ParSEval: Interactive Counterexample-driven Evaluation for Text-to-SQL

Chunyu Chen
Simon Fraser University
chunyu_chen@sfu.ca

Zhengjie Miao*
Simon Fraser University
zhengjie@sfu.ca

Yong Zhang
Huawei Technologies
yong.zhang3@huawei.com

Jiannan Wang
Tsinghua University
jnwang@tsinghua.edu.cn

ABSTRACT

Evaluating Text-to-SQL systems on a single static database instance often misses semantic errors that arise only under specific data conditions, such as duplicates, NULLs, or integrity assumptions. As a result, developers may obtain misleading execution-accuracy results and struggle to understand their systems' true behavior. We present ParSEval, an interactive demonstration system for configurable, counterexample-driven Text-to-SQL evaluation. Given schemas, ground-truth queries, and model predictions, ParSEval automatically generates test databases under configurable integrity assumptions, evaluates query pairs across multiple instances, and exposes concrete witness instances when execution results differ. Through a live dashboard, users can compare model runs, inspect disagreement patterns under different evaluation settings, such as set versus bag semantics and key-foreign-key constraints, and drill down into individual failures through generated counterexamples. We demonstrate how ParSEval helps developers evaluate iterative model improvements on private benchmarks and bootstrap customized Text-to-SQL test suites without manually crafting diagnostic database instances.

PVLDB Reference Format:
Chunyu Chen, Zhengjie Miao, Yong Zhang, and Jiannan Wang. ParSEval: Interactive Counterexample-driven Evaluation for Text-to-SQL. PVLDB, 19(12): 4846 - 4849, 2026.
doi:10.14778/3827998.3828137

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/sfu-db/ParSEval/tree/webui.

1 INTRODUCTION

Text-to-SQL systems are commonly evaluated by executing a predicted query and a ground-truth query on a test database and comparing their outputs. This execution-based paradigm is simple and widely adopted by benchmarks such as Spider [5] and BIRD [3]. A key challenge, however, is that the effectiveness of this evaluation depends heavily on the database instances used for testing. If the chosen instances do not expose the relevant query behaviors, important differences between two queries may remain hidden.

Constructing such test databases is expensive. To reveal subtle SQL errors, the test database often needs to include specific cases, such as NULL values, duplicates, or data that affects join and aggregation results [4]. Manually creating these instances is slow and labor-intensive. Randomly generated data is also often insufficient, because it may fail to trigger the behaviors that distinguish two non-equivalent queries [6]. As a result, existing evaluation workflows usually rely on a small number of fixed instances and therefore cover only limited cases.

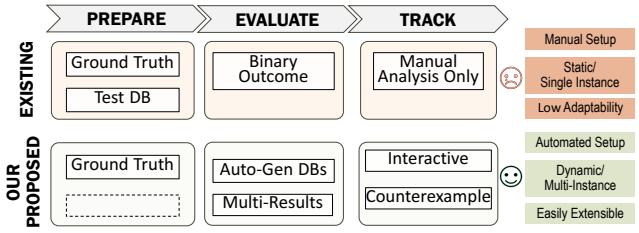

Figure 1: Comparison between fixed-database Text-to-SQL evaluation and ParSEval's generated multi-instance, counterexample-driven workflow.

This design also limits what users can learn from the evaluation. When two queries return the same result on one or a few test instances, the evaluation still cannot tell whether the agreement is robust under other data conditions. When the two queries return different results, users often need to inspect the queries and the data manually to understand what causes the difference. Existing workflows therefore provide limited help for explaining failure cases, because they do not return concrete instances showing where the two queries differ.

To address this challenge, we present ParSEval, an interactive system for multi-instance, counterexample-driven Text-to-SQL evaluation. Given schema, ground-truth queries, and model predictions, ParSEval automatically generates test databases under configurable integrity assumptions, evaluates query pairs across multiple instances, and surfaces concrete witness databases for debugging. Through a live web interface, attendees can upload Text-to-SQL predictions, compare model runs, analyze disagreement patterns under settings such as set versus bag semantics, and inspect counterexamples for failing pairs. In this way, ParSEval turns evaluation from checking query outputs on a few fixed databases into a process that can systematically test more cases and help users analyze failed ones.

Figure 1 illustrates the difference between ParSEval and existing workflows. Existing pipelines typically use manually created databases, evaluate query pairs on static instances, and return a simple pass/fail result with limited feedback. In contrast, ParSEval automatically generates database instances and evaluates query pairs under different settings. It also returns concrete databases for

*Corresponding author.

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828137

failed cases, allowing users to inspect the data conditions under which two queries differ, as illustrated by the following example.

Example 1. *Consider a ground-truth query Q_1 that counts the number of schools in a specific county using* `COUNT(School)` *and a predicted query Q_2 that instead uses* `COUNT(CDSCode)`*. On the test database instance I_1 shown in Figure 2, the two queries diverge as the* `NULL` *value in* `School`*, since SQL* `COUNT()` *ignores* `NULL` *values.*

```
Q1: SELECT COUNT(T1.School)
    FROM frpm AS T1
    INNER JOIN schools AS T2 ON T1.CDSCode = T2.CDSCode
    WHERE T1.County = 'Amador'

Q2: SELECT COUNT(T1.CDSCode)
    FROM frpm AS T1
    INNER JOIN schools AS T2 ON T1.CDSCode = T2.CDSCode
    WHERE T1.County = 'Amador'
```

CDSCode	School	County
1	Amador High	Amador
2	NULL	Amador
3	Pioneer Elementary	Amador

(a) `frpm` *relation (T_1)*

CDScode	···
1	···
2	···
3	···

(b) `schools` *relation (T_2)*

Figure 2: An example database instance I_1. Only columns used in the queries are shown.

Relation to prior work. Our demonstration focuses on the end-to-end interactive experience of ParSEval. While our prior work [1] introduced the core plan-aware database generation technique for SQL equivalence evaluation, this demo highlights the broader interactive workflow: project-based evaluation management, configurable assumption settings, cross-run comparison, exploration of disagreement patterns, and drill-down debugging with concrete counterexample databases over Text-to-SQL evaluation runs.

Live demonstration scenarios. In the live demonstration, attendees will follow two usage scenarios. In the first step, a developer uploads predictions from multiple Text-to-SQL models, compares evaluation runs, and inspects concrete failures through generated witness databases. In the second step, a user bootstraps a customized benchmark by uploading a schema and query set, configuring data assumptions, and synthesizing diagnostic test databases without manual data crafting. Together, these scenarios highlight ParSEval's core capabilities: configurable multi-instance evaluation, interactive comparison across runs, and concrete debugging through generated counterexamples.

2 BACKGROUND AND CONFIGURABLE EVALUATION

Query equivalence and Text-to-SQL Evaluation. Given a database schema $\mathbf{R}$, a ground-truth query Q_g, and a predicted query Q_p, Text-to-SQL evaluation aims to determine whether Q_p is semantically equivalent to Q_g. In practice, however, it is infeasible to check equivalence over all possible database instances. Existing evaluation, therefore, approximates this problem by executing both queries on one or a small number of test database instances and comparing their outputs. Let $S = \{D_1, D_2, \cdots, D_n\}$ denote a test suite of database instances over $\mathbf{R}$, and let Γ denote a set of integrity assumptions. Two queries are considered equivalent under Γ if they return identical results on all database instances over $\mathbf{R}$ that satisfy Γ. Consequently, the effectiveness of evaluation depends critically on how well the chosen test suite exposes semantically meaningful differences between the ground-truth queries and the predictions.

Plan-Aware Database Generation. Constructing such a test suite S is particularly challenging in the context of Text-to-SQL, where model-generated queries can vary widely and cannot be exhaustively enumerated. If the test database instance does not expose relevant query behaviors, evaluation results can be misleading. Our prior work [1] introduced the core plan-aware database generation technique underlying ParSEval. The key idea is to derive *coverage constraints* from the logical query plan, where each constraint captures a target operator behavior that the generated database instance should exercise, such as join outcomes, predicate branches, group-by behaviors, and NULL-sensitive cases. These coverage targets are encoded as constraints and solved by an SMT solver to synthesize concrete database instances.

While effective for generating diagnostic instances for individual ground truth queries, this approach can become inefficient at the scale of a full Text-to-SQL dataset containing hundreds or thousands of queries. To improve efficiency, ParSEval incrementally reuses previously generated instances when they already satisfy the coverage needs of newly evaluated queries, thereby avoiding redundant constraint solving and data synthesis across relevant queries, as illustrated in the example below.

Example 2. *Consider a query Q_3,* `SELECT School FROM frpm WHERE County = 'Orange'`*, share the same schema as Q_1 in Example 1. When ParSEval processes Q_3, instead of populating a new instance from scratch, it first evaluates the existing instance I_1 (previously generated for Q_1) against the coverage constraints of Q_3. Since I_1 already contains tuples where* `County` *is not 'Orange', these existing rows automatically satisfy the requirements for Q_3's negative predicate branch. By identifying this overlap, ParSEval skips the costly SMT call and only generates additional data for the remaining uncovered branches (e.g., the positive predicate branch and the project branch).*

Configurable Evaluation Settings. Building on this foundation, ParSEval extends the database instance generator into a configurable evaluation workflow. In addition to generating database instances that expose non-equivalent cases, the system allows users to specify evaluation settings and integrity assumptions, such as set versus bag semantics, key-foreign-key constraints, and NULL-related assumptions. This flexibility is important because non-equivalent queries in Text-to-SQL can still yield practically meaningful results under particular data assumptions. For example, the queries Q_1 and Q_2 in Example 1 produce identical outputs when the `School` column is assumed to be `NOT NULL`, even though they are not strictly semantically equivalent. In such cases, the predicted query may still satisfy the underlying user intent, even if it differs from the ground-truth query under unrestricted semantics.

Accordingly, ParSEval does not generate database instances solely from the declared schema. Instead, it allows users to configure additional constraints and evaluation settings, thereby defining multiple evaluation configurations for the same query pair. This enables users to compare strict semantic equivalence with more pragmatic notions of correctness under domain-specific data assumptions.

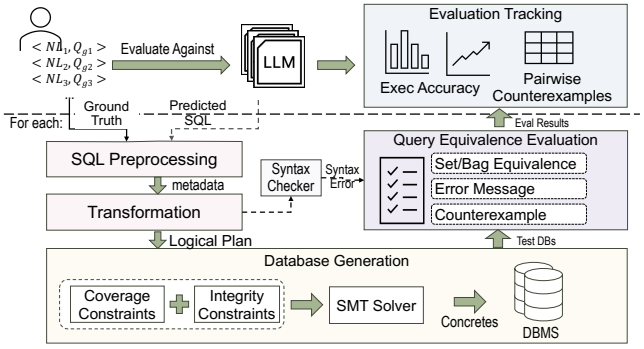

Figure 3: Architecture of ParSEval

3 SYSTEM OVERVIEW

As illustrated in Figure 3, ParSEval consists of three main modules: (1) a database generation module that parses input queries and constructs database instances guided by coverage constraints, (2) an equivalence evaluation module that compares the execution results of query pairs over the generated databases, and (3) an evaluation tracking module that helps developers analyze and compare the results of different runs.

Project Abstraction. As an end-to-end Text-to-SQL evaluation framework, ParSEval organizes data into reusable, trackable evaluation runs via a project abstraction. A project is defined as a directory that contains user inputs, such as model-generated queries and schemas, and system outputs, such as generated database instances and evaluation reports.

Input Preprocessing. ParSEval accepts either a single query Q_1 or a pair of SQL queries Q_1 and Q_2 over a given schema **R**. It parses and normalizes the input queries, detects syntax errors early in the pipeline, and converts them into logical query plans. These plans are then transformed into an intermediate representation that captures operator-level semantics and forms the basis for subsequent coverage analysis.

Coverage-Guided Database Generation. Using the extracted logical plan, ParSEval traverses the query plan bottom-up and derives coverage constraints from the behaviors of relational operators. Intuitively, these constraints capture the input conditions under which an operator produces or does not produce output, such as successful or failed joins and predicate branches. ParSEval then formulates database generation as a constraint satisfaction problem and solves it using an SMT solver [2]. The resulting satisfying assignment determines the concrete data values needed in a synthesized database instance to cover a target execution path in the query plan.

Equivalence Evaluation. ParSEval executes Q_1 and Q_2 on the generated test suite. Each database instance represents a different combination of coverage conditions. A query pair is considered equivalent under a given instance if the two queries produce identical results on that instance. Furthermore, ParSEval supports configurable set- and bag-semantics when comparing outputs. This allows developers to distinguish between unordered set agreement and stricter bag-sensitive equivalence when duplicates matter. ParSEval also records the concrete database instance that either disproves equivalence or witnesses agreement for the query pair.

Evaluation Tracking. After evaluation, ParSEval aggregates results across generated database instances and presents them through an interactive heatmap-based visualization together with concrete counterexamples. This view highlights the configurations under which query pairs agree or diverge, helping users quickly identify patterns in query behavior. For developers iteratively improving Text-to-SQL models, ParSEval also tracks performance changes across runs, showing whether a new model version resolves previously observed error types. This helps users compare different Text-to-SQL approaches and analyze conflicts between model assumptions and actual database schema.

4 DEMONSTRATION SCENARIOS

4.1 Text-to-SQL Model Evaluation

In the first scenario, we demonstrate how ParSEval supports iterative evaluation and debugging for Text-to-SQL systems. The developer collects predictions from multiple Text-to-SQL models and uses ParSEval to compare overall model performance and analyze cases in which a predicted query is not semantically equivalent to the ground-truth query.

Step 1. Get Started. The developer creates a new project through the web interface to track evaluation runs. After creating the project, the developer uploads Text-to-SQL results collected from multiple models, for example, in JSON format containing schema information and model-generated query pairs, as shown on the left of Figure 4. ParSEval automatically parses and preprocesses each query pair in the background, extracting the logical plans and schema constraints needed for database generation. The **Run** and **Status** columns (①) show the status of each submitted evaluation.

Step 2. Cross-Run Performance Comparison. Next, the developer can compare performance across model iterations by clicking the comparison button above the evaluation results list. ParSEval visualizes execution accuracy using line charts. The developer can also use the dataset and model filters to aggregate metrics across different datasets or models (②), providing a high-level view of overall progress and remaining blind spots.

Step 3. Execution Accuracy Analysis. To inspect a specific evaluation in more detail, the developer selects a run to view its detailed performance metrics (③). ParSEval summarizes execution accuracy using an interactive bar chart, allowing the developer to analyze performance variation under different evaluation settings, such as set versus bag semantics. The developer can also click the **Compare To** button to place metrics from two runs side by side, making it easy to determine whether a model update improved correctness consistently or only under specific assumptions. To further examine how equivalence outcomes vary across settings, the developer can inspect the confusion heatmap (④), which shows where query pairs agree or disagree across configurations, for example highlighting pairs that are equivalent under set semantics but fail under bag semantics.

Step 4. Interactive Error Analysis. To debug specific prediction failures, the developer drills down into non-equivalent query pairs. ParSEval presents an automatically generated counterexample, i.e., a database instance that disproves the predicted query's equivalence to the ground-truth query, together with the predicted and ground-truth SQL queries (⑤, ⑥). The developer can inspect

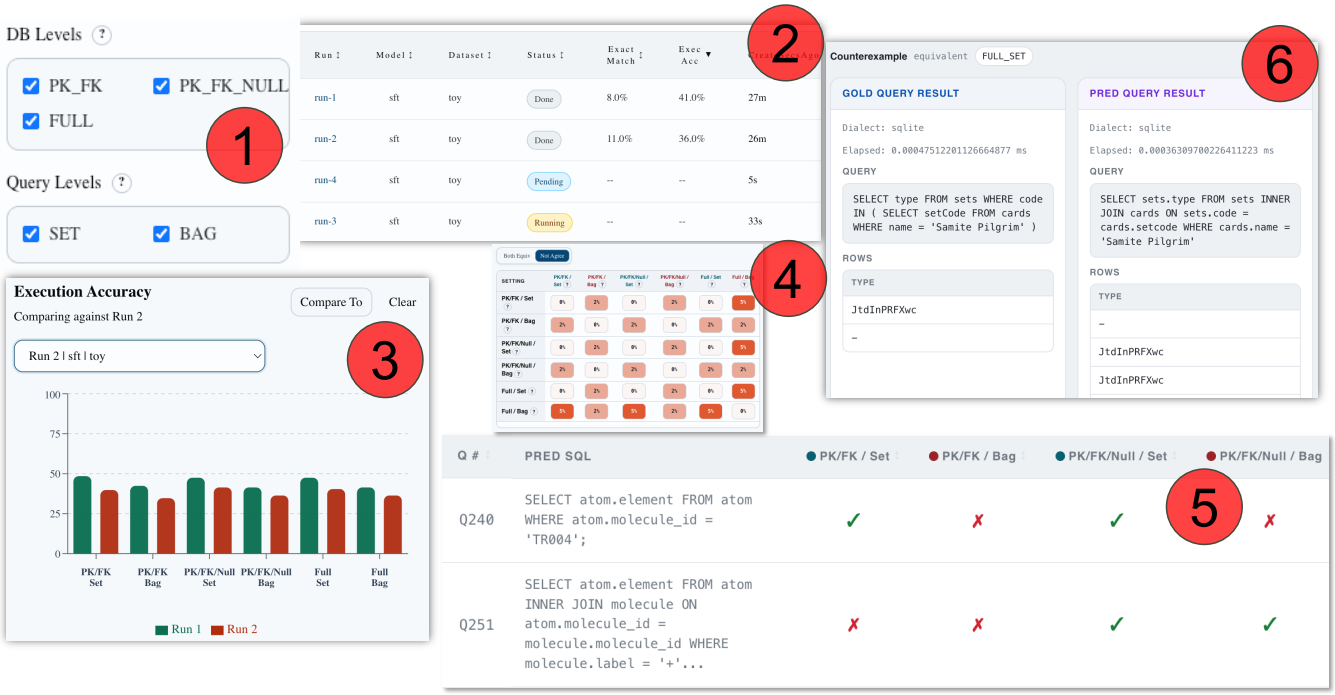

Figure 4: The User Interfaces of ParSEval

the data conditions that trigger the divergence and identify the specific semantic gap between the model prediction, the intended query, and the underlying schema assumptions.

4.2 Bootstrapping Custom Benchmarks

We also demonstrate how ParSEval supports the construction of new domain-specific Text-to-SQL benchmarks. Traditionally, this process requires not only writing ground-truth queries for a curated set of natural-language questions, but also manually crafting database instances with sufficient variety to distinguish predicted queries. ParSEval substantially reduces this data-preparation effort by automatically generating diagnostic test databases from the schema and query workload.

Step 1. Get started. The developer begins by uploading a custom database schema (e.g., from a financial or healthcare domain) alongside a set of natural language questions and corresponding ground-truth query pairs (Q_{NL}, Q_g). ParSEval uses SQLGlot to extract constraints from the input schema, such as `NOT NULL`, primary-key, and foreign-key constraints. The developer can choose whether generated instances are required to satisfy these constraints. For example, if the schema declares a `School` or `Department` column as `NOT NULL`, generated instances respect that requirement when constraint enforcement is enabled. Separately, the developer can select Bag or Set semantics for query-result comparison.

Step 2. Automated Test Databases Setup. Once the settings are configured, ParSEval automatically generates the diagnostic test suite. Instead of manually crafting data, the developer can directly inspect the generated database instances and use them as a reusable evaluation suite under the selected settings.

By the end of this scenario, attendees will see how ParSEval constructs a reusable test suite directly from a schema and query set, making customized benchmark preparation substantially easier.

ACKNOWLEDGMENTS

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

[1] Chunyu Chen, Zhengjie Miao, Yong Zhang, and Jiannan Wang. 2025. ParSEval: Plan-aware Test Database Generation for SQL Equivalence Evaluation. *Proceedings of the VLDB Endowment* 18, 11 (2025), 4750–4762.

[2] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 337–340.

[3] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems* 36 (2024).

[4] Shetal Shah, S Sudarshan, Suhas Kajbaje, Sandeep Patidar, Bhanu Pratap Gupta, and Devang Vira. 2011. Generating test data for killing SQL mutants: A constraint-based approach. In *2011 IEEE 27th International Conference on Data Engineering*. IEEE, 1175–1186.

[5] 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 *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 3911–3921.

[6] Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic Evaluation for Text-to-SQL with Distilled Test Suites. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 396–411.