

Benchmarking Text-to-SQL under Role-Based Access Control

YANG FEI, National University of Singapore, Singapore

YANGFAN JIANG, National University of Singapore, Singapore

YIN YANG, Hamad Bin Khalifa University, Qatar

XIAOKUI XIAO, National University of Singapore, Singapore

Given a database $\mathcal{S}$ and a natural language question Q , text-to-SQL systems aim to generate an SQL query that correctly answers Q when executed against $\mathcal{S}$. Currently, popular text-to-SQL benchmarks mostly assume unrestricted access to $\mathcal{S}$; in practice, however, user access is often restricted, e.g., through *role-based access control* (RBAC) policies. This leads to a potential disconnect between benchmarking results and real-world performance: an LLM with high benchmark scores might perform poorly in an access-controlled environment, by frequently violating RBAC, or rejecting a query q that could be answered with only permitted data in $\mathcal{S}$.

Motivated by this, we present a comprehensive text-to-SQL benchmarking framework with realistic RBAC constraints, which features an LLM-assisted workflow that augments existing text-to-SQL benchmarks with plausible user roles and access policies. To do so, we formulate the problem of role synthesis as a structured reasoning process over the database schema, in which the LLM first infers the application context from the schema, and then derives role responsibilities and access scopes consistent with this context. This process is audited by human-in-the-loop quality control, in which domain experts perform metric-guided screening on the generated roles. Besides the augmented dataset, the proposed framework also contains evaluation metrics that identify RBAC-specific failure modes, and disentangle SQL utility from access-control compliance.

We apply the proposed framework to several widely-used benchmarks, and conduct a systematic empirical study of state-of-the-art text-to-SQL systems. The results show that many solutions (especially open-weight LLMs) with high benchmarking scores under an unrestricted setting suffer sharp performance degradation once access constraints are in place, due to frequent RBAC violations.

CCS Concepts: • **Security and privacy** → **Access control**; • **Information systems** → **Relational database query languages**; • **Computing methodologies** → *Natural language processing*.

Additional Key Words and Phrases: Text-to-SQL, role-based access control, large language models

ACM Reference Format:

Yang Fei, Yangfan Jiang, Yin Yang, and Xiaokui Xiao. 2026. Benchmarking Text-to-SQL under Role-Based Access Control. *Proc. ACM Manag. Data* 4, 4 (SIGMOD), Article 260 (September 2026), 24 pages. <https://doi.org/10.1145/3837098>

1 Introduction

Structured query language (SQL) has long served as the *lingua franca* for database interactions. However, for many users, the complexity of SQL remains a significant barrier to effective data access [22, 26]. Text-to-SQL systems aim to eliminate this barrier by allowing users to express their intent in natural language and translating it into SQL. Recent advances in large language models (LLMs), with strong coding and reasoning capabilities, have enabled substantial progress in text-to-SQL performance [5, 16, 28, 32, 47].

Authors' Contact Information: Yang Fei, National University of Singapore, Singapore, Singapore, yfei11@u.nus.edu; Yangfan Jiang, National University of Singapore, Singapore, Singapore, jyangfan@u.nus.edu; Yin Yang, Hamad Bin Khalifa University, Doha, Qatar, yyang@hbku.edu.qa; Xiaokui Xiao, National University of Singapore, Singapore, Singapore, xkxiao@nus.edu.sg.



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

© 2026 Copyright held by the owner/author(s).

ACM 2836-6573/2026/9-ART260

<https://doi.org/10.1145/3837098>

State-of-the-art LLM-based text-to-SQL solutions typically combine prompt engineering and schema-aware reasoning to achieve high execution accuracy [54]. To evaluate and compare different approaches, several benchmarks have been established, including Spider [24, 51], BIRD [28], and LiveSQLBench [3]. On these benchmarks, recent systems report strong performance under the assumption that generated SQL queries are executed directly by the DBMS without additional constraints.

1.1 Text-to-SQL in the Wild

In real-world DBMS environments, database access is rarely unrestricted. To satisfy security, privacy, and regulatory requirements, production DBMSs enforce *access control policies* that determine which data a user is authorized to access. One of the most widely adopted mechanisms is role-based access control (RBAC), under which users are assigned roles with permissions over specific tables, attributes, operations, or views [13, 38]. In such settings, an unauthorized SQL query could trigger an execution error at the DBMS layer (e.g., MySQL’s error 1142¹), or silent result suppressions (e.g., Snowflake’s dynamic data masking²) that are non-trivial to debug.

This exposes a fundamental gap between real-world deployments and current text-to-SQL evaluation practices. RBAC enforcement should remain deterministic at the DBMS layer; however, modern text-to-SQL systems typically generate SQL outside the DBMS and must therefore make access-sensitive decisions before execution. In practice, this separation can lead to repeated execution failures, brittle system behavior, and subtle mismatches between user intent and the query results returned to the user, often requiring costly manual intervention. A DBMS-side checker can reject unauthorized SQL at execution time, but by that point the upstream text-to-SQL workflow may already have failed to produce a usable response.

To illustrate this issue, consider the following example.

EXAMPLE 1. *Consider an enterprise analytics application in which a text-to-SQL framework translates user requests into SQL queries executed by a backend DBMS. Suppose a user with the “Sales Analyst” role issues the natural language query: “Which departments have the highest average sales revenue per employee?” A vanilla text-to-SQL system may generate the following SQL query:*

```
SELECT e.department, AVG(s.amount) AS avg_revenue
FROM employees e
JOIN sales s ON e.emp_id = s.emp_id
GROUP BY e.department;
```

Under a typical RBAC policy, however, the “Sales Analyst” role may be permitted to access the sales table but not sensitive employee information such as identifiers or department assignments. Although the generated query is syntactically valid and semantically aligned with the user’s intent, it violates the RBAC policy and is rejected by the DBMS at execution time. As a result, even if a deterministic checker blocks the query, the application-level workflow fails to complete: the text-to-SQL component neither produces an authorized query nor correctly refuses the request before execution. The system is therefore safe at the DBMS layer but not useful at the application layer, leaving the request unresolved and requiring retries or manual intervention. More generally, such a checker only determines whether a generated SQL query is authorized; it does not reveal whether the system should have refused earlier, whether it missed a policy-compliant SQL alternative, or whether it over-refused an answerable request.

Despite the practical importance of access control, most existing text-to-SQL benchmarks still focus on translation accuracy under unrestricted database access. While a few recent efforts [23, 41,

¹<https://dev.mysql.com/doc/refman/9.5/en/grant.html>

²<https://docs.snowflake.com/en/user-guide/security-column-ddm-intro>

[45] have begun to consider access control, they mainly target coarse-grained settings (e.g., database- or table-level permissions) and lack systematic benchmarking pipelines or comprehensive empirical analysis, offering limited insight into practical text-to-SQL deployments under RBAC. To our knowledge, established benchmarks and evaluation protocols [3, 5, 6, 9, 15, 16, 28, 32, 36, 47, 48, 51, 52] do not account for RBAC. This leaves an overlooked failure mode that we term an *RBAC-rejected success*, where a generated SQL query is syntactically correct and would be judged correct under unrestricted-access evaluation, yet is rejected at execution time due to RBAC violations. Since existing benchmarks assume unrestricted access, standard metrics such as execution accuracy (EX) fail to capture such failures. As a result, current benchmarks and metrics do not measure a text-to-SQL system’s ability to generate authorized SQL queries under realistic access control constraints, leaving a practically important class of errors largely invisible.

As we shall demonstrate in Section 5, RBAC-rejected successes are in fact common. Systems that achieve high EX scores on existing benchmarks may frequently fail once RBAC rules are enforced, since simply prompting LLMs with access policies does not reliably prevent unauthorized SQL generation. Moreover, heuristic remedies such as schema filtering, prompt engineering, and fine-tuning are insufficient; LLMs may still hallucinate relationships or make flawed assumptions, leading to invalid SQL and failed executions. These observations indicate that RBAC alignment must be treated as a first-class concern in text-to-SQL systems rather than an afterthought handled solely at execution time. Accordingly, a critical mismatch remains between the strong performance reported on existing benchmarks and the requirements of practical, access-controlled deployments in real DBMS environments.

1.2 Contributions

To fill this gap, we present a comprehensive benchmarking framework for evaluating the RBAC-enforcing capabilities of text-to-SQL systems, as illustrated in Figure 1. In a nutshell, the proposed framework takes as input an existing text-to-SQL dataset, augments the data with plausible roles and access policies generated with an LLM with human-in-the-loop validation, and defines new evaluation metrics that evaluate both SQL correctness and RBAC compliance. We apply this framework to three widely-used benchmarks, namely Spider [24, 51], BIRD [28], and LiveSQLBench [3], producing RBAC-augmented datasets that support the evaluation of both access-policy compliance and SQL utility, and conduct a systematic empirical study that reveals previously unexamined behaviors of text-to-SQL systems under access control.

Our goal is not to replace deterministic DBMS-side enforcement with an LLM, but to evaluate the upstream text-to-SQL problem. We study whether a system can make the correct allow/deny decision given the natural language query and role policy and, when allowed, generate correct, policy-compliant SQL. A DBMS-side checker remains complementary, but does not resolve this evaluation problem. It can reject unauthorized SQL, yet does not indicate whether the system should have refused earlier, generated a policy-compliant alternative, or how much utility is lost to over-refusal.

Challenges. Designing an RBAC-aware text-to-SQL benchmark faces four challenges. First, synthesized roles and fine-grained policies must be semantically meaningful: they should reflect plausible application contexts while avoiding trivial or redundant permission patterns. Second, reliable ground truth requires precise permission verification, as each reference SQL must be analyzed to identify all required permissions over tables, columns, and operations. Third, automatically generated roles and policies may be unrealistic or imbalanced, requiring human validation of semantic plausibility, permission coverage, and role distinctness. Finally, existing text-to-SQL metrics do not

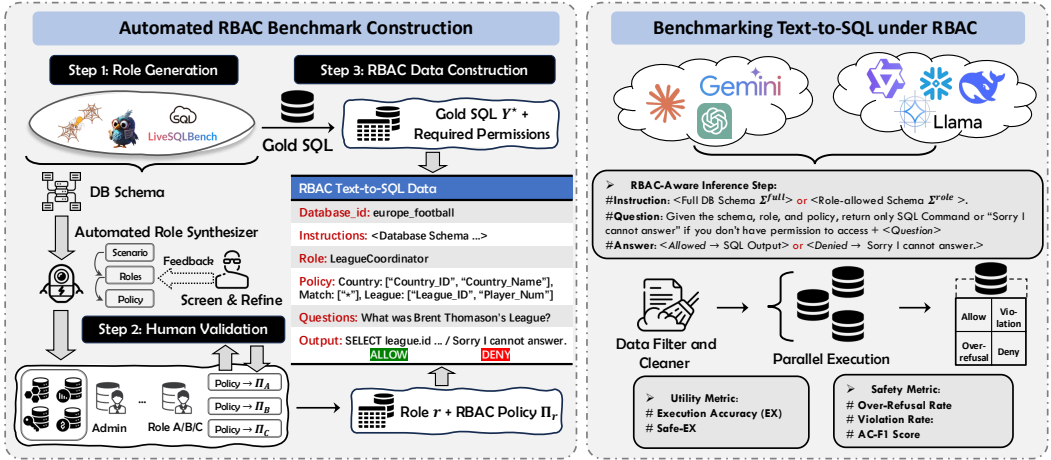


Fig. 1. Overview of the proposed RBAC-aware dataset construction and benchmarking pipeline.

capture RBAC-specific failures, motivating new metrics that jointly measure SQL utility and policy compliance.

Framework overview. At the core of the proposed framework is an automated and auditable pipeline that augments existing text-to-SQL benchmarks with realistic RBAC settings while preserving their original linguistic diversity and query semantics. For each database, the pipeline uses LLMs to synthesize plausible user roles and fine-grained access policies. To avoid degenerate role definitions, role synthesis follows a structured reasoning process tailored to text-to-SQL under RBAC constraints. Specifically, the model first infers an application context from the database schema, and then derives role responsibilities and access policies consistent with the schema structure. The resulting policies operate at the column-operation level, enabling evaluation under restricted access.

We further incorporate *human-in-the-loop* validation to ensure that the synthesized roles and policies are meaningful and faithful to practical settings. LLM-generated role configurations are first screened using interpretable quality metrics, including denial rate, coverage balance across operations, role distinctness, and semantic alignment with the database schema, and are then reviewed by human evaluators for plausibility. Configurations that fail these checks are rejected, with failure signals fed back to the synthesis stage to refine the role definitions and access policies. For permission assignment, each reference SQL query is parsed using SQLGlot [33] to extract the required permissions, which are then validated against each role's access policy. This process produces RBAC-aware ground truth that distinguishes between allowed queries, proper refusals, and unsafe cases in which a semantically correct query would violate access constraints.

Finally, we introduce evaluation metrics that explicitly disentangle SQL utility from policy compliance. These metrics capture not only whether a generated query produces the correct result under full access, but also whether it respects user permissions and avoids RBAC-rejected successes, namely cases in which syntactically correct SQL violates access policies.

Benchmarking results and analysis. We apply this framework to several widely used text-to-SQL benchmarks, resulting in large-scale evaluation resources spanning 53 databases, 399 tables, and 3,353 columns, with a total of 21,502 RBAC-annotated query instances. Using these benchmarks, we conduct a systematic empirical study of state-of-the-art text-to-SQL systems and observe persistent

RBAC-related failures. Failure analysis reveals the *refusal-cliff* phenomenon, i.e., an LLM may correctly follow RBAC rules during reasoning, yet still output a final SQL query that violates access control policies. Meanwhile, common mitigation measures, including *supervised fine-tuning* and *in-context learning*, show limited effectiveness in reducing RBAC failures. These findings highlight a substantial gap between strong benchmark performance under unrestricted settings and the requirements of access-controlled deployments, motivating text-to-SQL systems that explicitly reason about RBAC constraints.

We release the complete benchmarking pipeline, evaluation toolkit, and the resulting RBAC-augmented datasets in a public repository [12], to support reproducibility and future research.

2 Problem Setting

This section formalizes the RBAC-compliant text-to-SQL task studied in this paper, including its objectives and schema exposure mechanisms. We then describe the LLM-based role (profile) synthesis problem used to construct our benchmark.

2.1 Text-to-SQL under RBAC

LLM-based text-to-SQL. Let $S = (\mathcal{T}, C, \mathcal{R})$ denote a relational database schema, where $\mathcal{T}$ is the set of tables, C the set of columns, and $\mathcal{R}$ the set of foreign-key relations. Given a natural-language question Q and a schema text Σ describing S , the goal of text-to-SQL is to generate a SQL program Y whose execution result matches that of a canonical gold standard query Y^* .

An LLM-based system forms a prompt $\mathcal{P}(Q, \Sigma)$ and decodes an SQL program token by token using a language model M :

$$P_M(Y \mid \mathcal{P}(Q, \Sigma)) = \prod_{i=1}^{|Y|} P_M(y_i \mid \mathcal{P}(Q, \Sigma), y_{<i}).$$

Through the model M , a candidate program $\hat{Y}$ is obtained by greedy or sampling-based decoding, and then executed for evaluation.

Text-to-SQL with RBAC. We extend the standard text-to-SQL setting by introducing role-based access control. Each role r is associated with an access policy $\Pi_r \subseteq \mathcal{T} \times C \times \mathcal{O}$, where $\mathcal{O}$ denotes the set of SQL operations (e.g., SELECT, INSERT, UPDATE, DELETE). A permission $(t, c, o) \in \Pi_r$ authorizes role r to apply operation o to column c of table t . Given a gold SQL Y^* , we extract the required permission set $\text{Perm}(Y^*) \subseteq \mathcal{T} \times C \times \mathcal{O}$, corresponding to all (table, column, operation) triples exercised by Y^* . The ground-truth access decision for an instance (Q, S, r) is defined as:

$$y = \begin{cases} \text{allow,} & \text{if } \text{Perm}(Y^*) \subseteq \Pi_r, \\ \text{deny,} & \text{otherwise.} \end{cases}$$

This construction follows the standard assumption adopted in text-to-SQL benchmarks that the gold SQL, authored by domain experts, accurately captures the data access and operations required to answer the query. Accordingly, in our benchmark, we treat the gold SQL as a correct and complete operational specification of the user's intended data access behavior.

A system receives (Q, Σ, r, Π_r) , where Π_r is the serialized role policy provided in the prompt, and returns either a refusal symbol $\perp$ or a SQL program $\hat{Y}$. Emitting $\perp$ induces $\hat{y} = \text{deny}$; emitting $\hat{Y}$ induces $\hat{y} = \text{allow}$, after which $\hat{Y}$ is evaluated along two dimensions: (i) *RBAC compliance*, by checking whether $\text{Perm}(\hat{Y}) \subseteq \Pi_r$, and (ii) *execution correctness*, by comparing its execution result with that of the gold SQL Y^* . Notably, even when Y^* is authorized under the policy Π_r , the generated SQL $\hat{Y}$ may still violate RBAC by invoking unnecessary or unauthorized tables, columns, or operations.

Table 1. Databases, tables, columns, and query counts per data source with and without role design.

Data Source	DBs	Tables	Columns	w/o role	w/ role
Spider	20	80	439	1,034	6,926
BIRD	11	75	798	1,534	10,175
LiveSQLBench	22	244	2,116	592	4,401
Total	53	399	3,353	3,160	21,502

We consider two modes for the schema available: (i) Σ^{full} , which describes all information in $\mathcal{S}$; (ii) $\Sigma^{\text{role}}(r)$, which describes only schema elements permitted by Π_r . This controls the information exposed at prompt time without changing the underlying database.

Task objective. For each instance, the model must produce the correct access decision y under the role's RBAC policy. Conditional on both the ground-truth and predicted decision being allow, the model must further generate a SQL program $\hat{Y}$ that is execution-correct with respect to Y^* and compliant with the role's fine-grained permissions. This formulation naturally induces a two-stage task structure, separating policy compliance from SQL generation quality while remaining compatible with execution-based evaluation.

Scope and limitations. Our benchmark focuses on *explicit* RBAC compliance: whether the system output references only the tables, columns, and operations authorized by the role policy Π_r . It does not model *inference-based* leakage, such as inferring salary ranges from authorized columns `hourly_wage` and `hours_worked` when `salary` is denied. Such indirect leakage requires policy models beyond standard RBAC, such as inference control or semantic privacy, and is left as future work in Section 8.

2.2 RBAC Role Synthesis

To construct a realistic benchmark for RBAC-compliant text-to-SQL, an important task is to systematically synthesize a set of plausible role profiles for a given database schema $\mathcal{S}$. We refer to this process as *role synthesis*. Given $\mathcal{S}$, the objective is to generate a role set $\mathbf{R}_\mathcal{S} = \{r_1, r_2, \dots, r_k\}$, where each role r_i is defined as a tuple (n_i, d_i, Π_{r_i}) consisting of a role name n_i , a natural-language description d_i , and a fine-grained RBAC policy Π_{r_i} .

An ideal role set should satisfy the following key properties. First, the synthesized roles must be semantically plausible and reflect real-world job functions and responsibilities that are logically grounded in the application domain of the database $\mathcal{S}$. For example, in an enterprise data analytics setting such as in Example 1, roles such as *Sales Analyst* and *HR Admin* are coherent, whereas *Student* and *Teacher* are not. Second, the roles should represent a reasonable division of responsibilities. The objective is not to create a strict partition (e.g., creating a separate role for every single table), as real-world roles often have overlapping permissions, but to avoid trivial or redundant role definitions. Ideally, generated roles should be semantically distinct, and reflect a realistic organizational structure.

Formally, automated role synthesis seeks to construct a role set $\mathbf{R}_\mathcal{S}$ that balances semantic plausibility and role distinctness. In our framework, this synthesis process is implemented using an LLM as a generative backend, and is realized through a structured pipeline that combines constrained generation with human-in-the-loop screening and iterative refinement, ensuring that the resulting role set $\mathbf{R}_\mathcal{S}$ satisfies the above criteria.

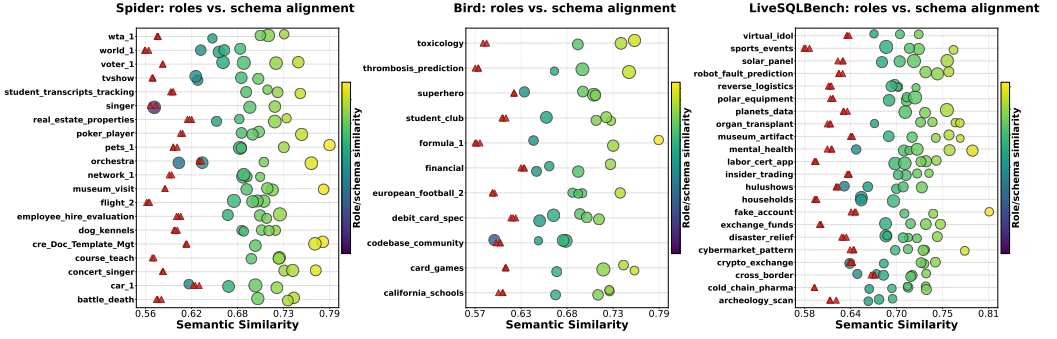


Fig. 2. **Role Coverage and Semantic Alignment.** Each row (e.g., wta_1) corresponds to a database. Each circular marker ● represents a synthesized role, with size indicating the proportion of accessible columns and horizontal axis representing semantic similarity to the schema (purple: low, yellow: high). Red triangles ▲ denote *DataOperator* roles.

3 RBAC-Aware Benchmark Construction

We design a three-step, automated pipeline for constructing RBAC-aware text-to-SQL datasets, illustrated in Figure 1. The pipeline augments existing text-to-SQL benchmarks with role information and access-control semantics by leveraging state-of-the-art LLMs for role synthesis and structured analysis for permission verification. In what follows, Section 3.1 describes the source text-to-SQL datasets, Section 3.2 presents our role generation framework, and Section 3.3 details the RBAC dataset construction process.

3.1 Data Sources and Pre-Processing

We ground our study on three well-established text-to-SQL corpora: (i) *Spider*, a benchmark for complex cross-domain semantic parsing, equipped with a public evaluation framework and curated SQLite resources [51]; (ii) *BIRD*, which incorporates large-scale, real-world databases from diverse professional domains, thereby strengthening realism in data noise, external knowledge dependency, and query complexity [28]; and (iii) *LiveSQLBench-Base-Full-v1*, which simulates end-to-end workloads from enterprise practices, supplemented with auditable executing and testing scripts [3]. For simplicity, hereafter we use the original names *Spider*, *BIRD*, and *LiveSQLBench* to denote the filtered source datasets.

To achieve a unified RBAC-aware evaluation while preserving the original design intents of the source datasets, we apply minimal and auditable preprocessing. Specifically, we adopt the official development splits of *Spider* and *BIRD* to support reproducible experiments without relying on closed-source test servers. For *LiveSQLBench*, we keep analytical queries together with administrative management operations to broaden the task surface across all queries. We also utilize the official database backend from each data source, SQLite execution for *Spider* and *BIRD*, and PostgreSQL for *LiveSQLBench*. We further develop a general-purpose evaluator that supports execution with values based on these backends. The resulting RBAC-aware evaluation pipeline preserves the original question statements, gold SQL answers, data content, and official database backends. Table 1 reports the affected counts alongside corpus statistics.

3.2 Automated Role and Policy Synthesis

As described in Section 2.2, given a database S , we synthesize a set of user roles R_S using an automated role synthesis pipeline with an LLM and human-in-the-loop validation. The LLM is

provided with structured schema information, including table names, column attributes, data types, primary keys, and foreign key relationships, which together define the logical organization of the database.

A naive instantiation of this idea is to present the schema as plain text and directly prompt the model to generate roles. In practice, however, this approach often produces incoherent or overly fragmented role sets that fail to capture implicit domain relationships and realistic access patterns. For example, consider a school database containing tables *StudentGrade*, *TeacherSalary*, and *CampusBuilding*. When prompted directly, the model may generate isolated roles such as *Student* with access only to *StudentGrade*, *Teacher* with access only to *TeacherSalary*, and *Architect* with access only to *CampusBuilding*. While such roles may satisfy the basic schema coverage constraint in Section 2.2, they fail to reflect realistic role responsibilities. In practice, teachers require access to student grades for assessment, and both students and teachers typically share access to campus facilities.

Context-aware role and policy construction. To address these limitations, we adopt a structured role synthesis strategy that constrains the model’s inference process using schema semantics, rather than treating role generation as a free-form text completion task. Specifically, we decompose role synthesis and policy construction into a sequence of semantically grounded reasoning steps that guide the model to reason about the database at multiple levels of abstraction.

As illustrated in Step 1 of Figure 1, the model is first guided to infer a coherent application context implied by the database schema. This step requires the model to reason over table semantics, attribute names, and foreign key relationships to identify high-level usage scenarios, such as grading, enrollment management, or administrative workflows in an academic database. The inferred context serves as an intermediate semantic representation that anchors subsequent role construction. Conditioned on the inferred application context, the model then derives a small set of user roles together with their intended responsibilities. Each role is required to correspond to a meaningful organizational function, rather than an arbitrary subset of tables and columns. The synthesis process is explicitly structured to allow realistic overlap and dependency among roles, reflecting common access patterns in real DBMSs.

After synthesizing the role set, the model proceeds to derive a corresponding access control policy through a context-aware reasoning stage. Rather than assigning permissions independently, the model derives fine-grained access scopes that are consistent with the role’s responsibilities, including column-level and operation-level permissions, depending on the target benchmark. By separating context inference, role responsibility definition, and permission derivation, this structured reasoning process reduces degenerate role definitions and encourages semantically coherent and internally consistent RBAC policies. Finally, we synthesize scoped administrator roles to better reflect practical DBMS deployments. Each database is assigned 2 or 3 *DataOperator* roles with overlapping but incomplete access scopes. These roles provide broad schema coverage while still creating non-trivial deny cases for evaluation. Details are deferred to extended version [11].

Automated screening and human validation. To ensure that the synthesized roles and their corresponding access policies are meaningful and faithful to practical settings, we incorporate a *human-in-the-loop* validation stage into the role synthesis pipeline. The validation process follows a reject-regenerate paradigm that combines automatic quality screening with targeted human inspection, while maintaining explicit records of validation signals and regeneration decisions for auditability purposes.

We first perform an automatic quality check on each synthesized role configuration using a set of interpretable metrics, including denial rate, coverage balance across query operations, role distinctness measured via permission overlap, and semantic alignment between role descriptions

and the database schema; details of the metrics definitions are deferred to the extended version of this paper [11]. These checks are fully automated and deterministic, producing explicit pass or fail signals together with diagnostic metrics that characterize the quality of the generated roles. When a role configuration fails any check, the specific failure reasons and associated metric diagnostics are recorded and packaged as structured feedback to the role synthesis stage. The context-grounded role and policy generation process is then re-executed using this feedback, enabling iterative refinement in a transparent and reproducible manner. The automatic reject-regenerate loop iterates until all quality checks are satisfied.

Once a role configuration passes all automatic checks, it is passed to human validation. Four annotators with database and access-control expertise first complete a lightweight calibration on held-out cases to align the review criteria. They then review each synthesized role configuration, including its roles and policies, for semantic plausibility and consistency with the inferred application context. Each configuration receives a binary accept/reject judgment and is accepted only if at least three of the four annotators approve it; otherwise, it is rejected and regenerated using the collected rejection reasons as structured feedback. Only role configurations that pass both automatic screening and human validation are retained for subsequent permission verification and dataset construction. Note that human validation is performed at the *database-level* role configuration level (53 in total), not at the expanded role-query instance level. Once a configuration is accepted, allow/deny labels for all role-query pairs are produced automatically via structured SQL analysis and policy matching. Detailed validation statistics, including first-pass acceptance, regeneration, manual revision, and total annotation time, are reported in extended version [11].

Validation via semantic alignment. We further assess whether synthesized roles are semantically aligned with the underlying database schema. For each role, we concatenate its name and description and embed the resulting text using a sentence embedding model (OpenAI/text-embedding-3-small in our experiments). We similarly embed a schema description formed from the database's table and column names. We compute cosine similarity between the role and schema embeddings as a lightweight proxy for semantic alignment. For example, *FinancialAuditor* should be closer to a database with transactions and accounts tables than to one with players and matches. This check helps verify that LLM-generated roles are coherent with their assigned databases, rather than arbitrary.

Figure 2 reports the results on the three source datasets. Across all datasets, most synthesized roles show high semantic alignment with the schema while covering only a subset of the database access space. In contrast, *DataOperator* roles show lower semantic similarity, consistent with their design as scoped administrators. This pattern indicates that ordinary synthesized roles capture role-specific database semantics, while administrators mainly serve as structural safeguards rather than semantically specialized roles.

Human experts inspect the visualized results, identify outlier roles with low semantic similarity to the application context (e.g., the leftmost role in the *singer* database of Spider), and provide feedback to the LLM for regeneration.

3.3 RBAC-Aware Instance Construction

As shown in Figure 1, after obtaining a validated role set R_S and the corresponding access policies for each database, the dataset construction phase expands each source text-to-SQL instance into role-conditioned evaluation instances. In particular, for each example (S, Q, Y^*) in a source dataset, where Q is the natural language question and Y^* is the gold SQL, the framework pairs it with every synthesized role $r \in R_S$ associated with the same database. Each resulting instance inherits the

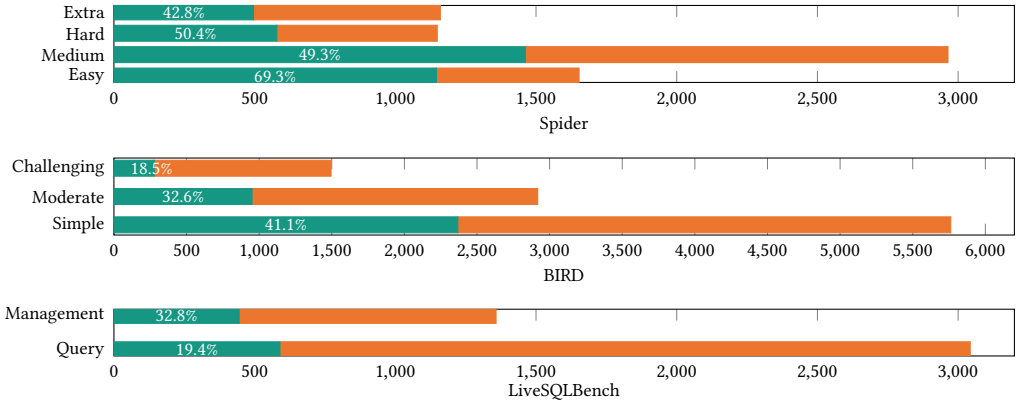


Fig. 3. **Query distribution and difficulty.** Stacked bars show allowed (green) vs. denied (orange) query execution rates for each dataset and difficulty level/task category.

original database identifier, question, and gold SQL, and is augmented with role-specific information, including the role description and the corresponding access policy.

A critical component of this phase is the permission verification mechanism, which deterministically labels each role-query pair as allowed or denied. We perform structured SQL analysis to extract the permissions required by the gold SQL Y^* . Specifically, we parse Y^* using SQLGlot [33] to deterministically recover the set of referenced base tables and accessed columns, and map Y^* to its CRUD operation type. To obtain a complete and accurate dependency set, we analyze clause constructs like JOIN, GROUP BY, ORDER BY, and predicate filters, which may introduce additional column references in a complex SQL. We then validate these extracted requirements against the role's access policy to decide whether the query should be allowed. The extracted requirement set is then matched against the role's fine-grained access policy, which may specify permissions at the column level as per operation type. This produces a clear ground-truth access decision for each role-query pair.

Based on the access decision, the expected output field is constructed in a role-aware manner. If the reference query is authorized under the role's policy, the instance is labeled as allowed and retains the original gold SQL as the expected output. Otherwise, the instance is labeled as denied, and the expected output is replaced with a standardized denial response.

The resulting RBAC dataset retains a consistent schema across benchmark adaptations, including (i) database identifier, (ii) schema instructions, (iii) role designation, (iv) serialized access policy (authorized tables, columns, and operations), (v) the user's natural language question, (vi) expected output (gold SQL or denial response), and (vii) query difficulty. The distribution of difficulty and denial rate is illustrated in Figure 3. In addition, we retain auxiliary metadata, such as the extracted permission requirements from Y^* and the matched policy outcomes, to support downstream analysis and verification. With this structured synthesis framework, the system is able to transform traditional text-to-SQL resources into RBAC-aware datasets while maintaining their semantic integrity.

Role and policy distribution. Table 2 summarizes the resulting role and policy distribution across the three benchmarks. Here the number *# Roles* reports the number of globally distinct role names, while *Avg per DB* measures the number of roles configured in each individual database including DataOperators.

Table 2. Role and policy distribution.

Statistic	Spider	BIRD	LiveSQLBench
# Distinct semantic role names	73	45	90
# Configured roles per DB, range	5–8	5–8	6–9
# Configured roles per DB, avg	6.5	6.7	7.4
Entries (DataOperator%)	6,926 (42.1)	10,175 (43.8)	4,401 (40.4)
Allow / Deny (%)	53 / 47	35 / 65	24 / 76
Col. cov. w/o DataOperator [‡]	0/23/46/30	14/38/31/17	67/29/4/0
Avg union cov. w/o DataOperator	97.3%	95.2%	92.4%

[‡]% of role–DB pairs in bins <25/25–50/50–75/≥75 (excl. DataOperator); see Table 4 for ablation.

Access scopes are highly non-uniform: most synthesized roles remain substantially constrained, with only a small fraction of the column space accessible. The union of all non-DataOperator roles covers 97.3% (Spider), 95.2% (BIRD), and 92.4% (LiveSQLBench) of the column space on average, ensuring broad coverage without relying on a single privileged role. The role union including DataOperator covers 100% of schema columns on every Spider and BIRD database and 99.1% on LiveSQLBench databases, any residual uncovered columns treated as universally denied.

4 Metrics and Evaluation Protocol

As mentioned in Section 1, a key challenge in assessing the RBAC capabilities for text-to-SQL systems is the lack of standardized metrics and evaluation protocols that jointly account for access control and SQL correctness. In this section, we present our evaluation framework tailored to the RBAC setting.

4.1 Metric Design

Evaluating text-to-SQL systems in an RBAC context requires moving beyond traditional accuracy metrics toward a multi-faceted framework that measures both policy compliance (safety) and SQL code correctness (utility). In contrast to conventional benchmarks that mainly rely on the EX metric (explained below), we design a comprehensive suite consisting of the specialized *access control* (AC) metrics, the fine-grained *six-category outcome space*, and the holistic *Safe-EX* metric, elaborated shortly.

Limitations of execution accuracy. Existing text-to-SQL benchmarks adopt the *execution accuracy* (EX) metric, which deems an output correct (referred to as *execution-correct*) if its execution result matches that of the reference (gold) SQL [28, 51]. EX correctly handles semantically equivalent but syntactically different SQL queries. For example, for the question “Show all department names,” with gold SQL `SELECT name FROM department`, the output `SELECT T1.name FROM department AS T1` is execution-correct, even though it does not exactly match the gold SQL textually.

However, as mentioned in Section 1, EX is insufficient in an RBAC context, since it does not cover the outcome that a query should be denied (see Section 2.1). In fact, EX can be misleading since it measures a model’s coding capability but not its compliance, leading to *RBAC-rejected successes* in which a system produces syntactically-correct yet unauthorized queries, e.g., the SQL in Example 1.

Six-category outcome space. To disentangle access-control decisions from SQL generation quality, we further evaluate execution correctness for all queries that the system classifies as allowed. Each query falls into exactly one of the following categories:

- (1) *Correct (C).* The query was correctly allowed with execution-correct SQL. This is an ideal outcome.

- (2) Wrong (W). The query was correctly allowed, but the output SQL is execution-incorrect. This is a conventional failure captured by EX.
- (3) Proper refusal (PR). The query was correctly denied. This is the other ideal outcome besides Correct.
- (4) Violation correct (VC). The query was incorrectly generated with execution-correct SQL (RBAC-rejected success). This represents an attempted RBAC violation.
- (5) Violation wrong (VW). The query was incorrectly allowed with execution-incorrect SQL. This indicates both an RBAC violation and a coding failure.
- (6) Over-refusal (OR). The query was denied despite being authorized, reflecting a security misjudgment and utility loss.

This outcome space enables fine-grained diagnosis of failure modes. For example, a high VC rate indicates that a system understands the schema but ignores access constraints, whereas a high VW rate often correlates with schema misunderstanding under restricted exposure (see Section 6.1).

Access control (AC) metrics. Based on this categorization, we define access-control-aware (AC) metrics that explicitly evaluate authorization behavior under RBAC. Let N denote the total number of role-query instances in the evaluation set. We define the following safety metrics:

- Violation rate $\downarrow$ is $(|VC| + |VW|)/N$, the fraction of all instances for which the ground-truth decision is deny but the system generates SQL. It captures unauthorized query attempts.
- Over-refusal rate $\downarrow$ is $|OR|/N$, the fraction of all instances for which the ground-truth decision is allow but the system refuses the query. It captures utility loss from denying legitimate access, but does not by itself create a security concern.
- AC-F1 score $\uparrow$ evaluates the access decision as a binary allow/deny classification task, with allow as the positive class. Correct and Wrong are true positives, because the system correctly decides to allow the query; SQL utility metrics distinguish C from W. Violation Correct and Violation Wrong are false positives, and Over-Refusal is a false negative. AC-F1 is the harmonic mean of the resulting precision and recall, penalizing both excessive violations and excessive refusals.

These metrics are orthogonal to SQL correctness and focus exclusively on whether the system's behavior aligns with RBAC policy.

Overall safety-utility metric. While the above metrics provide fine-grained insights, practitioners often require a single, holistic metric that summarizes the practical utility of a system. To this end, we define *Safe Execution Accuracy (Safe-EX)* as the fraction of ground-truth allowed instances for which the system returns SQL that is both execution-correct and RBAC-compliant:

$$\text{Safe-EX} = \frac{C}{N_+},$$

where N_+ denotes the number of instances whose ground-truth access decision is allow, and C is the number of correctly allowed instances with execution-correct SQL. This metric directly answers an important question for a practitioner: "How often does this system return safe, execution-correct SQL to the user?". It captures the safety versus utility trade-off in a single number, where a high score requires both high SQL accuracy and strict policy adherence. Symmetrically, we define *Safe-Deny* = PR/N^- as the fraction of denied queries that are correctly refused, where N^- denotes the number of deny instances. This metric complements Safe-EX by quantifying how reliably a system blocks unauthorized access. We report Safe-Deny alongside Safe-EX in the SFT study (Table 7).

4.2 Evaluation Protocol

Multi-pass evaluation. Instead of evaluating all queries and roles, we adopt a stratified sampling strategy where each natural language query is paired with exactly one randomly selected role and

its policy per evaluation pass. We prioritize this approach to preserve the task difficulty distribution of the original benchmarks. Since complex databases (often associated with “Hard” or “Extra Hard” queries) necessitate a larger number of synthesized roles to cover their schemas, evaluating all (query, role) pairs would disproportionately over-represent these difficult instances, thereby skewing the overall metrics. To mitigate the variance introduced by random role assignment, we repeat the evaluation process with $k = 5$ distinct random seeds. Meanwhile, we report all averaged metrics with standard deviation, ensuring that our results reflect robust system performance rather than sampling artifacts.

Execution and pre-filtering. To improve efficiency, we apply a lightweight heuristic filter to exclude obvious refusal responses or non-SQL text prior to execution. For the remaining queries, we follow the ephemeral database mechanism to ensure execution validity, particularly for LiveSQL-Bench which involves data modification (CRUD) operations. Each generated SQL is executed in an isolated transaction or temporary instance that is rolled back post-verification. This design isolates state changes, ensuring that the execution result of one query does not interfere with the correctness of subsequent evaluations.

5 Experimental Evaluation

This section reports experimental results for text-to-SQL under RBAC. We run each model in two settings: standard text-to-SQL and RBAC-conditioned generation using our augmented benchmarks. All models share the same zero-shot prompt template and evaluation harness; the RBAC setting only adds role/policy inputs. We report utility and safety metrics as defined in Section 4.1, and Section 6 analyzes failure modes in detail.

5.1 Setup

As shown in Figure 1, to reduce bias from model-specific prompt strategies and example styles, we adopt a consistent task instruction. For selected models, we conduct two sets of experiments: (i) standard text-to-SQL evaluation without access control, and (ii) access control-aware evaluation using our RBAC-augmented benchmarks. All queries are executed using an aligned backend including SQLite and PostgreSQL to ensure consistent and lightweight execution across datasets.

Models. We selected a comprehensive and representative set of models spanning different deployment settings, training paradigms, and model scales, including both general-purpose language models and specialized text-to-SQL models. The evaluated models include:

- Commercial models, including Google Gemini-2.5-Flash [7], Anthropic Claude Sonnet-4.5-20250929 [2], and OpenAI GPT-4o-mini, GPT-5-mini and GPT-5 [34].
- Open-weight general-purpose models, including DeepSeek V3.2 [30] (both Reasoning and Coder variants), Gemma-3 4B and 27B [42], Qwen-2.5 14B-Instruct [43] and Coder-7B-Instruct [20].
- Specialized text-to-SQL models, Llama3-SQLCoder-8B [8] and Snowflake-Arctic-R1-7B [49].

All models are evaluated without supervised fine-tuning by default and are provided with the same zero-shot template, a fixed temperature setting of 0 when supported by the provider, and a unified execution pipeline with dataset-specific backends (SQLite for Spider and BIRD, and PostgreSQL for LiveSQLBench). Model names follow the providers’ public identifiers. All artifacts are available in our public GitHub repository [12].

5.2 Overall Performance

We evaluate each model along two dimensions: (i) utility, measured by EX in the conventional setting and Safe-EX under RBAC, and (ii) safety, measured by AC metrics, including Violation rate, OverRefusal rate, and AC-F1, as defined in Section 4.1. Table 3 reports the overall results.

Table 3. Overall Performance on Spider, BIRD, and LiveSQLBench considering RBAC. **Bold** marks the best and underline the second-best value within each dataset block. Open-weight LLM rows and commercial LLM rows have different color shades.

Dataset	Model	EX $\uparrow$	Safe-EX $\uparrow$	Violation $\downarrow$	OverRefusal $\downarrow$	AC-F1 $\uparrow$	Avg. Cost/Task (USD $\times 10^{-3}$)	
		w/o role	w/ role	w/ role			w/o role	w/ role
Spider	Snowflake-R1-7b	78.14	80.09 $\pm$ 1.55	45.43 $\pm$ 0.92	0.08 $\pm$ 0.07	70.14 $\pm$ 0.83	–	–
	Llama3-SQLCoder-8b	58.70	64.06 $\pm$ 0.95	46.46 $\pm$ 0.92	0.00 $\pm$ 0.00	69.74 $\pm$ 0.78	–	–
	Gemma3-4b	68.76	72.61 $\pm$ 0.73	46.40 $\pm$ 0.96	0.00 $\pm$ 0.00	69.76 $\pm$ 0.80	0.016	0.017
	Gemma3-27b	80.08	80.31 $\pm$ 1.34	31.72 $\pm$ 0.74	0.93 $\pm$ 0.10	76.31 $\pm$ 0.70	0.036	0.037
	Qwen2.5-Coder-7b-Instruct	74.66	75.69 $\pm$ 0.64	46.33 $\pm$ 0.88	0.00 $\pm$ 0.00	69.80 $\pm$ 0.76	0.057	0.059
	Qwen2.5-14b-Instruct	70.21	71.38 $\pm$ 0.73	34.22 $\pm$ 1.30	0.54 $\pm$ 0.20	75.30 $\pm$ 0.84	0.060	0.062
	Deepseek/V3.2-Reasoning	78.82	75.73 $\pm$ 1.19	1.97 $\pm$ 0.25	2.36 $\pm$ 0.33	95.94 $\pm$ 0.37	0.112	0.115
	Deepseek/V3.2-Coder	76.98	73.78 $\pm$ 1.46	31.43 $\pm$ 0.66	0.58 $\pm$ 0.18	76.79 $\pm$ 0.57	0.112	0.115
	Claude-Sonnet-4.5	85.40	80.99 $\pm$ 0.86	4.55 $\pm$ 0.22	3.01 $\pm$ 0.42	93.03 $\pm$ 0.48	1.420	1.448
	Gemini-2.5-Flash	84.24	81.31 $\pm$ 1.51	10.13 $\pm$ 0.29	2.09 $\pm$ 0.25	89.38 $\pm$ 0.31	0.163	0.166
	OpenAI/GPT-4o-mini	74.76	61.37 $\pm$ 0.95	15.61 $\pm$ 1.11	10.37 $\pm$ 0.53	76.86 $\pm$ 1.49	0.068	0.069
	OpenAI/GPT-5-mini	75.15	71.25 $\pm$ 0.63	3.42 $\pm$ 0.47	1.72 $\pm$ 0.32	95.27 $\pm$ 0.29	0.134	0.136
	OpenAI/GPT-5	74.37	67.73 $\pm$ 1.29	2.55 $\pm$ 0.38	1.97 $\pm$ 0.31	<u>95.79 $\pm$ 0.33</u>	0.670	0.682
BIRD	Snowflake-R1-7b	41.88	62.14 $\pm$ 1.05	63.77 $\pm$ 0.84	0.07 $\pm$ 0.07	52.90 $\pm$ 0.96	–	–
	Llama3-SQLCoder-8b	21.92	38.80 $\pm$ 1.47	64.02 $\pm$ 0.87	0.00 $\pm$ 0.00	52.88 $\pm$ 0.95	–	–
	Gemma3-4b	22.18	37.56 $\pm$ 1.25	64.00 $\pm$ 0.89	0.00 $\pm$ 0.00	52.89 $\pm$ 0.96	0.043	0.044
	Gemma3-27b	40.70	50.69 $\pm$ 1.68	32.72 $\pm$ 0.53	1.66 $\pm$ 0.25	66.59 $\pm$ 0.77	0.097	0.098
	Qwen2.5-Coder-7b-Instruct	31.77	45.90 $\pm$ 1.74	63.99 $\pm$ 0.89	0.00 $\pm$ 0.00	52.89 $\pm$ 0.96	0.152	0.154
	Qwen2.5-14b-Instruct	28.70	45.76 $\pm$ 0.86	54.32 $\pm$ 0.91	0.13 $\pm$ 0.07	56.79 $\pm$ 0.96	0.157	0.158
	Deepseek/V3.2-Reasoning	39.79	56.69 $\pm$ 1.25	8.65 $\pm$ 0.50	1.37 $\pm$ 0.16	<u>87.34 $\pm$ 0.61</u>	0.300	0.302
	Deepseek/V3.2-Coder	38.29	55.95 $\pm$ 1.42	48.50 $\pm$ 1.02	0.13 $\pm$ 0.08	59.54 $\pm$ 1.07	0.300	0.302
	Claude-Sonnet-4.5	53.69	65.02 $\pm$ 0.89	7.37 $\pm$ 0.49	1.81 $\pm$ 0.15	88.12 $\pm$ 0.90	3.571	3.599
	Gemini-2.5-Flash	<u>51.66</u>	67.01 $\pm$ 0.89	20.49 $\pm$ 1.07	0.91 $\pm$ 0.14	76.59 $\pm$ 1.24	0.391	0.394
	OpenAI/GPT-4o-mini	36.42	41.05 $\pm$ 1.15	28.55 $\pm$ 0.69	5.05 $\pm$ 0.49	64.77 $\pm$ 0.49	0.173	0.175
	OpenAI/GPT-5-mini	41.36	56.46 $\pm$ 0.94	12.49 $\pm$ 0.59	0.65 $\pm$ 0.14	84.30 $\pm$ 0.48	0.323	0.326
	OpenAI/GPT-5	43.05	59.25 $\pm$ 1.56	10.46 $\pm$ 0.68	0.68 $\pm$ 0.11	86.36 $\pm$ 0.74	1.617	1.629
LiveSQLBench	Snowflake-R1-7b	5.91	18.63 $\pm$ 4.33	76.15 $\pm$ 1.60	0.27 $\pm$ 0.08	37.54 $\pm$ 1.38	–	–
	Gemma3-4b	3.38	10.44 $\pm$ 2.10	67.53 $\pm$ 1.46	2.57 $\pm$ 0.20	37.08 $\pm$ 1.30	0.393	0.394
	Gemma3-27b	8.45	16.59 $\pm$ 1.80	55.27 $\pm$ 1.28	1.93 $\pm$ 0.36	42.70 $\pm$ 0.90	0.884	0.884
	Qwen2.5-Coder-7b-Instruct	5.07	14.73 $\pm$ 1.54	68.11 $\pm$ 1.28	0.84 $\pm$ 0.24	39.37 $\pm$ 1.63	1.377	1.378
	Qwen2.5-14b-Instruct	7.26	15.80 $\pm$ 3.89	55.44 $\pm$ 2.93	1.11 $\pm$ 0.17	43.89 $\pm$ 1.79	1.386	1.388
	Deepseek/V3.2-Reasoning	<u>23.14</u>	24.73 $\pm$ 3.58	25.44 $\pm$ 1.91	6.11 $\pm$ 0.64	52.03 $\pm$ 1.36	2.743	2.746
	Deepseek/V3.2-Coder	18.92	32.16 $\pm$ 1.34	39.73 $\pm$ 2.89	2.80 $\pm$ 0.70	49.00 $\pm$ 2.49	2.743	2.746
	Claude-Sonnet-4.5	21.79	20.31 $\pm$ 2.82	12.40 $\pm$ 2.25	8.58 $\pm$ 0.56	58.30 $\pm$ 3.06	30.136	30.165
	Gemini-2.5-Flash	21.28	22.48 $\pm$ 6.06	31.76 $\pm$ 3.27	3.78 $\pm$ 0.94	52.30 $\pm$ 2.26	3.085	3.088
	OpenAI/GPT-4o-mini	13.01	13.36 $\pm$ 3.48	22.60 $\pm$ 1.31	12.23 $\pm$ 1.55	38.71 $\pm$ 2.86	1.496	1.498
	OpenAI/GPT-5-mini	18.58	30.09 $\pm$ 0.95	17.70 $\pm$ 1.93	4.32 $\pm$ 0.31	<u>63.22 $\pm$ 2.65</u>	2.565	2.567
	OpenAI/GPT-5	27.20	32.17 $\pm$ 2.75	16.72 $\pm$ 2.14	4.53 $\pm$ 0.64	63.80 $\pm$ 2.41	12.823	12.835

Since EX is measured on the original unrestricted benchmark, whereas Safe-EX is measured only on ground-truth allowed instances in the RBAC-augmented benchmark, the two metrics are not directly comparable as raw percentages. Our EX scores closely match publicly reported single-model results on Spider, BIRD, and LiveSQLBench under zero-shot prompt settings.

Access control results. Under RBAC, AC-F1 varies substantially across models and datasets, with violation rates consistently exceeding over-refusal rates. On Spider, the flagship commercial model GPT-5 attains strong AC-F1 results and relatively low (but still non-zero) violation rates. In contrast, several open-weight models show significantly higher violation rates despite their similar or higher EX scores compared to GPT-5. On BIRD and LiveSQLBench, all evaluated models show degraded AC-F1 and elevated violation rates. For instance, Snowflake-R1-7b, a strong model in terms of EX scores, records a 63.77% violation rate on BIRD, while multiple models on LiveSQLBench retain

double-digit violation rate alongside low AC-F1. These results indicate that enforcing RBAC remains challenging even for strong text-to-SQL models.

Utility results. Safe-EX consistently declines from Spider to BIRD and further on LiveSQLBench, reflecting increasing schema complexity and stricter access constraints. Moreover, although EX and Safe-EX use different denominators, with EX measured on the original unrestricted benchmark and Safe-EX on RBAC-allowed augmented instances, the results still show that high unrestricted SQL accuracy does not necessarily translate to robust performance under access control. Several models with competitive EX achieve only modest Safe-EX once RBAC is enforced.

Utility-safety trade-offs. We observe a systematic tradeoff between SQL utility and access-control safety when moving from the base model to their reasoning-oriented variants in Table 3. Using Safe-EX and AC-F1 as utility and safety metrics, respectively, Snowflake-R1-7B consistently improves utility over its base model Qwen2.5-Coder-7B-Instruct on all three benchmarks, yet the former’s safety is no better than the latter’s. This pattern is most pronounced on LiveSQLBench, where Safe-EX increases but AC-F1 declines, indicating that reasoning-oriented post-training prioritizes executable SQL generation under constraints, but weakens refusal alignment at decision time. This confirms that better SQL execution under RBAC does not automatically confer better policy compliance, and motivates joint objectives or decoding controls that couple reasoning with calibrated denial behaviors.

Commercial vs. open-weight LLMs. A clear separation emerges between commercial and open-weight models. Commercial models generally achieve higher AC-F1 scores and lower violation rates than open-weight baselines, although violations remain nontrivial on harder datasets such as BIRD and LiveSQLBench. In contrast, many open-weight models, including text-to-SQL-specialized models with strong EX performance on standard leaderboards, exhibit substantially higher violation rates. This gap may partly reflect differences in pre-training and post-training objectives. Commercial models are typically trained with extensive instruction tuning and safety alignment, which may partially transfer to RBAC compliance. By contrast, open-weight text-to-SQL models are often optimized primarily for SQL accuracy, which may leave them less robust to access-control constraints.

Task complexity and access control. We further analyze the effect of task difficulty on RBAC compliance by grouping queries according to the difficulty annotations of the source benchmarks. Table 5 reports representative results on BIRD. Due to space constraints, full results covering additional models and all three benchmarks are provided in extended version [11]. Across all benchmarks and models, we observe a clear and consistent inverse relationship between task complexity and RBAC compliance: as queries progress from easier to harder categories, safety performance (i.e., AC-F1) consistently degrades. Notably, this degradation is driven primarily by increased RBAC violations rather than over-refusals, indicating that under higher cognitive load, models tend to prioritize SQL generation utility over access control constraints.

Multi-table join complexity. We further stratify violation behavior by the number of JOIN operations in gold SQL. Figure 4 shows the results for DeepSeek-v3.2-Coder. 0-JOIN bin contains queries without any JOIN clause: single-table SELECTs on Spider/BIRD, and additionally CRUD queries on LiveSQLBench. The violation rate increases with the number of JOINS, consistent with the expectation that cross-table reasoning makes policy compliance harder.

Effect of scoped administrators. Table 4 compares model performance on Spider with and without DataOperator roles. Removing DataOperator lowers the violation rate, since it removes many non-trivial deny-case instances, but model rankings remain largely stable. This indicates that

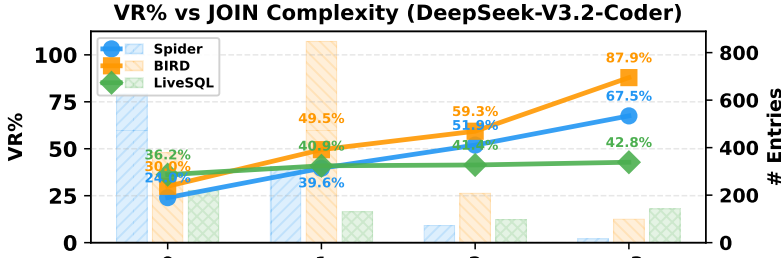


Fig. 4. Violation rate vs. # JOIN operations in the gold SQL.

Table 4. Model performance with/without DataOperator.

Model	Safe-EX↑		Viol-Rate(%)↓		OR-Rate(%)↓		AC-F1↑	
	w/	w/o	w/	w/o	w/	w/o	w/	w/o
DeepSeek-Coder	73.8	72.6	31.4	22.4	0.6	0.8	76.8	82.8
Claude-Sonnet-4.5	81.0	82.9	4.5	4.5	3.0	2.5	93.0	93.9
GPT-4o-mini	61.4	62.0	15.6	10.5	10.4	10.7	76.9	81.2
GPT-5	67.7	69.2	2.6	2.5	2.0	1.8	95.8	96.2

scoped administrators make the benchmark harder by adding genuine access-control decisions, without distorting relative model comparisons.

Granularity and adaptability. Our benchmark currently covers standard *column-level* and *operation-level* RBAC, which are granularities supported by many relational DBMSs. Finer-grained *row/cell-level* policies are not included in the main release due to their substantially higher synthesis and human validation costs, and are left as future work. As a limited feasibility check, we further conducted a row-level study to verify that our benchmarking framework can be extended to row-level predicates; details and results are deferred to extended version [11].

6 Empirical Analysis of RBAC Compliance

This section presents a detailed empirical analysis of text-to-SQL behavior under RBAC constraints. Beyond aggregate performance metrics, the analysis focuses on systematic factors that influence access-control compliance during SQL generation. Specifically, we analyze the impact of schema exposure, characterize representative access-control failures in model reasoning, and evaluate the effectiveness and limitations of two common heuristic remedies: *supervised fine-tuning* and *in-context learning*.

6.1 Impact of Schema Exposure

This subsection examines whether RBAC compliance in text-to-SQL can be enforced by simply removing descriptions of unauthorized schema elements. To this end, we evaluate the impact of schema exposure under two different settings: *Full-Schema*, where the model has access to the entire database schema, and *Role-Schema*, where only schema elements accessible to the role are provided. Table 6 reports the results and summarizes the following violation types:

- **Valid-Guess:** attempting to access valid but unauthorized columns;
- **Invalid-Guess:** attempting to access non-existent columns;
- **Fail-to-Refuse:** generating a SQL query using only authorized columns when the query should instead be refused.

Inspection of the violation cases shows that under the Full-Schema setting, a large number of violations involve accessing known unauthorized columns. For example, DeepSeek-Coder records

Table 5. RBAC-BIRD difficulty tier performance.

Model	Difficulty	Safe-EX↑	Viol. (%)↓	OR (%)↓	AC-F1↑
Snowflake-R1-7b	Simple	69.21±0.99	58.46±1.61	0.05±0.06	58.37±1.65
	Moderate	50.27±1.53	65.60±1.08	0.14±0.18	50.97±1.30
	Challenge	47.01±6.00	80.00±0.74	0.00±0.00	33.13±1.32
Llama3-SQLCoder-8b	Simple	48.42±1.71	58.81±1.69	0.00±0.00	58.27±1.72
	Moderate	21.64±3.71	65.69±1.00	0.00±0.00	51.08±1.12
	Challenge	21.39±3.00	80.17±1.00	0.00±0.00	33.08±1.40
Gemma3-27b	Simple	59.29±1.13	30.15±1.49	1.91±0.45	70.95±1.64
	Moderate	35.96±4.74	36.07±1.64	1.76±0.26	63.25±1.31
	Challenge	33.39±5.59	35.84±1.32	0.52±0.51	51.51±1.43
Qwen2.5-Coder-7b	Simple	55.31±1.32	58.84±1.72	0.00±0.00	58.26±1.73
	Moderate	30.55±3.48	65.60±0.92	0.00±0.00	51.12±1.08
	Challenge	24.52±3.50	80.09±1.06	0.00±0.00	33.11±1.41
Qwen2.5-14b	Simple	53.77±0.79	48.69±1.48	0.14±0.09	62.63±1.67
	Moderate	32.63±3.06	57.52±1.64	0.14±0.11	54.24±1.32
	Challenge	27.57±2.13	69.18±2.44	0.09±0.17	36.31±1.97
Deepseek-V3.2-Coder	Simple	62.72±1.15	42.89±1.35	0.19±0.16	65.49±1.56
	Moderate	47.42±3.86	49.16±1.73	0.09±0.11	58.15±1.29
	Challenge	32.18±5.92	68.14±1.36	0.00±0.00	36.78±1.60
Claude-Sonnet-4.5	Simple	70.42±0.20	7.08±0.50	1.91±0.23	89.70±0.62
	Moderate	57.40±3.81	7.67±1.05	1.81±0.57	87.27±1.37
	Challenge	48.72±7.58	7.88±0.93	1.47±0.70	79.62±3.51
Gemini-2.5-Flash	Simple	73.13±0.68	18.25±1.34	0.95±0.15	80.69±1.26
	Moderate	57.99±2.05	22.48±2.51	0.95±0.36	74.04±2.38
	Challenge	49.84±5.03	25.02±2.20	0.69±0.44	59.80±3.00
GPT-5	Simple	68.60±1.62	9.31±0.90	0.86±0.09	88.79±0.68
	Moderate	44.39±5.50	11.20±1.20	0.41±0.26	85.39±1.51
	Challenge	36.57±7.66	13.33±1.04	0.52±0.33	73.55±2.62

Table 6. Comparison of *Full-Schema* vs *Role-Schema* on BIRD RBAC Data.

Model	Setting	#ValidG	#InvalidG	#FailtoRefuse	Viol (%)	VC	VW	OR (%)	AC-F1
Snowflake-R1-7b	Full-Schema	579.4±9.3	5.6±1.0	977.6±12.9	63.77±0.84	489.4±16.6	488.2±12.5	0.07±0.07	52.90±0.96
	Role-Schema	235.2±11.3	14.8±3.7	846.2±15.7	55.20±1.02	25.4±2.2	820.8±14.1	0.05±0.05	56.49±1.08
	Δ	-344.2±14.6	+9.2±3.8	-131.4±20.3	-8.57±1.32	-464.0±16.7	+332.6±18.8	-0.02±0.09	+3.59±1.44
Gemma-27b	Full-Schema	28.0±5.4	2.0±0.9	501.6±8.1	32.72±0.53	151.2±8.9	350.4±9.6	1.66±0.25	66.59±0.77
	Role-Schema	22.4±1.9	3.2±1.2	450.2±17.7	29.37±1.15	32.6±2.6	417.6±19.1	1.51±0.25	69.03±1.10
	Δ	-5.6±5.7	+1.2±1.5	-51.4±19.5	-3.35±1.27	-118.6±9.3	+67.2±21.4	-0.15±0.35	+2.44±1.34
Qwen2.5-14b-Instruct	Full-Schema	494.6±16.5	8.0±2.7	832.8±13.9	54.32±0.91	296.4±7.4	536.4±12.1	0.13±0.07	56.79±0.96
	Role-Schema	244.2±11.4	36.0±6.6	658.6±14.5	42.96±0.94	30.2±3.2	628.4±12.6	0.20±0.10	62.34±0.93
	Δ	-250.4±20.1	+28.0±7.1	-174.2±20.1	-11.36±1.31	-266.2±8.1	+92.0±17.5	+0.07±0.12	+5.55±1.34
Deepseek-Coder	Full-Schema	481.4±15.9	1.4±1.5	743.6±15.7	48.50±1.02	325.0±18.4	418.6±8.0	0.13±0.08	59.54±1.07
	Role-Schema	241.4±6.5	9.8±3.1	608.0±13.2	39.66±0.86	52.6±6.7	555.4±15.1	0.59±0.07	63.71±0.92
	Δ	-240.0±17.2	+8.4±3.4	-135.6±20.5	-8.84±1.33	-272.4±19.6	+136.8±17.1	+0.46±0.11	+4.17±1.41
GPT-5-mini	Full-Schema	66.2±3.8	0.2±0.4	191.4±9.1	12.49±0.59	42.8±5.3	148.6±4.6	0.65±0.14	84.30±0.48
	Role-Schema	143.6±8.2	0.0±0.0	409.6±5.8	26.72±0.38	30.4±3.6	379.2±5.6	0.33±0.22	72.46±0.65
	Δ	+77.4±9.0	-0.2±0.4	+218.2±10.8	+14.23±0.70	-12.4±6.4	+230.6±7.2	-0.32±0.26	-11.84±0.81

an average of 481.4 such cases. In contrast, under the Role-Schema setting, accesses to known unauthorized columns occur much less frequently, with an average of 241.4 cases, which is likely

Model reasoning: To translate the given natural language question into an executable SQL query, we need to focus on the following points:

1. Identify records from the AuditAndCompliance table that have more than half of their findings as critical ($\text{critFindNum} > \text{FINDTALLY} / 2$).
2. Ensure that the remediation deadline (REMED_DUE) has already passed.
3. Only change the audit status AudtrailState to Overdue if it is not already Complete or Overdue.

...

Here is the SQL query that meets the requirements:

```
SELECT
    AUDIT_TRACE, AudtrailState, FINDTALLY, ...
FROM
    AuditAndCompliance
WHERE
    AudtrailState NOT IN ('Complete', 'Overdue')
    AND critFindNum > FINDTALLY / 2
    AND REMED_DUE < CURRENT_DATE
```

However, the above query only filters the records. To actually change the status to Overdue, we need to perform an update operation.

...

Finally, the correct SQL query is:

Final output:

```
SELECT * FROM AuditAndCompliance
WHERE
    AudtrailState NOT IN ('Complete', 'Overdue')
    AND critFindNum > FINDTALLY / 2
    AND REMED_DUE < CURRENT_DATE
```

Fig. 5. A case study illustrating RBAC failures in reasoning.

due to the absence of schema visibility for those columns. However, rather than refusing the query, models tend to generate SQL using authorized columns and, in some cases, hallucinated ones, resulting in an increase in *violation wrong* cases. A closer examination of the detailed results shows that under the Role-Schema setting, the number of *violation correct* cases (see Section 4.1) drops substantially for several models. For example, DeepSeek-Coder shows a reduction from 325.0 to 52.6 cases, and GPT-5-mini decreases from 42.8 to 30.4 cases. At the same time, both the number of violation wrong cases and the overall violation rate increase for strong models such as GPT-5-mini, with the violation count rising from 148.6 to 379.2, corresponding to an increase of 14.2 percentage points.

In sum, these results indicate that simply restricting schema visibility to role-accessible columns reduces explicit data leakage but does not effectively enforce RBAC policies, as models continue to hallucinate and rarely refuse unauthorized queries.

6.2 Case Study: RBAC Failures in Reasoning

We observe a characteristic failure mode in Snowflake-R1-7b, a state-of-the-art reasoning-first model, where the final output contradicts the model's own intermediate reasoning. Specifically, the model correctly reasons about access constraints and the requested operation, yet emits an SQL query that violates the intended RBAC decision at generation time.

Table 7. RBAC supervised fine-tuning study.

Model		SafeEX(%)↑	SafeDeny(%)↑	Viol.(%)↓	OR(%)↓	AC-F1↑
<i>Evaluation on Spider-dev</i>						
Llama3-SQL	SFT	46.96±0.19	90.89±0.94	4.24±0.46	20.44±0.57	72.83±0.76
Coder-8b	Δ	-17.10±1.07	+90.89±0.94	-42.22±0.88	+20.44±0.57	+3.10±0.85
Qwen2.5-	SFT	51.06±1.52	92.70±0.93	3.38±0.39	19.88±0.95	74.31±0.94
Instruct-14b	Δ	-20.31±2.00	+66.35±3.15	-30.83±1.66	+19.34±1.06	-0.99±1.32
<i>Evaluation on BIRD-dev</i>						
Llama3-SQL	SFT	17.11±1.08	79.15±0.92	13.36±0.62	14.49±0.70	60.61±1.67
Coder-8b	Δ	-21.69±1.49	+79.07±0.89	-50.66±0.89	+14.49±0.70	+7.72±1.64
Qwen2.5-	SFT	31.74±1.33	86.77±1.01	8.48±0.69	12.12±0.51	69.80±1.28
Instruct-14b	Δ	-14.01±1.60	+71.56±0.59	-45.84±0.56	+11.99±0.52	+13.00±1.06
<i>Evaluation on LiveSQLBench</i>						
Qwen2.5-	SFT	4.08±0.40	88.99±1.24	8.45±0.91	15.84±0.88	37.84±3.11
Instruct-14b	Δ	-11.72±4.00	+61.21±1.33	-46.99±1.49	+14.73±0.96	-6.05±2.97

As shown in Figure 5, the model explicitly reasons that the request requires modifying audit records by updating the audit status to Overdue. This is a write operation on AuditAndCompliance table and should therefore be denied under restricted access. However, despite identifying the need for an update operation during reasoning, the model ultimately produces a read-only SELECT query, effectively bypassing the access-control decision.

This reasoning-action inconsistency aligns with the *refusal cliff* phenomenon [50], in which a reasoning model maintains strong refusal intentions during internal reasoning but fails to preserve them in the final output. Across RBAC instances that require denial, the model’s chain-of-thought frequently reaches the correct deny conclusion, yet decoding favors a non-refusal SQL query. This effect is more pronounced for harder queries with longer reasoning chains or more complex access constraints. We hypothesize two potential contributing factors. First, a *decoding prior* toward task completion may outweigh refusal tokens during generation. Second, *post-training objectives* may emphasize usefulness or task success over strict policy adherence. These factors suggest that enforcing RBAC compliance requires tighter coupling between reasoning, decoding, and policy-aware generation mechanisms.

6.3 Limitations of Heuristic Remedies

Our analysis reveals significant safety gaps in current text-to-SQL systems, especially for open-weight models. A natural question is whether these gaps can be mitigated using standard heuristic remedies. To this end, we evaluate two commonly adopted approaches, *supervised fine-tuning* (SFT) and *in-context learning*.

Supervised fine-tuning. We construct a role-aware training dataset from the Spider training split using our role synthesis pipeline. Each instance is formatted as an instruction task containing the database schema, user query, and assigned role, with the target response being either the gold SQL for allow cases or a standardized refusal for deny cases. We evaluate two representative open-weight models, Llama-3-SQLCoder-8B and Qwen2.5-14B-Instruct, and defer implementation details to extended version [11].

Table 7 and Figure 6(a) compare fine-tuned models with their zero-shot baselines. On the in-domain RBAC-Spider benchmark, SFT substantially improves safety compliance. For example,

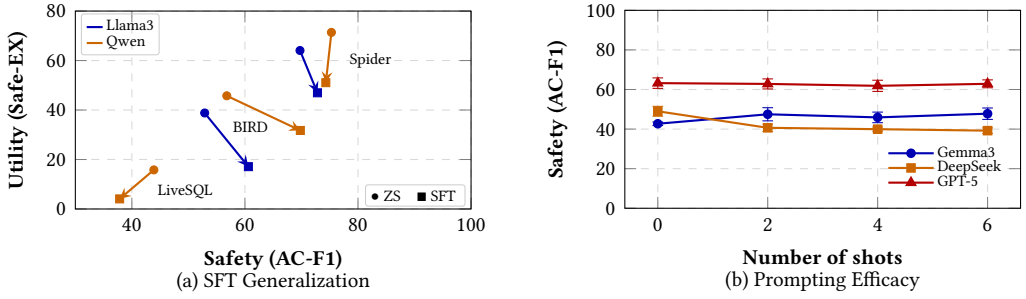


Fig. 6. Analysis of heuristic remedies.

Llama3-SQLCoder-8B improves its AC-F1 from 69.7 to 72.8, with Safe-Deny rising from near zero to over 90% and violation rate dropping from 46% to 4%. This shows that SFT can enforce RBAC in familiar domains. However, this safety gain comes at a clear utility cost, as Safe-EX drops on both models due to elevated over-refusal.

We next evaluate whether these improvements generalize to the more complex RBAC-BIRD and RBAC-LiveSQLBench benchmarks. The limitations of fine-tuning become apparent. Although AC-F1 improves on BIRD (+7.7 for Llama3-SQLCoder and +13.0 for Qwen2.5-14B), it drops on LiveSQLBench (−6.1 for Qwen2.5-14B), and the fine-tuned models remain well below top commercial models such as GPT-5 across all out-of-domain settings. Moreover, Safe-EX drops sharply and Over-Refusal rate increases, while Safe-Deny remains high (>70%). This indicates that the fine-tuned models become biased toward refusal rather than learning generalized RBAC reasoning. Instead of acquiring a generalized understanding of access control, the LLM adopts an ineffective, risk-averse denial strategy. Thus, text-to-SQL under RBAC constraints remains a complex task, not merely a pattern-matching problem solvable by fine-tuning on a single dataset.

In-context learning. We further examine whether few-shot prompting can improve RBAC compliance without updating model parameters. We conduct an ablation study on the challenging RBAC-LiveSQLBench using prompts with $k \in \{2, 4, 6\}$ demonstrations, in addition to the zero-shot baseline. To avoid biasing decisions toward refusal or acceptance, we curate demonstrations in *balanced pairs*, where each additional two shots adds one authorized and one unauthorized query. All experiments follow the stratified multi-pass evaluation protocol in Section 4.2 for fair comparison.

Figure 6(b) shows that few-shot prompting does not provide consistent safety improvements. Contrary to the expectation that more context improves compliance, we observe *non-monotonic fluctuations* across models. While balanced demonstrations help Gemma3-27B, improving AC-F1 from 43% to 48%, they cause a safety regression in DeepSeek-Coder, where AC-F1 drops by nearly 9 points after receiving examples. This suggests that, for some models, few-shot examples introduce noise or distract from intrinsic refusal boundaries rather than clarifying RBAC logic. For stronger models such as GPT-5, additional demonstrations yield negligible gains, indicating a performance ceiling. Consequently, few-shot prompting is highly model-dependent and unstable, making it insufficient for security-critical database interfaces.

6.4 Implications for Practical Deployment

Our experiments reveal a fundamental challenge for practitioners deploying text-to-SQL systems in access-controlled DBMS environments: such systems must simultaneously achieve high SQL generation accuracy and reliable RBAC compliance. The SFT results indicate that fine-tuning open-weight models can substantially improve compliance in a controlled setting. However, this

approach is relatively brittle and does not readily extend to more complex application scenarios. As shown in the previous subsection, safety alignment learned on Spider does not generalize to BIRD or LiveSQLBench, implying that fine-tuning may be required separately for each database. In practice, this may further require maintaining distinct LoRA adapters for different user roles, similar to security-domain isolated deployments [21], which is difficult to scale and costly to maintain in real systems.

Leading commercial models, in contrast, show stronger zero-shot adherence to RBAC constraints, reflecting more generalizable instruction-following capabilities. These models provide a higher baseline level of access-control compliance, likely due to extensive safety alignment during post-training. Table 3 reports the mean per-query cost for role-free and role-aware executions, illustrating the operational cost of such models. Per-task costs are estimated from input/output token counts using each provider’s published pricing; open-weight models are priced via DeepInfra’s hosted inference rates where available, and reported as “–” otherwise. This gives database practitioners a practical trade-off between the control and transparency of open-weight fine-tuning and the convenience and stronger default safety of higher-cost commercial models.

Overall, reliable RBAC is not solved by model selection alone. Improving the intrinsic safety of the LLM is necessary, but robust deployment must also combine language reasoning with external, deterministic access-control enforcement. Our benchmark provides a practical framework for measuring this progress and comparing trade-offs among safety, utility, and cost in enterprise-oriented natural language database interfaces.

7 Related Work

Text-to-SQL benchmarks and methods. Text-to-SQL has evolved from a semantic parsing task into a core database application [5, 15, 17, 25, 26, 29, 32, 44, 47, 48, 52], driven by increasingly challenging benchmarks. Spider [51] established cross-domain generalization over complex schemas, while later benchmarks such as BIRD [28] introduced enterprise-scale databases with noisy and domain-specific data. More recently, LiveSQLBench [3] simulates interactive analytical workflows, and EvoSchema [53] studies robustness to schema evolution. Despite advancing text-to-SQL accuracy and efficiency [25], these benchmarks largely assume unrestricted database access and do not consider access control constraints that are ubiquitous in practical DBMS environments.

Accordingly, recent text-to-SQL methods focus on supporting complex SQL generation through modular pipelines [9, 18, 35]. These include in-context reasoning with task decomposition (e.g., DIN-SQL [35]), database-oriented pre-processing such as schema pruning and example selection [16, 36], domain-specific fine-tuning via instruction tuning or multi-task learning [4, 27, 46], and post-generation refinement using execution-based verification [10]. Despite these advances, existing work has almost exclusively targeted semantic correctness under full-access assumptions, leaving access control compliance largely unexamined.

Database access control. Secure access in enterprise databases is traditionally governed by established access control models, including discretionary and mandatory access control [13, 38], with role-based access control (RBAC) being the dominant paradigm in practice. RBAC assigns permissions to roles rather than users, enabling scalable governance, but integrating LLM-based query generation introduces new challenges, as access decisions must interact with probabilistic model outputs.

Recent work has begun to explore how LLM-based systems can be integrated with access control mechanisms. Database-side approaches, such as BridgeScope [45], enforce privileges via tool-based execution but often expose full schemas to preserve SQL accuracy, increasing the risk of schema inference or prompt-based attacks [31, 40]. Parameter-based approaches, including PermLLM [21]

and related adapter-based methods [1, 14, 19, 37], embed access restrictions into model parameters, but typically assume clearer domain boundaries and do not generalize well to dynamic, cross-domain SQL generation. A concurrent work [23] examines role-conditioned refusal on Spider and BIRD using manually specified roles and read-only queries. In contrast, our work introduces an automated framework for synthesizing realistic roles and fine-grained RBAC policies for arbitrary text-to-SQL benchmarks, and evaluates compliance on LiveSQLBench with full CRUD workloads. We further conduct a large-scale empirical study on this more realistic RBAC setting, enabling systematic analysis of failure modes that are not visible under existing benchmarks.

These trade-offs highlight a severe tension between security, performance, and flexibility. Simple mitigations, such as restricting schema exposure based on roles, are insufficient, as we demonstrate empirically in Section 6.1. Other efforts, including OrgAccess [39] and DePLOI [41], address complementary problems, such as natural language reasoning over access control policies or using text-to-SQL to synthesize and audit policies. However, they do not address the core challenge of evaluating whether a text-to-SQL model can generate or abstain according to role policies at inference time, while final access enforcement remains deterministic, which highlights the need for a dedicated benchmark.

8 Conclusion

In this paper, we introduced a benchmarking framework for evaluating text-to-SQL systems under RBAC constraints. The framework augments existing text-to-SQL benchmarks with realistic roles, fine-grained access policies, RBAC-aware ground truth, and compliance-aware metrics. We instantiate it on three widely used datasets, producing RBAC-augmented benchmarks for systematic evaluation. Our empirical study shows that current text-to-SQL systems, despite strong execution accuracy under unrestricted access, often violate RBAC policies when access constraints are enforced. Common mitigation strategies, including schema restriction, prompt-based policy specification, and supervised fine-tuning, do not reliably eliminate these failures.

These results show a clear gap between existing text-to-SQL evaluation and the needs of access-controlled database deployments. They also motivate further study of text-to-SQL under RBAC constraints, which remains an important but underexplored problem for practical database systems. Future work includes richer role hierarchies and more varied access policy patterns, such as instance-level access control, which are not covered by current benchmarks. Our benchmark focuses on *explicit* RBAC compliance, i.e., whether generated SQL references only authorized resources. It does not model inference-based leakage, such as deriving restricted attributes from correlated authorized ones. Capturing such indirect channels requires policy models beyond standard RBAC, such as inference control or semantic privacy, and is left for future work.

Acknowledgments

This research is supported by the National Research Foundation, Singapore, and Infocomm Media Development Authority under its Trust Tech Funding Initiative. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the National Research Foundation, Singapore, and Infocomm Media Development Authority.

This work is also supported by the Ministry of Education, Singapore, under Tier-2 Grant MOE-000761-01, and a Google Gift Grant.

References

- [1] Saeed Almheiri, Yerulan Kongrat, Adrian Santosh, Ruslan Tasmukhanov, Josemaria Loza Vera, Muhammad Dehan Al Kautsar, and Fajri Koto. 2025. Role-Aware Language Models for Secure and Contextualized Access Control in

- Organizations. In *IJCNLP-AAACL*. 490–511.
- [2] Anthropic. 2025. System Card: Claude Opus 4 & Claude Sonnet 4. <https://www-cdn.anthropic.com/4263b940cabb546a0e3283f35b686f4f3b2ff47.pdf>.
- [3] BIRD-SQL Team. 2025. LiveSQLBench-base-full-v1. <https://huggingface.co/datasets/birdsql/livesqlbench-base-full-v1>. Hugging Face Datasets.
- [4] Shuaichen Chang and Eric Fosler-Lussier. 2023. Selective Demonstrations for Cross-domain Text-to-SQL. In *Findings of EMNLP*. 14174–14189.
- [5] Kaiwen Chen, Yueting Chen, Nick Koudas, and Xiaohui Yu. 2025. Reliable Text-to-SQL with Adaptive Abstention. In *SIGMOD*. 1–30.
- [6] Yeounoh Chung, Gaurav T. Kakkar, Yu Gan, Brenton Milne, and Fatma Özcan. 2025. Is Long Context All You Need? Leveraging LLM’s Extended Context for NL2SQL. *PVLDB* 18, 8 (2025), 2735–2747.
- [7] George Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Naveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261* (2025).
- [8] Defog. 2024. llama-3-sqlcoder-8b. <https://huggingface.co/defog/llama-3-sqlcoder-8b/>.
- [9] Ju Fan, Zihui Gu, Songyue Zhang, Yuxin Zhang, Zui Chen, Lei Cao, Guoliang Li, Samuel Madden, Xiaoyong Du, and Nan Tang. 2024. Combining small language models and large language models for zero-shot NL2SQL. *PVLDB* 17, 11 (2024), 2750–2763.
- [10] Yuankai Fan, Zhenying He, Tonghui Ren, Can Huang, Yinan Jing, Kai Zhang, and X Sean Wang. 2024. Metasql: A generate-then-rank framework for natural language to sql translation. In *ICDE*. 1765–1778.
- [11] Yang Fei, Yangfan Jiang, Yin Yang, and Xiaokui Xiao. 2026. Benchmarking Text-to-SQL under Role-Based Access Control. *arXiv preprint arXiv:2607.22115* (2026). <https://arxiv.org/abs/2607.22115>
- [12] Yang Fei, Yangfan Jiang, Yin Yang, and Xiaokui Xiao. 2026. RBAC-Text2SQL Benchmark: Code and Data. GitHub repository. <https://github.com/2020dfff/RBAC-Text2SQL-Benchmark>
- [13] David Ferraiolo, D Richard Kuhn, and Ramaswamy Chandramouli. 2003. *Role-based access control*. Artech house.
- [14] William Fleshman and Benjamin Van Durme. 2024. AdapterSwap: Continuous Training of LLMs with Data Removal and Access-Control Guarantees. In *CAMLLIS*. 210–223.
- [15] Han Fu, Chang Liu, Bin Wu, Feifei Li, Jian Tan, and Jianling Sun. 2023. Catsql: Towards real world natural language to sql applications. *PVLDB* 16, 6 (2023), 1534–1547.
- [16] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *PVLDB* 17, 5 (2024), 1132–1145.
- [17] Zihui Gu, Ju Fan, Nan Tang, Lei Cao, Bowen Jia, Sam Madden, and Xiaoyong Du. 2023. Few-shot text-to-sql translation using structure and content prompt learning. In *SIGMOD*. 1–28.
- [18] Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. 2025. Next-generation database interfaces: A survey of llm-based text-to-sql. *IEEE TKDE* (2025).
- [19] Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. 2024. LoraHub: Efficient Cross-Task Generalization via Dynamic LoRA Composition. In *COLM*.
- [20] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. 2024. Qwen2.5-coder technical report. *arXiv preprint arXiv:2409.12186* (2024).
- [21] Bargav Jayaraman, Virendra Marathe, Hamid Mozaffari, William Shen, and Krishnaram Kenthapadi. 2025. Permissioned LLMs: Enforcing Access Control in Large Language Models. In *NeurIPS*. 81743–81773.
- [22] Hyeonji Kim, Byeong-Hoon So, Wook-Shin Han, and Hongrae Lee. 2020. Natural language to SQL: Where are we today? *PVLDB* 13, 10 (2020), 1737–1750.
- [23] Dörde Klisura, Joseph Khoury, Ashish Kundu, Ram Krishnan, and Anthony Rios. 2026. Role-Conditioned Refusals: Evaluating Access Control Reasoning in Large Language Models. In *Findings of EAACL*. 6018–6034.
- [24] Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. 2025. Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows. In *ICLR*.
- [25] Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024. The Dawn of Natural Language to SQL: Are We Fully Ready? *PVLDB* 17, 11 (2024), 3318–3331.
- [26] Fei Li and Hosagrahar V Jagadish. 2014. NaLIR: an interactive natural language interface for querying relational databases. In *SIGMOD*. 709–712.
- [27] Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024. Codes: Towards building open-source language models for text-to-sql. In *SIGMOD*. 1–28.
- [28] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2023. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. In *NeurIPS*. 42330–42357.

- [29] Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. 2024. Table-gpt: Table fine-tuned gpt for diverse table tasks. In *SIGMOD*. 1–28.
- [30] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437* (2024).
- [31] Xinjian Luo, Ting Yu, and Xiaokui Xiao. 2025. Prompt Inference Attack on Distributed Large Language Model Inference Frameworks. In *CCS*. 1739–1753.
- [32] Kyle Luoma and Arun Kumar. 2025. Snails: Schema naming assessments for improved llm-based sql inference. In *SIGMOD*. 1–26.
- [33] Toby Mao. 2023. SQLGlot. <https://github.com/tobymao/sqlglot>.
- [34] OpenAI. 2025. OpenAI Platform. <https://platform.openai.com/docs/models/>.
- [35] Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. In *NeurIPS*. 36339–36348.
- [36] Tonghui Ren, Yuankai Fan, Zhenying He, Ren Huang, Jiaqi Dai, Can Huang, Yinan Jing, Kai Zhang, Yifan Yang, and X. Sean Wang. 2024. PURPLE: Making a Large Language Model a Better SQL Writer. In *ICDE*. 15–28.
- [37] Soumadeep Saha, Akshay Chaturvedi, Joy Mahapatra, and Utpal Garain. 2025. sudoLLM: On Multi-role Alignment of Language Models. In *Findings of EMNLP*. 366–384.
- [38] Ravi S Sandhu, Edward J Coyne, Hal L Feinstein, and Charles E Youman. 1996. Role-Based Access Control Models. *IEEE Computer* 29, 2 (1996), 38–47.
- [39] Debdeep Sanyal, Umakanta Maharana, Yash Sinha, Hong Ming Tan, Shirish Karande, Mohan Kankanhalli, and Murari Mandal. 2025. OrgAccess: A Benchmark for Role Based Access Control in Organization Scale LLMs. *arXiv preprint arXiv:2505.19165* (2025).
- [40] Richard Shay, Uri Blumenthal, Vijay Gadepally, Ariel Hamlin, John Darby Mitchell, and Robert K Cunningham. 2019. Don't even ask: Database access control through query control. *ACM SIGMOD Record* 47, 3 (2019), 17–22.
- [41] Pranav Subramaniam and Sanjay Krishnan. 2024. DePLOI: Applying NL2SQL to Synthesize and Audit Database Access Control. *arXiv preprint arXiv:2402.07332* (2024).
- [42] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. 2025. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786* (2025).
- [43] Qwen Team. 2024. Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115* (2024).
- [44] Matthias Urban and Carsten Binnig. 2024. CAESURA: Language Models as Multi-Modal Query Planners. In *CIDR*.
- [45] Lianggui Weng, Dandan Liu, Rong Zhu, Bolin Ding, and Jingren Zhou. 2026. BridgeScope: A Universal Toolkit for Bridging Large Language Models and Databases. In *CIDR*.
- [46] Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I Wang, et al. 2022. UnifiedSKG: Unifying and Multi-Tasking Structured Knowledge Grounding with Text-to-Text Language Models. In *EMNLP*. 602–631.
- [47] Xiangjin Xie, Guangwei Xu, Lingyan Zhao, and Ruijie Guo. 2025. Opensearch-sql: Enhancing text-to-sql with dynamic few-shot and consistency alignment. In *SIGMOD*. 1–24.
- [48] Yicun Yang, Zhaoguo Wang, Yu Xia, Zhuoran Wei, Haoran Ding, Ruzica Piskac, Haibo Chen, and Jinyang Li. 2025. Automated Validating and Fixing of Text-to-SQL Translation with Execution Consistency. In *SIGMOD*. 1–28.
- [49] Zhewei Yao, Guoheng Sun, Lukasz Borchmann, Zheyu Shen, Minghang Deng, Bohan Zhai, Hao Zhang, Ang Li, and Yuxiong He. 2025. Arctic-Text2SQL-R1: Simple Rewards, Strong Reasoning in Text-to-SQL. *arXiv preprint arXiv:2505.20315* (2025).
- [50] Qingyu Yin, Chak Tou Leong, Linyi Yang, Wenxuan Huang, Wenjie Li, Xiting Wang, Jaehong Yoon, Jinjin Gu, et al. 2025. Refusal Falls off a Cliff: How Safety Alignment Fails in Reasoning? *arXiv preprint arXiv:2510.06036* (2025).
- [51] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *EMNLP*. 3911–3921.
- [52] Chao Zhang, Yuren Mao, Yijiang Fan, Yu Mi, Yunjun Gao, Lu Chen, Dongfang Lou, and Jinshu Lin. 2024. Finsql: Model-agnostic llms-based text-to-sql framework for financial analysis. In *SIGMOD*. 93–105.
- [53] Tianshu Zhang, Kun Qian, Siddhartha Sahai, Yuan Tian, Shaddy Garg, Huan Sun, and Yunyao Li. 2025. Evoschema: Towards Text-to-SQL Robustness against Schema Evolution. *PVLDB* 18, 10 (2025), 3655–3668.
- [54] Fan Zhou, Siqiao Xue, Danrui Qi, Wenhui Shi, Wang Zhao, Ganglin Wei, Hongyang Zhang, Caigai Jiang, Gangwei Jiang, Zhixuan Chu, et al. 2024. Db-gpt-hub: Towards open benchmarking text-to-sql empowered by large language models. *arXiv preprint arXiv:2406.11434* (2024).

Received 17 January 2026; revised 19 May 2026; accepted 12 June 2026