WITHIN AND BEYOND SQL: SEARCHING FOR EXECUTION EVIDENCE IN TEXT-TO-SQL VERIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Text-to-SQL maps natural language questions to executable SQL queries over databases. Reliable text-to-SQL requires verifying whether generated SQL faithfully satisfies the semantics of the question. Existing verification remains challenging because intermediate discrepancies may be masked by downstream operations, while omitted requirements leave no corresponding computation from which execution evidence can be directly obtained. We introduce EviSQL, a reference-free verification framework that searches for execution evidence both within and beyond the computation expressed by candidate SQL. Within the SQL, EviSQL reconstructs the computation trace of SQL and tests selected operations through local perturbations, tracking whether the resulting differences propagate to the final answer. Beyond the SQL, EviSQL grounds question requirements in the original wording and relates them to the computation trace, using intent-bound queries to investigate possible semantic coverage gaps. Experiments on *ROSE-VEC* and *BIRD-VC* show that EviSQL consistently improves semantic error detection over existing verification methods. The resulting execution evidence further supports fine-grained error diagnosis and targeted SQL correction, improving the reliability of generated queries. Code and experimental resources are available at `https://anonymous.4open.science/r/EviSQL-30D4`.

1 INTRODUCTION

Text-to-SQL translates natural language questions into SQL queries, facilitating database access. It has been increasingly applied to practical scenarios including electronic health records (Lee et al., 2022), domain-specific enterprise databases (Chen et al., 2025), and industrial data analytics (Ghosh et al., 2025). These applications underscore the growing practical importance of text-to-SQL.

Recent text-to-SQL benchmarks increasingly emphasize realistic databases and complex query semantics, as reflected in BIRD (Li et al., 2023) and Spider 2.0 (Lei et al., 2025). Meanwhile, stronger LLM-based systems have reduced many obvious generation failures, leaving increasingly subtle semantic errors in otherwise executable SQL (Pourreza & Rafiei, 2023a; Wang et al., 2025a; Cen et al., 2025). This makes semantic verification increasingly important, as successful execution alone does not establish whether a candidate SQL query faithfully reflects the requirements of the question.

Prior studies have identified annotation issues in text-to-SQL benchmarks that may distort measured performance (Zhong et al., 2023; Pourreza & Rafiei, 2023b; Wretblad et al., 2024; Jin et al., 2026b). Meanwhile, semantic errors in generated SQL have motivated extensive work on diagnosis and correction (Chen et al., 2023b; Wang et al., 2025b; Xu et al., 2025; Li et al., 2025b). Both settings require determining whether a candidate SQL faithfully reflects the semantics of the question, even when a correct reference SQL is unavailable or unreliable. This motivates verification without a reference SQL, where candidate correctness is assessed using the question and target database.

Existing approaches assess the correctness of candidate SQL queries from different perspectives. One line of work evaluates complete SQL queries with respect to the semantics of the question (Chen et al., 2023a; Kim et al., 2025; Pei et al., 2026). SQLens (Gong et al., 2025) instead performs more fine-grained analysis over predefined SQL components. SAR-Agent (Jin et al., 2026a) verifies SQL queries by iteratively constructing targeted verification queries and inspecting their execution results.

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

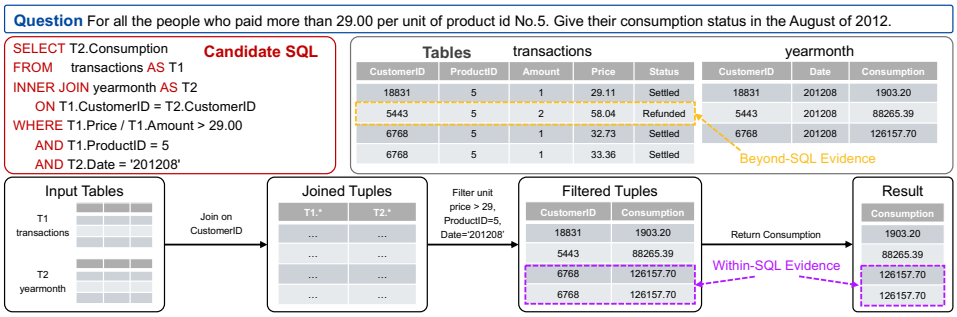


Figure 1: Motivating example of execution evidence within and beyond the candidate SQL.

However, these approaches may not provide sufficient execution evidence to establish semantic correctness. We therefore view reference-free verification as a search for execution evidence.

We distinguish the two scopes according to whether sufficient execution evidence can be obtained within the computation scope induced by the candidate SQL. For *Within-SQL* verification, the evidence can be derived from the candidate computation trace, although downstream operations may obscure the effect of a local discrepancy. For *Beyond-SQL* verification, the requirement has no established realization in that scope and motivates targeted investigation over the database.

***Example* 1.** As illustrated in figure 1, duplicate rows for customer 6768 violate the entity-level answer implied by the question, providing *Within-SQL* evidence. Meanwhile, inspecting the unused `Status` attribute reveals that the transaction for customer 5443 was `Refunded`, violating the rule that only `Settled` transactions count as payments and providing *Beyond-SQL* evidence.

Based on this distinction, we propose **EviSQL**, a reference-free verification framework that searches for execution evidence both within and beyond the candidate SQL. For *Within-SQL* verification, EviSQL reconstructs the candidate computation trace, validates model-proposed local perturbations, and checks whether an accepted change affects the final answer. For *Beyond-SQL* verification, EviSQL anchors question requirements in the original text, maps them to the trace, and investigates possible gaps with Intent-bound exploratory SQL. The resulting evidence supports fine-grained error localization and targeted SQL correction.

We evaluate EviSQL on *ROSE-VEC* (Pei et al., 2026) and *BIRD-VC*, a benchmark constructed from predictions of six models with corrected annotations from (Jin et al., 2026a) as references. These evaluations cover semantic error detection, localization, and diagnosis-guided correction. For SQL issue resolution, we further evaluate EviSQL on *BIRD-CRITIC-SQLite* (Li et al., 2025b). Across all experiments, EviSQL achieves state-of-the-art performance.

To summarize, our contributions are as follows:

- We propose EviSQL, a reference-free verification framework that searches for question-grounded execution evidence both within and beyond the candidate SQL.
- We develop node-scoped perturbation checks with downstream final-answer comparison, together with an Intent coverage audit that links text-grounded requirements to trace nodes and verifies answer-impact witnesses from targeted exploration.
- Experiments on *ROSE-VEC* and *BIRD-VC* show improvements in text-to-SQL verification and diagnosis-guided correction, with a separate task-adapted study of SQL issue resolution.

2 Related Work

Text-to-SQL Verification. FLEX (Kim et al., 2025) and ROSE (Pei et al., 2026) can use reference SQL as part of their verification process. Our experiments evaluate their reference-free configurations. SQLens (Gong et al., 2025) combines database and LLM signals for diagnosis at the SQL-clause level. SAR-Agent (Jin et al., 2026a) incrementally constructs targeted verification queries for auditing SQL annotations through database interaction. Our verifier couples such database evidence to an explicit audit of requirements quoted from the question: each requirement is mapped to the

candidate trace, investigated when its implementation is uncertain, and checked against the actual candidate answer before it can support rejection. A separate line of work constructs distinguishing databases for alternative SQL queries, using multiple candidates in SQLDriller (Yang et al., 2025) and prediction–reference pairs in SpotIt and SpotIt+ (Klopfenstein et al., 2026; Tremante et al., 2026). Our setting instead verifies a given candidate over the target database.

SQL Correction in Text-to-SQL. Recent LLM-based text-to-SQL systems incorporate self-correction mechanisms during inference (Pourreza & Rafiei, 2023a; Wang et al., 2025a). Other approaches improve generation through iterative refinement, candidate selection, or execution-aware reasoning (Talaei et al., 2024; Pourreza et al., 2025; Dai et al., 2026). Dedicated repair frameworks provide more targeted error handling. SQLFixAgent (Cen et al., 2025) coordinates multiple agents for error detection and repair. MapleDoctor (Shen et al., 2026) detects error symptoms through static analysis and applies rule-based or LLM-based repairs. Beyond SQL correction during generation, SQL-ACT (Li et al., 2025b) uses ReAct (Yao et al., 2023) to resolve user-reported SQL issues. However, existing correction methods often identify potential errors from execution feedback, static analysis, or model-generated critiques, but provide limited execution evidence showing why a specific semantic error leads to an incorrect answer.

3 PRELIMINARIES

Problem Formulation. Given a natural language question Q, schema S, database instance D over schema S, and candidate SQL q, we study reference-free semantic verification without a gold SQL query or correctness label. *Error detection* determines whether q satisfies the requirements of Q, while *error diagnosis* identifies and explains violations of these requirements.

Semantic Correctness. For a candidate SQL q, semantic correctness requires its answer $q(D)$ to satisfy Q. We define the semantic error indicator as:

$$\mathbb{I}(Q, S, D, q) = \begin{cases} 1, & \text{if } q \text{ is semantically incorrect for } Q \text{ on } D, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Decision Scope. A query that violates an explicit question requirement remains semantically incorrect even if it happens to return the expected answer on D. A correct verdict is conditional on D, the available evidence, and the adopted interpretation of Q. It does not establish equivalence across all possible database instances, some of which may violate constraints not explicitly expressed in S.

4 METHOD

As illustrated in Figure 2, EviSQL comprises four components: *Computation Trace Construction* builds a semantic trace from the candidate SQL. *In-Trace Verification* examines selected computations through local perturbations, while *Out-of-Trace Exploration* investigates uncovered question requirements using diagnostic SQL probes. *Final-Impact Judgement* checks whether identified discrepancies affect the final answer, enabling early rejection when an error is confirmed or a final decision based on evidence from both reviews.

4.1 COMPUTATION TRACE CONSTRUCTION

EviSQL first creates a database snapshot σ and obtains the execution result $R = q(\sigma)$. It then parses q with schema facts from S into a semantic DAG $G = \mathcal{T}(q, S)$. Following logical dependencies, the DAG is flattened into a *Computation Trace* τ that exposes candidate computations and their input context. Algorithm 1 constructs this trace without executing intermediate nodes in advance.

States and Dependencies. A semantic state denotes a relation produced within the candidate computation. Each producing node is represented as $v = (\mathrm{id}_v, O_v, I_v, o_v, T_v)$, where O_v denotes its operation, $I_v \subseteq V_{\text{state}}$ its input states, $o_v \in V_{\text{state}}$ its output state, and T_v its complete node SQL when a faithful executable context exists. Source relations initialize root states, while joins and set operations may consume multiple input states with distinct roles. We define the semantic DAG as

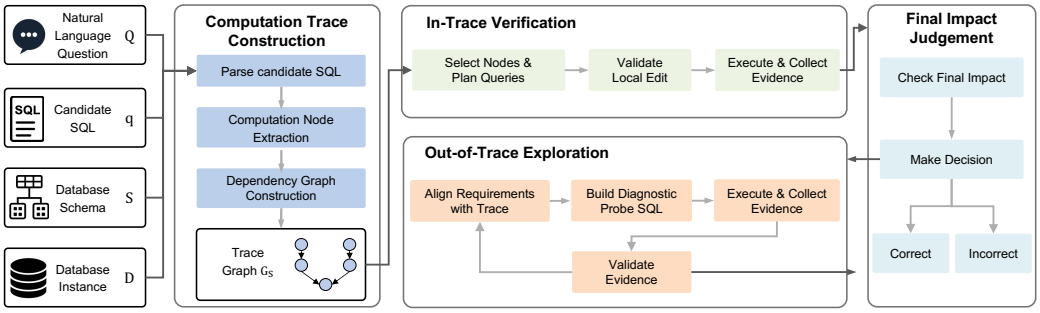


Figure 2: Overview of the EviSQL framework.

$G = (V_{\text{state}} \cup V_{\text{op}}, \mathcal{A})$, where

$$\mathcal{A} = \{(s, v) \mid v \in V_{\text{op}}, s \in I_v\} \cup \{(v, o_v) \mid v \in V_{\text{op}}\},$$
$$o_u \in I_v \Longrightarrow \text{pos}_{\widehat{\tau}}(u) < \text{pos}_{\widehat{\tau}}(v), \qquad u, v \in V_{\text{op}}. \tag{2}$$

Here $\widehat{\tau}$ denotes a topological order over operation nodes. The resulting trace follows this dependency order, retaining executable computations and associating internal operations with downstream executable nodes that contain them. It therefore provides explicit input dependencies and stable node references for subsequent in-trace verification and out-of-trace exploration.

Executable Node SQL. The SQL attached to a node preserves the upstream joins and restrictions required to produce its input. For an executable node, $X_v = T_v(\sigma)$ denotes its logical output, which is materialized only when requested during verification. Thus, only the candidate query q is executed in advance to obtain R. Operations without independent SQL remain represented in the trace and may be associated with downstream executable nodes that contain them. Unsupported or unbound operations provide no independent node-level evidence, particularly for complex correlated or window expressions. The precise support boundaries are described in section A.1.

Algorithm 1 Computation trace construction

Require: Candidate q, schema S, database D
Ensure: Snapshot σ, result R, trace τ
1: $\sigma \leftarrow \text{DatabaseSnapshot}(D)$
2: $R \leftarrow \text{Execute}(q, \sigma)$
3: $G \leftarrow \text{ParseDAG}(q, S)$
4: $\widehat{\tau} \leftarrow \text{TopoOps}(G)$
5: **for** node v in $\widehat{\tau}$ **do**
6: Attach T_v when executable, otherwise retain its containing executable context
7: **end for**
8: $\tau \leftarrow \text{Flatten}(G, \widehat{\tau})$
9: **return** (σ, R, τ)

4.2 In-Trace Verification

In-Trace Verification examines whether computations in q violate question requirements by selecting relevant trace nodes from Q, τ, and R. For each selected node, it requests the original result or proposes a local perturbation that changes only the selected operation. Only selected nodes are materialized, which avoids unnecessary executions. If no node SQL is requested, evidence review is skipped, while the Intent coverage audit remains mandatory. Both reviews use Q, τ, σ, R as fixed context. The Intent inventory $\mathcal{I}$ records requirements anchored in Q, their review status, and deferred concerns. The execution records $\mathcal{E}$ record SQL execution attempts together with their status and observed results. During review, w denotes a proposed witness that links a final answer discrepancy to a requirement in Q and supporting records in $\mathcal{E}$. If no witness is proposed, $w = \text{null}$ and early rejection is not triggered. The outputs $\mathcal{I}, \mathcal{E}$ represent normal completion, while early exits are specified in each algorithm. Algorithm 2 summarizes this process.

Local Validation. Before executing the original and edited node queries, the program verifies that the proposed edit is confined to the selected operation and preserves its input relations and unrelated upstream SQL. Let A_v and A'_v denote the normalized syntax trees of T_v and the proposed replacement T'_v. The selected operator determines the editable region r_v, and M_{r_v} masks this region during structural comparison. The structural check requires

$$M_{r_v}(A_v) = M_{r_v}(A'_v), \qquad A_v \neq A'_v. \tag{3}$$

Algorithm 2 In-trace verification

Require: Q, τ, σ, R
Ensure: Intents $\mathcal{I}$, execution records $\mathcal{E}$
$\mathcal{E}, \mathcal{I} \leftarrow \emptyset$
Select v and optional T'_v
for each selected v **do**
  Validate any T'_v
  Execute admitted requests on σ
  Execute admitted $C_v[T'_v](\sigma)$ if unique
  Append records to $\mathcal{E}$
end for
if $\mathcal{E} \neq \emptyset$ **then**
  Review $\mathcal{E}$, update $\mathcal{I}, w$
  Mark deferred Intents in $\mathcal{I}$
  Validate $\mathcal{I}$ and w
  if $\mathrm{EarlyReject}(w)$ **then**
    return INCORRECT, w
  end if
end if
return $\mathcal{I}, \mathcal{E}$

Algorithm 3 Out-of-trace exploration

Require: $\mathcal{I}, \mathcal{E}$ with fixed context
Ensure: Intents $\mathcal{I}$, execution records $\mathcal{E}$
Retain and extend $\mathcal{I}$ from Q
while true **do**
  Review $\mathcal{I}, \mathcal{E}$, update $\mathcal{I}, w$
  Validate $\mathcal{I}$ and w
  if $\mathrm{EarlyReject}(w)$ **then**
    return INCORRECT, w
  end if
  if Intent audit is complete **then**
    return $\mathcal{I}, \mathcal{E}$
  end if
  Propose SQL for uncovered Intents
  if no admissible SQL remains **then**
    return INCOMPLETE
  end if
  Execute admitted SQL on σ
  Append records to $\mathcal{E}$
end while

For example, an aggregate edit must preserve its input relation, so any unrelated modification to a join or filter violates this constraint. The accepted original and edited queries are then executed on the same database snapshot. This validation establishes the scope of the edit rather than its semantic correctness. If an edit fails validation, the corresponding concern remains unresolved. The model must further explain why the selected operation conflicts with the requirements expressed in Q.

Downstream Impact. Let T_v and T'_v denote the original and edited node computations, and let $C_v[\cdot]$ denote the downstream context used to reconstruct the full query when the rewrite is unique. The corresponding complete answers are $C_v[T_v](\sigma) = R$ and $C_v[T'_v](\sigma) = R'_v$. However,

$$T_v(\sigma) \neq T'_v(\sigma) \;\not\Rightarrow\; C_v[T_v](\sigma) \neq C_v[T'_v](\sigma). \tag{4}$$

Downstream operations may mask a local difference, while ambiguous embedding makes R'_v unavailable. The resulting evidence is passed to *Final-Impact Judgement* for impact assessment. Unresolved concerns remain as deferred Intents in $\mathcal{I}$, with stable identifiers for *Out-of-Trace Exploration*.

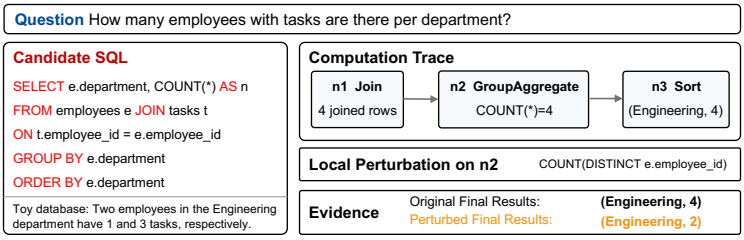


Figure 3: Illustration of the *In-Trace Verification* workflow.

***Example* 2.** For the question asking how many employees with tasks there are per department, figure 3 considers two employees in the engineering department with one and three tasks. The original aggregation `COUNT(*)` counts 4 joined rows, while the perturbation `COUNT(DISTINCT e.employee_id)` at n_2 returns 2 employees. After propagating the local edit through the remaining computation, the final answer changes from $(\text{Engineering}, 4)$ to $(\text{Engineering}, 2)$. Since the question counts employees with tasks rather than joined rows, this difference supports INCORRECT.

4.3 OUT-OF-TRACE EXPLORATION

When verification continues after *In-Trace Verification*, *Out-of-Trace Exploration* retains unresolved concerns with their Intent identifiers and extracts additional requirements from Q. Each Intent

$i = (\mathrm{id}_i, r_i, a_i) \in \mathcal{I}$ records a stable identifier, a requirement description, and a supporting quotation from Q. The model maps each Intent to trace nodes that collectively implement the required operations and relation paths, leaving missing operations or uncertain mappings unresolved. The program checks quotations and identifier consistency without establishing the completeness of $\mathcal{I}$ or the semantic validity of the mappings. Algorithm 3 summarizes this exploration.

Intent Coverage and Gap Investigation. For uncovered Intents, the model explores computations beyond the candidate trace according to the entity roles and scope in Q. Equivalent relation paths may satisfy the same Intent, so different join syntax alone does not indicate an error. Each exploration query is bound to an Intent identifier and gap hypothesis, targets an uncovered Intent, runs read only on snapshot σ, and remains within the exploration budget. Queries may retrieve diagnostic attributes before collecting evidence compatible with the candidate output, without computing the complete answer to Q. Validated evidence updates the investigation and guides subsequent exploration, while empty or matching outputs only exclude tested discrepancies and do not establish semantic coverage. A COVERED Intent cites trace nodes, while a CHECKED Intent has been investigated with valid evidence without implying correct implementation. Unless verification terminates early, every Intent must be COVERED or CHECKED, with deferred concerns required to be CHECKED.

Answer-Impact Witnesses. An *answer-impact witness* is execution evidence grounded in Q that establishes an error in the final answer. *Out-of-Trace Exploration* constructs candidate witnesses identifying required tuples absent from the complete result R or tuples in R that violate an Intent. Evidence must cite observed rows matching the candidate output structure, with diagnostic attributes kept separate. The program checks provenance and tuple membership in R, while the model explains the discrepancy under the corresponding requirement. *Final-Impact Judgement* assesses the evidence for final impact and possible early rejection.

4.4 FINAL-IMPACT JUDGEMENT

Final Impact Check. *Final-Impact Judgement* applies a shared evidence gate to determine whether a proposed witness w supports early rejection. A witness is accepted only when its execution evidence is programmatically validated and its interpretation is grounded in Q:

$$\mathrm{EarlyReject}(w) = \mathrm{ExecValid}_{\mathrm{prog}}(w; R, \mathcal{E}) \wedge \mathrm{Grounded}_{\theta}(w; Q). \tag{5}$$

For *In-Trace Verification*, this requires a valid local edit that changes the complete final answer, while for *Out-of-Trace Exploration*, the exploration result must satisfy output, provenance, and candidate membership checks. The model must explain why the observed discrepancy contradicts Q, preventing unsupported answer differences from serving as decisive evidence.

Decision Making. Without early rejection, synthesis requires completed planning, all requested in-trace evidence reviews, and the full Intent coverage audit. Using the candidate SQL, trace, and recorded evidence, the model reconciles local concerns with downstream computation and checks whether apparently uncovered requirements are implemented elsewhere. Current data may mask semantic conflicts, such as an omitted status restriction that all relevant rows happen to satisfy. If no other condition or schema constraint entails the restriction, a completed review may reject with witness=null, identifying the question requirement and responsible node or missing path without claiming an execution witness. Algorithm 4 summarizes the procedure.

Algorithm 4 Final-impact judgement

Require: $Q, q, R, \tau, \mathcal{E}$ and review status
Ensure: CORRECT or INCORRECT
if a witness w satisfies $\mathrm{EarlyReject}(w)$ **then**
  return INCORRECT
end if
if required reviews are incomplete **then**
  return INCORRECT
end if
Review the candidate and collected evidence against Q
if a validated witness or grounded semantic conflict is found **then**
  return INCORRECT
end if
return CORRECT

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Datasets. We evaluate EviSQL on three datasets. *ROSE-VEC* (Pei et al., 2026) contains 585 human-annotated samples whose semantic correctness labels are established by expert annotators. We further construct *BIRD-VC* by using the corrected SQL queries from *Arcwise-Plat-SQL* (Jin et al., 2026a) to establish evaluation references and collecting predictions from six text-to-SQL methods, including CodeS (Li et al., 2024), CSC-SQL (Sheng & Xu, 2025), DAIL-SQL (Gao et al., 2024), DeepEye-SQL (Li et al., 2026), OmniSQL (Li et al., 2025a), and TA-SQL (Qu et al., 2024), yielding 2,106 candidate SQL queries. *BIRD-CRITIC-SQLite* (Li et al., 2025b) contains 500 realistic SQL issues from database applications. Dataset details are provided in section A.5.

Evaluation Metrics. For *error detection*, we report Cohen's Kappa (κ) (Cohen, 1960), precision, accuracy, recall, and F_1 score. For *diagnosis-guided correction*, we report Execution Accuracy (EX, %) (Yu et al., 2018) and Verified Accuracy (VA, %), where VA measures the percentage of candidate SQL queries that are semantically correct based on human verification. For *error-reason coverage* on *ROSE-VEC*, we report Human Error Coverage (HEC, %), computed using an LLM judge. For evaluation on *BIRD-CRITIC*, we report the Success Rate (SR %) (Li et al., 2025b).

Baselines. For *error detection* and *diagnosis-guided correction*, we compare EviSQL with LLM Verifier, FLEX (Kim et al., 2025), SQLens (Gong et al., 2025), SAR-Agent (Jin et al., 2026a), and ROSE (Pei et al., 2026). For evaluation on *BIRD-CRITIC*, we compare against LLM Repair, SQL-ACT (Li et al., 2025b), SQLFixAgent (Cen et al., 2025), SQLens (Gong et al., 2025), and MapleDoctor (Shen et al., 2026). We follow the official implementations or recommended configurations of all baselines and use GPT-5.6 Sol (OpenAI, 2026) as the backbone. Further baseline and implementation details are provided in section A.6.

5.2 ERROR DETECTION

As shown in table 1, EviSQL achieves the strongest overall performance on both datasets. Cohen's Kappa (κ) measures agreement with correctness labels beyond chance, providing a more robust assessment of verification reliability than accuracy alone. On *ROSE-VEC*, EviSQL achieves a κ score of 70.46 and an F_1 score of 80.49, outperforming the strongest baselines by 10.91 and 8.47 points, respectively. On *BIRD-VC*, it reaches a κ score of 82.48 and an F_1 score of 89.60, with gains of 5.23 and 2.63 points, respectively. Although SQLens achieves the highest recall on both datasets, its substantially lower precision and κ indicate more false positive predictions. In contrast, EviSQL maintains a stronger balance between precision and recall while achieving the highest agreement with correctness labels. The consistent gains in κ further indicate more reliable correctness judgments rather than improvements driven by aggressive error detection.

Table 1: Error detection results on *ROSE-VEC* and *BIRD-VC*. All values are in percentage points.

Method	*ROSE-VEC*					*BIRD-VC*				
	κ	Precision	Accuracy	Recall	F_1	κ	Precision	Accuracy	Recall	F_1
LLM Verifier	56.03	60.39	79.15	88.00	71.63	69.52	78.13	85.00	87.78	82.68
FLEX	59.55	69.89	82.74	74.29	72.02	72.22	86.80	86.75	79.63	83.06
SQLens	32.71	43.77	61.71	**98.29**	60.56	34.97	53.82	64.77	**95.93**	68.95
SAR-Agent	50.39	55.40	75.38	90.86	68.83	77.25	83.05	88.84	91.27	86.97
ROSE	58.89	69.15	82.39	74.29	71.63	68.32	82.10	84.76	80.09	81.08
EviSQL	**70.46**	**70.87**	**86.50**	93.14	**80.49**	**82.48**	**89.92**	**91.55**	89.29	**89.60**

Error Detection across Generators. As shown in table 2, outputs from stronger text-to-SQL models are generally more challenging to verify, since their remaining errors tend to be more subtle as generation accuracy improves. Nevertheless, EviSQL maintains high agreement across all six generators, including high-performing models such as OmniSQL, CSC-SQL, and DeepEye-SQL. This demonstrates its reliable error detection across generators with different levels of accuracy.

Table 2: Error detection across generators on *BIRD-VC*. *BIRD-VC* Execution Accuracy (EX, %) reports execution accuracy on our collected dataset, while *BIRD-VC* Verified Accuracy (VA, %) reports accuracy based on human verification. The remaining rows report Cohen's Kappa κ.

Method	DAIL-SQL	TA-SQL	CodeS	OmniSQL	CSC-SQL	DeepEye-SQL
BIRD-VC EX (%)	42.74	55.27	48.72	68.38	68.09	73.22
BIRD-VC VA (%)	43.02	54.70	47.86	68.66	68.09	72.93
LLM Verifier	81.49	71.88	81.71	68.22	71.15	62.79
FLEX	65.80	58.79	66.76	50.82	57.48	49.82
SQLens	48.55	33.22	45.79	31.58	25.38	22.82
SAR-Agent	80.18	77.15	83.96	71.41	76.00	66.37
ROSE	71.08	66.62	74.82	62.19	67.49	52.62
EviSQL	**84.87**	**82.75**	**85.73**	**78.60**	**77.60**	**77.41**

5.3 Diagnosis-Guided Correction

To assess whether error diagnosis can effectively support downstream SQL repair, we evaluate different methods under the same correction setting. Each SQL is first verified by the corresponding method and is repaired only when predicted as incorrect, using the same LLM repair prompt with the error information provided by each method as guidance, as detailed in section A.6. Unlike detection, EviSQL continues out-of-trace verification after an in-trace error is identified during correction, aiming to uncover additional errors for repair. The original DeepEye-SQL predictions achieve 73.22% EX and 72.93% VA. As shown in table 3, EviSQL achieves the largest improvements on both metrics, reaching 90.31% EX and 92.02% VA, corresponding to absolute gains of 17.09 and 19.09 percentage points, respectively. Accordingly, EviSQL provides more effective diagnostic guidance for downstream SQL correction.

Table 3: Diagnosis-guided correction of 351 DeepEye-SQL predictions on *BIRD-VC* (EX: 73.22%, VA: 72.93%). Parentheses denote absolute gains in percentage points.

Method	EX (%, Δ pp)	VA (%, Δ pp)
LLM Verifier	86.32 (+13.11)	88.03 (+15.10)
FLEX	86.32 (+13.11)	88.03 (+15.10)
SQLens	84.90 (+11.68)	87.75 (+14.81)
SAR-Agent	87.75 (+14.53)	90.88 (+17.95)
ROSE	84.90 (+11.68)	87.75 (+14.81)
EviSQL	**90.31 (+17.09)**	**92.02 (+19.09)**

5.4 Evaluation on *BIRD-CRITIC-SQLite*

Beyond detecting SQL queries that fail to satisfy question requirements, we further evaluate whether EviSQL can help resolve practical user issues on *BIRD-CRITIC-SQLite* Li et al. (2025b). This benchmark requires models to diagnose and resolve user-reported issues involving problematic SQL. For EviSQL, we feed the detected SQL diagnoses into an LLM with a repair prompt to generate the final corrected SQL. As shown in Table 4, EviSQL achieves success rates of 55.63% on Query, 72.00% on Management, 46.81% on Personalization, and 55.60% overall, outperforming all other methods. In comparison, SQLens performs poorly, particularly on Management SQL issues, as its diagnostic mechanism provides limited support for such operations. Overall, EviSQL effectively resolves complex user-reported SQL issues, while its fine-grained diagnoses provide targeted guidance for downstream SQL repair by identifying the specific semantic errors that require correction.

Table 4: SQL issue resolution performance of different methods on *BIRD-CRITIC-SQLite*. Q, M, and P denote Query, Management, and Personalization issues, respectively.

Method	Success Rate (SR %)			
	Q	M	P	ALL
LLM Repair	33.80	29.33	27.66	31.40
SQL-ACT	42.25	64.00	37.59	44.20
SQLFixAgent	51.06	70.67	43.26	51.80
SQLens	47.18	41.33	40.43	44.40
MapleDoctor	51.41	70.67	40.43	51.20
EviSQL	**55.63**	**72.00**	**46.81**	**55.60**

5.5 Human-Annotated Error Coverage

For incorrect SQL, *ROSE-VEC* provides human comments describing the underlying errors (Pei et al., 2026), allowing us to evaluate whether a verifier recovers the annotated semantic defects. Human-Annotated Error Coverage (HEC) is defined in Appendix A.6. As shown in table 5, EviSQL achieves the highest HEC of 80.32%, outperforming all other methods. In contrast, SQLens achieves only 49.37% HEC despite its high error-detection recall.

Table 5: Human-Annotated Error Coverage (HEC, %) on *ROSE-VEC* across different methods.

Method	LLM Verifier	FLEX	SQLens	SAR-Agent	ROSE	EviSQL
HEC (%)	68.17	74.76	49.37	75.24	72.87	**80.32**

5.6 Ablation and Cost Analysis

Component Ablation. We evaluate the contributions of three components of EviSQL on *ROSE-VEC* and *BIRD-VC*, reporting Cohen's κ in figure 4. The complete framework achieves κ scores of 70.46 and 82.48, respectively. Removing *In-Trace Verification* disables verification of suspicious transitions within the candidate computation, reducing performance by 2.48 and 2.66 points on the two datasets. Removing *Out-of-Trace Exploration* restricts evidence search to the candidate computation, with the largest degradation of 4.31 points observed on *ROSE-VEC*. Together, the two evidence-search paths provide complementary coverage of errors within the candidate computation and semantics omitted from it. Removing *Final-Impact Judgement* causes smaller but consistent performance drops, while its inclusion prevents intermediate discrepancies that do not affect the final answer from being treated as decisive evidence.

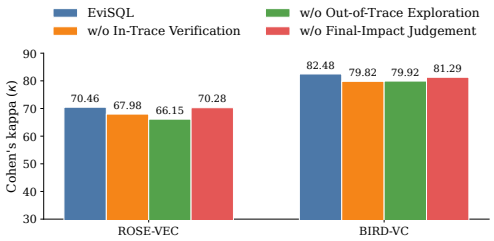


Figure 4: Ablation study of EviSQL.

Cost Analysis. We report the inference cost of EviSQL per sample in terms of SQL executions, SQL execution time, LLM calls, and token consumption. As shown in table 6, *Out-of-Trace Exploration* incurs the largest LLM cost, while removing it reduces average calls to 2.48 and 2.37 on the two datasets. *Final-Impact Judgement* provides modest accuracy gains but improves efficiency, reducing LLM calls by about 12% and token consumption by 15–16% compared with its ablated variant.

Table 6: Cost comparison on *ROSE-VEC* and *BIRD-VC*, with all metrics reported as per-sample averages. Execs denotes the number of SQL executions, Times denotes SQL execution time in milliseconds, Calls denotes the number of LLM calls, and Tokens denotes token consumption.

Method	*ROSE-VEC*				*BIRD-VC*			
	Execs	Times	Calls	Tokens	Execs	Times	Calls	Tokens
EviSQL	2.65	166.98	3.58	19,738	2.40	239.19	3.41	20,622
EviSQL w/o *In-Trace Verification*	2.88	115.69	3.09	17,926	2.59	299.43	3.02	18,860
EviSQL w/o *Out-of-Trace Exploration*	2.12	54.49	2.48	12,603	2.03	78.29	2.37	13,325
EviSQL w/o *Final-Impact Judgement*	2.66	120.54	4.05	23,338	2.41	234.56	3.88	24,659

6 Conclusion

We present EviSQL, a reference-free text-to-SQL verification framework that searches for execution evidence both within and beyond the computation expressed by the candidate SQL. It explicitly examines both the semantics expressed by candidate SQL and the question requirements that the candidate may omit. Across challenging text-to-SQL settings, EviSQL consistently improves semantic error detection and enables more effective diagnosis-guided correction. A task-adapted trace repair pipeline also achieves strong performance in resolving practical user-reported SQL issues. More broadly, reliable text-to-SQL verification requires reasoning not only about what a candidate SQL computes, but also about what the question requires beyond that computation.

AI USE STATEMENT

In this work, we used generative AI tools to assist with method implementation, experimental data analysis, literature search, and manuscript polishing. Codex (OpenAI, 2025) was used to assist with algorithm implementation and experimental data analysis. All generated code was carefully tested and verified by two authors, while all analysis procedures and results were independently reviewed and validated by the authors. GPT-5.6 Sol (OpenAI, 2026) was used to assist with literature search, identification of related work, manuscript polishing, and correction of grammatical and typographical errors. All literature identified with AI assistance was manually verified against the original sources before being added to the references, and all AI assisted manuscript content was carefully reviewed and revised by the authors. We take full responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

To support reproducibility, we provide comprehensive documentation of the method and experimental settings. A detailed description of EviSQL, including computation trace construction, node-local validation, Intent coverage auditing, and evidence-gated final judgement, is provided in section 4. Experimental settings, evaluation protocols, model configurations, and implementation details are described in section 5 and the appendix. The appendix describes the four prompt stages of EviSQL and includes the direct verifier and repair baseline prompts. The full EviSQL prompt templates are available in the `prompts` folder at the root of the anonymized project. We further document the benchmark processing and release the complete *BIRD-VC* dataset with all annotations, including identified errors, to facilitate reproduction of the refined benchmark results. Finally, anonymized source code, experiment scripts, and experimental resources are available at `https://anonymous.4open.science/r/EviSQL-30D4`.

REFERENCES

Jipeng Cen, Jiaxin Liu, Zhixu Li, and Jingjing Wang. Sqlfixagent: Towards semantic-accurate text-to-sql parsing via consistency-enhanced multi-agent collaboration. In *Thirty-Ninth AAAI Conference on Artificial Intelligence, Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence, Fifteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2025*, pp. 49–57, 2025.

Shijie Chen, Ziru Chen, Huan Sun, and Yu Su. Error detection for text-to-sql semantic parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, volume EMNLP 2023 of *Findings of ACL*, pp. 11730–11743, 2023a.

Yongnan Chen, Zhuo Chang, Shijia Gu, Yuanhang Zong, Mei Zhang, Shiyu Wang, Zixiang He, Hongzhi Chen, Wei Jin, and Bin Cui. ADEPT-SQL: A high-performance text-to-sql application for real-world enterprise-level databases. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations), ACL 2025*, pp. 275–283, 2025.

Ziru Chen, Shijie Chen, Michael White, Raymond J. Mooney, Ali Payani, Jayanth Srinivasa, Yu Su, and Huan Sun. Text-to-sql error correction with language models of code. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), ACL 2023*, pp. 1359–1372, 2023b.

Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960.

Yaxun Dai, Wenxuan Xie, Xialie Zhuang, Tianyu Yang, Ziyi Liu, Haiqin Yang, Yiying Yang, Yuhang Zhao, Pingfu Chao, and Wenhao Jiang. Reex-sql: Reasoning with execution-aware reinforcement learning for text-to-sql. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2026*, pp. 824–847. Association for Computational Linguistics, 2026.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-sql empowered by large language models: A benchmark evaluation. *Proceedings of the VLDB Endowment*, 17(5):1132–1145, 2024.

Pushpendu Ghosh, Aryan Jain, and Promod Yenigalla. Sqlgenie: A practical LLM based system for reliable and efficient SQL generation. In Georg Rehm and Yunyao Li (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 1004–1012. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.ACL-INDUSTRY.71. URL https://doi.org/10.18653/v1/2025.acl-industry.71.

Yue Gong, Chuan Lei, Xiao Qin, Kapil Vaidya, Balakrishnan Narayanaswamy, and Tim Kraska. Sqlens: An end-to-end framework for error detection and correction in text-to-sql. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025*, 2025.

Tengjun Jin, Yoojin Choi, Yuxuan Zhu, and Daniel Kang. Pervasive annotation errors break text-to-sql benchmarks and leaderboards. *Proceedings of the VLDB Endowment*, 19(5):931–944, 2026a.

Tengjun Jin, Yoojin Choi, Yuxuan Zhu, and Daniel Kang. Text-to-sql benchmarks are broken: An in-depth analysis of annotation errors. In *16th Conference on Innovative Data Systems Research, CIDR 2026*, 2026b.

Heegyu Kim, Taeyang Jeon, Seunghwan Choi, Seungtaek Choi, and Hyunsouk Cho. FLEX: expert-level false-less execution metric for text-to-sql benchmark. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics*, pp. 4448–4475, 2025.

Rocky Klopfenstein, Yang He, Andrew Tremante, Yuepeng Wang, Nina Narodytska, and Haoze Wu. Spotit: Evaluating text-to-SQL evaluation with formal verification. In *The Fourteenth International Conference on Learning Representations*, 2026.

Gyubok Lee, Hyeonji Hwang, Seongsu Bae, Yeonsu Kwon, Woncheol Shin, Seongjun Yang, Minjoon Seo, Jong-Yeup Kim, and Edward Choi. EHRSQL: A practical text-to-sql benchmark for electronic health records. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022*, 2022.

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. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows. In *The Thirteenth International Conference on Learning Representations*, 2025.

Boyan Li, Chong Chen, Zhujun Xue, Yinan Mei, and Yuyu Luo. Deepeye-sql: A software-engineering-inspired text-to-sql framework. *Proceedings of the ACM on Management of Data*, 4 (3):158:1–158:28, 2026.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. Codes: Towards building open-source language models for text-to-sql. *Proceedings of the ACM on Management of Data*, 2(3):127, 2024.

Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. Omnisql: Synthesizing high-quality text-to-sql data at scale. *Proceedings of the VLDB Endowment*, 18(11):4695–4709, 2025a.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chen-Chuan Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can LLM already serve as A database interface? A big bench for large-scale database grounded text-to-sqls. In *Advances in Neural Information Processing Systems*, 2023.

Jinyang Li, Xiaolong Li, Ge Qu, Per Jacobsson, Bowen Qin, Binyuan Hui, Shuzheng Si, Nan Huo, Xiaohan Xu, Yue Zhang, Ziwei Tang, Yuanshuai Li, Florensia Widjaja, Xintong Zhu, Feige Zhou, Yongfeng Huang, Yannis Papakonstantinou, Fatma Özcan, Chenhao Ma, and Reynold Cheng. SWE-SQL: illuminating LLM pathways to solve user SQL issues in real-world applications. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025*, 2025b.

OpenAI. Introducing codex. https://openai.com/index/introducing-codex/, 2025. Accessed: 2026-08-22.

OpenAI. GPT-5.6: Frontier intelligence that scales with your ambition. https://openai.com/index/gpt-5-6/, 2026. Accessed: 2026-08-22.

Wenqi Pei, Shizheng Hou, Boyan Li, Chen Han, Zhichao Shi, and Yuyu Luo. ROSE: an intent-centered evaluation metric for NL2SQL. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, pp. 5682–5709, 2026.

Mohammadreza Pourreza and Davood Rafiei. DIN-SQL: decomposed in-context learning of text-to-sql with self-correction. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023*, 2023a.

Mohammadreza Pourreza and Davood Rafiei. Evaluating cross-domain text-to-sql models and benchmarks. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 1601–1611, 2023b.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan Ö. Arik. CHASE-SQL: multi-path reasoning and preference optimized candidate selection in text-to-sql. In *The Thirteenth International Conference on Learning Representations, ICLR 2025*, 2025.

Ge Qu, Jinyang Li, Bowen Li, Bowen Qin, Nan Huo, Chenhao Ma, and Reynold Cheng. Before generation, align it! A novel and effective strategy for mitigating hallucinations in text-to-sql generation. In *Findings of the Association for Computational Linguistics, ACL 2024*, volume ACL 2024 of *Findings of ACL*, pp. 5456–5471, 2024.

Jiawei Shen, Chengcheng Wan, Ruoyi Qiao, Jiazhen Zou, Hang Xu, Yuchen Shao, Yueling Zhang, Weikai Miao, and Geguang Pu. Understanding, detecting, and repairing real-world in-context-learning-based text-to-sql errors. *Proceedings of the ACM on Software Engineering*, 3(FSE): 3722–3745, 2026.

Lei Sheng and Shuai-Shuai Xu. CSC-SQL: corrective self-consistency in text-to-sql via reinforcement learning. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, IJCNLP-AACL 2025*, pp. 1473–1496, 2025.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. Chess: Contextual harnessing for efficient sql synthesis. *arXiv preprint arXiv:2405.16755*, 2024.

Andrew Tremante, Yang He, Rocky Klopfenstein, Yuepeng Wang, Nina Narodytska, and Haoze Wu. Spotit+: Verification-based text-to-sql evaluation with database constraints. *arXiv preprint arXiv:2603.04334*, 2026.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. MAC-SQL: A multi-agent collaborative framework for text-to-sql. In *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025*, pp. 540–557, 2025a.

Dingzirui Wang, Longxu Dou, Xuanliang Zhang, Qingfu Zhu, and Wanxiang Che. DAC: decomposed automation correction for text-to-sql. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 385–402, 2025b.

Niklas Wretblad, Fredrik Riseby, Rahul Biswas, Amin Ahmadi, and Oskar Holmström. Understanding the effects of noise in text-to-SQL: An examination of the BIRD-bench benchmark. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 356–369, 2024.

Wenbo Xu, Haifeng Zhu, Liang Yan, Chuanyi Liu, Peiyi Han, Shaoming Duan, and Jeff Z. Pan. TS-SQL: Test-driven self-refinement for text-to-SQL. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 2864–2889, 2025.

Yicun Yang, Zhaoguo Wang, Yu Xia, Zhuoran Wei, Haoran Ding, Ruzica Piskac, Haibo Chen, and Jinyang Li. Automated validating and fixing of text-to-sql translation with execution consistency. *Proceedings of the ACM on Management of Data*, 3(3):134:1–134:28, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023*, 2023.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3911–3921, 2018.

Ruiqi Zhong, Charlie Snell, Dan Klein, and Jason Eisner. Non-programmers can label programs indirectly via active examples: A case study with text-to-sql. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 5126–5152. Association for Computational Linguistics, 2023.