, we repeat the evaluation with Qwen2.5-7B-Instruct (Table 7). The smaller model produces substantially lower generation quality—parse failures rise from 4% to 29% owing to repetition-based decoding degeneration—but the security ordering is preserved: process-level exposure remains 35.5 → 0.0, SQLLEAK falls from 0.523 to 0.069, and leakage provenance remains 100% context-derived for Full and 100% hallucinated for all authorization-first variants. The security properties of the framework are structural and do not depend on generator capability.

6.8 Summary of Findings

- Authorization-first construction eliminates process-level schema exposure entirely (35.5 → 0.0 objects per request) at *zero* utility cost relative to a budget-matched masking baseline, as verified across all retrieval budgets.
- Context minimization reduces unauthorized SQL references by 4.7×, but this effect comes from the smaller context, not from authorization ordering; the two contributions are separable and are reported separately.
- All residual unauthorized references under our framework are hallucinated rather than context-derived, empirically establishing that context minimization cannot replace SQL validation.
- Semantic enrichment helps only when identifiers are opaque, with a +11.5 pp recall gain under enterprise-style abbreviation and no measurable gain under Spider's natural-language identifiers.
- Prompt-injection extraction drops from 0.40–0.55 to 0.00 in the obfuscated setting, with a no-context control confirming that the reduction reflects context minimization rather than model reticence.

7 Discussion and Limitations

7.1 What the Safety Property Does and Does Not Guarantee

The context safety property of Eq. (7) holds by construction under three assumptions: **(A1)** the access policy correctly reflects the intended authorization; **(A2)** provenance labels are complete, so that every derived artifact records all base objects it may reveal; and **(A3)** the trusted components are not themselves compromised. If a generated description mentions another schema object but the dependency is not recorded, indirect leakage remains possible. Automatically generated descriptions should therefore be reviewed as part of metadata governance. Our implementation mitigates this risk by generating each description in isolation and by running an identifier-extraction pass over the generated text, but neither measure constitutes a formal proof.

The framework is an information-minimization layer, not a complete confidentiality mechanism. An attacker may still infer hidden objects from public business knowledge, common naming conventions, database error messages, timing, or query results. As Section 6.3 shows, hallucinated identifiers can coincide with real unauthorized columns independently of anything the system disclosed. Correct SQL validation, tool permissions, and DBMS access control therefore remain necessary.

7.2 A Side Channel Worth Noting

Our validation procedure surfaced a side channel that is orthogonal to context construction but relevant to the same threat model. PostgreSQL reports a column-level permission failure as `permission denied for table T`, whereas a reference to a nonexistent column yields `42703 column does not exist`. An attacker who can observe raw error codes can therefore distinguish "this object exists but you may not see it" from "this object does not exist," which is exactly the distinction that our

non-revealing abstention is designed to prevent at the application layer. Deployments should normalize database error messages before returning them to users.

7.3 Evaluation Limitations

Answerability is defined with respect to the reference query: a pair is labeled unanswerable if any object referenced by the gold SQL is unauthorized. An alternative authorized formulation may exist in some cases, so this is a conservative approximation. It agrees with execution-based ground truth on all 240 pairs, but agreement with execution does not establish the nonexistence of alternative formulations.

We report schema-level correctness (SchemaF1) rather than execution accuracy because we do not execute generated queries against live databases. Execution-based evaluation would strengthen the utility measurement and is left to future work.

The evaluation covers 20 databases and 240 (question, role) pairs from Spider with synthetically generated `GRANT` policies. Spider is a public benchmark whose schemas may be partially memorized; the no-context control in the prompt-injection study quantifies but does not eliminate this concern. Real enterprise authorization structures involve dynamic policy changes, hierarchical roles, row-level predicates, and multi-tenant cache isolation, none of which are modeled here. Extending the evaluation to BIRD and non-public schemas remains future work.

Finally, the satisfiability gate is a keyword-coverage heuristic. Its operating point (DCR 0.58 at FRR 0.25) is usable but far from ideal, and a semantically grounded intent-slot analysis would likely improve the trade-off substantially.

8 Conclusion

This paper studies pre-generation schema exposure in LLM-based text-to-SQL agents. When a complete database schema enters a component that may be manipulated through prompt injection or observed across an organizational trust boundary, unauthorized metadata can support schema reconnaissance and targeted query construction. Post-generation SQL validation restricts execution but cannot revoke context already observed by lower-trust components.

We proposed an authorization-first framework that constructs a user-specific authorized schema subspace before relevance retrieval, extends authorization to schema-derived artifacts through provenance-aware inheritance, and abstains non-revealingly when the authorized subspace cannot support a request. On an access-control-augmented Spider benchmark, the framework eliminates process-level schema exposure entirely at zero utility cost relative to a budget-matched post-retrieval masking baseline, and reduces unauthorized SQL references by 4.7×. A leakage-provenance analysis shows that every residual unauthorized reference originates from model hallucination rather than from the supplied context, providing direct evidence that information minimization and post-generation validation mitigate orthogonal risks.

The framework complements rather than replaces SQL validation and DBMS access control, introducing an information-minimization boundary in front of potentially manipulable generation components. Future work includes execution-based

evaluation, semantically grounded satisfiability analysis, dynamic policy handling, and evaluation on non-public enterprise schemas.

Statements and Declarations

Funding

This work was partially supported by National Natural Science Foundation of China (U25A20543).

Competing Interests

The authors declare that they have no competing interests.

Author Contributions

All authors contributed to the study conception and design, analysis and interpretation of the results, and preparation and revision of the manuscript. All authors reviewed and approved the final manuscript.

Data Availability

The datasets used and/or analysed during the current study are available from the corresponding author on reasonable request.

References

[1] Lei, F., Chen, J., Ye, Y., Cao, R., Shin, D., Su, H., Yu, T.: Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. in *Proc. Int. Conf. Learn. Representations (ICLR)* (2025)

[2] Gao, D., et al.: Text-to-SQL empowered by large language models: A benchmark evaluation. *Proc. VLDB Endow.*, vol. 17, no. 5, pp. 1132–1145 (2024)

[3] Shi, L., Tang, Z., Zhang, N., Zhang, X., Yang, Z.: A survey on employing large language models for text-to-SQL tasks. *ACM Comput. Surv.*, vol. 58, no. 2, pp. 1–37 (2025)

[4] Motlagh, F.N., Hajizadeh, M., Majd, M., Najafi, P., Cheng, F., Meinel, C.: When prompts become payloads: A framework for mitigating SQL injection attacks in large language model-driven applications. arXiv preprint arXiv:2605.10176 (2026)

[5] He, F., Zhu, T., Ye, D., Liu, B., Zhou, W., Yu, P.S.: The emerged security and privacy of LLM agents: A survey with case studies. *ACM Comput. Surv.*, vol. 58, no. 6, pp. 1–36 (2025)

[6] OWASP Foundation: OWASP Top 10 for large language model applications (2023)

[7] Saltzer, J.H., Schroeder, M.D.: The protection of information in computer systems. *Proc. IEEE*, vol. 63, no. 9, pp. 1278–1308 (1975)

[8] Sandhu, R.S.: Rationale for the RBAC96 family of access control models. in *Proc. ACM RBAC Workshop*, 1995, pp. 11–18 (1995)

[9] Yu, T., et al.: Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. in *Proc. EMNLP*, 2018, pp. 3911–3921 (2018)

[10] Li, J., et al.: Can LLM already serve as a database interface? A big bench for large-scale database-grounded text-to-SQLs. in *Proc. NeurIPS* (2023)

[11] Huang, Y., Guo, J., Mao, W., Gao, C., Han, P., Liu, C., Ling, Q.: Exploring the landscape of text-to-SQL with large language models: Progress, challenges, and opportunities. arXiv preprint arXiv:2505.23838 (2025)

[12] Nascimento, E., García, G.M., Feijó, L., Victorio, W., Izquierdo, Y.T., Oliveira, A.R., Casanova, M.A.: Text-to-SQL meets the real world. in *Proc. Int. Conf. Enterprise Inf. Syst. (ICEIS)*, vol. 1, 2024, pp. 61–72 (2024)

[13] Li, H., Zhang, J., Liu, H., Fan, J., Zhang, X., Zhu, J., Chen, H.: Codes: Towards building open-source language models for text-to-SQL. *Proc. ACM Manag. Data*, vol. 2, no. 3, pp. 1–28 (2024)

[14] Chang, S., Wang, J., Dong, M., Pan, L., Zhu, H., Li, A.H., Xiang, B.: Dr. Spider: A diagnostic evaluation benchmark towards text-to-SQL robustness. arXiv preprint arXiv:2301.08881 (2023)

[15] Lan, W., Wang, Z., Chauhan, A., Zhu, H., Li, A., Guo, J., Xiang, B.: Unite: A unified benchmark for text-to-SQL evaluation. arXiv preprint arXiv:2305.16265 (2023)

[16] Zhang, B., Ye, Y., Du, G., Hu, X., Li, Z., Liu, C.H., Mao, H.: SQLBench: A comprehensive evaluation for text-to-SQL capabilities of large language models. arXiv preprint arXiv:2403.02951 (2024)

[17] Chen, P.B., Yang, D., Li, W., Wenz, F., Zhang, Y., Tatbul, N., Stonebraker, M.: BEAVER: An enterprise benchmark for text-to-SQL. arXiv preprint arXiv:2409.02038 (2024)

[18] Shen, Z., et al.: Improving retrieval-augmented text-to-SQL with AST-based ranking and schema pruning. in *Proc. EMNLP*, 2024, pp. 7865–7879 (2024)

[19] Wu, D., et al.: SchemaRAG: A schema-aware retrieval-augmented generation

framework for text-to-SQL. *Proc. ACM Manag. Data*, vol. 4, no. 1, pp. 1–26 (2026)

[20] Bui, C.D., Nguyen, H.H., Ngo, T.Q., Vu-Thi, H.K., Nguyen, C.H., Nguyen, D.V., Ngo, S.T.: A systematic survey of LLM-based text-to-SQL: Methodologies, security vulnerabilities, and future challenges. *PeerJ Comput. Sci.*, vol. 12, Art. no. e3773 (2026)

[21] Shay, R., Blumenthal, U., Gadepally, V., Hamlin, A., Mitchell, J.D., Cunningham, R.K.: Don't even ask: Database access control through query control. *ACM SIGMOD Rec.*, vol. 47, no. 3, pp. 17–22 (2019)

[22] Rizvi, S., Mendelzon, A., Sudarshan, S., Roy, P.: Extending query rewriting techniques for fine-grained access control. in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, 2004, pp. 551–562 (2004)

[23] Jarman, J., McCart, J.A., Berndt, D., Ligatti, J.: A dynamic query-rewriting mechanism for role-based access control in databases (2008)

[24] Zhang, X.M.: Designing a SQL query rewriter to enforce database row-level security. Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, USA (2016)

[25] Klisura, D., Khoury, J., Kundu, A., Krishnan, R., Rios, A.: Role-conditioned refusals: Evaluating access control reasoning in large language models. in *Findings ACL: EACL*, 2026, pp. 6018–6034 (2026)

[26] Sullutrone, G., Sala, L., Trigiante, L., Bergamaschi, S.: Text-to-refused-SQL: A comprehensive evaluation of LLM refusal in text-to-SQL. in *Proc. SEBD*, 2025, pp. 637–651 (2025)

[27] Subramaniam, P., Krishnan, S.: DePLOI: Applying NL2SQL to synthesize and audit database access control. arXiv preprint arXiv:2402.07332 (2024)

[28] Parte, M.S.E.D.L., Wang, Y., Martínez-Ortega, J.F., Martínez, N.L.: A secure and efficient LLM-based system for natural language query-to-SQL translation in IoT data management. *IEEE Access*, vol. 14, pp. 28435–28456 (2026)

[29] Abedini, S., Mohapatra, S., Emerson, D.B., Shafieinejad, M., Cresswell, J.C., He, X.: MaskSQL: Safeguarding privacy for LLM-based text-to-SQL via abstraction. arXiv preprint arXiv:2509.23459 (2025)

[30] Khatick, J.P., Mandal, S.K.: Preserving database schema privacy while generating SQL queries using generative AI. *ICTACT J. Soft Comput.*, vol. 15, no. 3 (2025)

[31] Song, Y., Liu, R., Chen, S., Ren, Q., Zhang, Y., Yu, Y.: SecureSQL: Evaluating data leakage of large language models as natural language interfaces to databases.

in *Findings ACL: EMNLP*, 2024, pp. 5975–5990 (2024)

[32] Peng, X., Zhang, Y., Yang, J., Stevenson, M.: On the vulnerabilities of text-to-SQL models. in *Proc. IEEE 34th Int. Symp. Softw. Reliab. Eng. (ISSRE)*, 2023, pp. 1–12 (2023)

[33] Lin, M., Zhang, H., Lao, J., Li, R., Zhou, Y., Yang, C., Tang, M.: Are your LLM-based text-to-SQL models secure? Exploring SQL injection via backdoor attacks. *Proc. ACM Manag. Data*, vol. 3, no. 6, pp. 1–27 (2025)

[34] Kim, E., Lee, S.: SQL injection in LLM-generated queries: Systematic analysis of detection gaps and security risks. *IEEE Access* (2026)

[35] Chafik, S., Ezzini, S., Berrada, I.: Enhancing security in text-to-SQL systems: A novel dataset and agent-based framework. *Nat. Lang. Process.*, vol. 31, no. 6, pp. 1399–1422 (2025)

[36] Klisura, D., Rios, A.: Unmasking database vulnerabilities: Zero-knowledge schema inference attacks in text-to-SQL systems. in *Findings NAACL*, 2025, pp. 6954–6976 (2025)

[37] Gan, Y., Yang, Y., Ma, Z., He, P., Zeng, R., Wang, Y., Ji, S.: Navigating the risks: A survey of security and privacy threats in LLM-based agents. *ACM Trans. Softw. Eng. Methodol.* (2026)

[38] Ling, Y., Yu, S., Chen, Z., Fang, C.: Toward secure LLM agents: Threat surfaces, attacks, defenses, and evaluation. arXiv preprint arXiv:2606.10749 (2026)

[39] Zhao, W., Li, Z., Zhang, P., Sun, J.: ClawGuard: A runtime security framework for tool-augmented LLM agents against indirect prompt injection. arXiv preprint arXiv:2604.11790 (2026)

[40] Alpay, F., Alpay, T.: AgentSecBench: Measuring prompt injection, privacy leakage, and tool-use integrity in LLM agents. arXiv preprint arXiv:2605.26269 (2026)

[41] Kereopa-Yorke, B., Diaz, G., Wright, H., Johnston, R., Rosario, R.F.D., Lynar, T.: Oracle poisoning: Corrupting knowledge graphs to weaponise AI agent reasoning. arXiv preprint arXiv:2605.09822 (2026)

[42] Xu, H., Li, C., Ma, X., Ou, X., Zhang, Z., He, T., Qin, B.: The evolution of tool use in LLM agents: From single-tool call to multi-tool orchestration. arXiv preprint arXiv:2603.22862 (2026)

[43] Luo, J., Peng, S., Dai, J., Chen, Z., Shen, Z., Hong, G., Yang, M.: AgentGuard: An attribute-based access control framework for tool-use LLM-based agents. arXiv preprint arXiv:2605.28071 (2026)

[44] Doshi, A., Hong, Y., Xu, C., Kang, E., Kapravelos, A., Kästner, C.: Towards verifiably safe tool use for LLM agents. in *Proc. IEEE/ACM 48th Int. Conf. Softw. Eng. (ICSE)*, 2026, pp. 201–205 (2026)

[45] Wang, Y., Zhang, J., Cai, T., Liu, Z., Sun, Q., Sun, Z., Zhu, Y.: From agent traces to trust: A survey of evidence tracing and execution provenance in LLM agents. arXiv preprint arXiv:2606.04990 (2026)

[46] Helou, M.E., Ryder, B., Troiani, C., Diaconu, J., Muyal, H., Yannuzzi, M.: Hybrid inspection and task-based access control in zero-trust agentic AI. arXiv preprint arXiv:2605.02682 (2026)

[47] Errico, H., Ngiam, J., Sojan, S.: Securing the Model Context Protocol (MCP): Risks, controls, and governance. arXiv preprint arXiv:2511.20920 (2025)