NL2SQL-BUGs: A Benchmark for Detecting Semantic Errors in NL2SQL Translation

Xinyu Liu
The Hong Kong University of Science
and Technology (Guangzhou)
Guangzhou, China
xliu371@connect.hkust-gz.edu.cn

Shuyu Shen
The Hong Kong University of Science
and Technology (Guangzhou)
Guangzhou, China
sshen190@connect.hkust-gz.edu.cn

Boyan Li
The Hong Kong University of Science
and Technology (Guangzhou)
Guangzhou, China
bli303@connect.hkust-gz.edu.cn

Nan Tang*
The Hong Kong University of Science
and Technology (Guangzhou)
Guangzhou, China
nantang@hkust-gz.edu.cn

Yuyu Luo
The Hong Kong University of Science
and Technology (Guangzhou)
Guangzhou, China
yuyuluo@hkust-gz.edu.cn

Abstract

Natural Language to SQL (*i.e.,* NL2SQL) translation is crucial for democratizing database access, but even state-of-the-art models frequently generate semantically incorrect SQL queries, hindering the widespread adoption of these techniques by database vendors. While existing NL2SQL benchmarks primarily focus on correct query translation, we argue that a benchmark dedicated to identifying common errors in NL2SQL translations is equally important, as accurately detecting these errors is a prerequisite for any subsequent correction – whether performed by humans or models. To address this gap, we propose NL2SQL-BUGs, *the first* benchmark dedicated to detecting and categorizing semantic errors in NL2SQL translation. NL2SQL-BUGs adopts a two-level taxonomy to systematically classify semantic errors, covering 9 main categories and 31 subcategories. The benchmark consists of 2,018 expert-annotated instances, each containing a natural language query, database schema, and SQL query, with detailed error annotations for semantically incorrect queries. Through comprehensive experiments, we demonstrate that current large language models exhibit significant limitations in semantic error detection, achieving an average detection accuracy of 75.16%. Specifically, our method successfully detected **106** errors (accounting for **6.91%**) in BIRD, a widely-used NL2SQL dataset, which were previously undetected annotation errors. This highlights the importance of semantic error detection in NL2SQL systems. The benchmark is publicly available at https://nl2sql-bugs.github.io/.

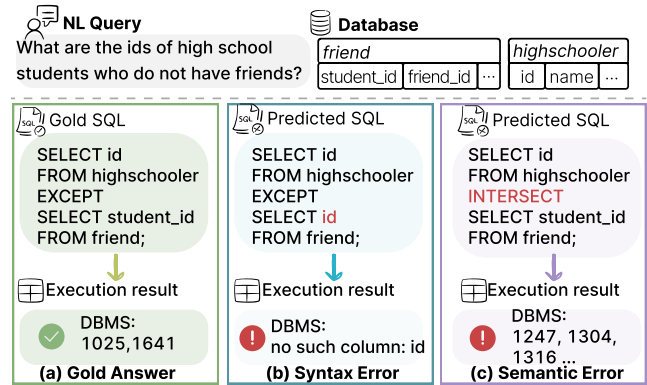*Nan Tang is the corresponding author.

Figure 1: Error Types in Generated NL2SQL Queries.

CCS Concepts

• **Information systems** → **Query languages**; • **Computing methodologies** → **Natural language generation**.

Keywords

Text-to-SQL, Large Language Model, Interface for Databases

1 Introduction

Over the past few decades, significant progress has been made in translating natural language queries (NL) into corresponding SQL queries, commonly referred to as NL2SQL (or Text-to-SQL) [9, 19], to democratize data analysis [20–23, 30, 36, 41]. Recent advancements, particularly with large language models (LLMs) [46, 50], have greatly improved the ability to understand complex queries and generate accurate SQL translations [3, 6, 28, 32], leading to better performance on benchmarks like Spider [45] and BIRD [15]. However, despite these advancements, the current state-of-the-art
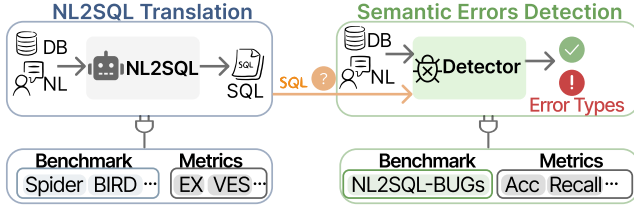
Figure 2: Semantic Errors Detection for NL2SQL.

models still achieve only around 75% accuracy on BIRD [15]. This indicates that roughly 25% of the cases fail because of **NL2SQL translation errors**, even in these curated benchmarks. In real-world production environments, where databases are more complex [9, 33, 47] and user inputs are more diverse [5, 24], model performance is likely to degrade further.

NL2SQL translation errors can be categorized into two types: *syntax errors* and *semantic errors*. Syntax errors, as shown in Figure 1(b), occur when the SQL query violates SQL grammar rules or contains invalid references to tables, columns, or operators. These errors are relatively easy to detect because they trigger immediate execution failures and return error messages from the database.

In contrast, **semantic errors are syntactically correct but fail to reflect the intended meaning of the user's query**, which are thus more subtle and harder to detect. For example, as shown in Figure 1(c), consider a situation where the user asks for students who do not have friends, but the generated SQL query uses INTERSECT instead of EXCEPT, returning students who have friends rather than those without friends.

All SQL compilers can detect syntax SQL errors, but they cannot detect semantic SQL errors. As a result, the performance gap in NL2SQL systems is largely attributed to semantic errors, which constitute a significant portion of the errors in both the Spider and BIRD datasets. Specifically, after examining the erroneous translations produced by the CodeS NL2SQL model [14], we found that 168 out of 170 errors (**98.8%**) in Spider and 658 out of 667 errors (**98.7%**) in BIRD were semantic. This overwhelming proportion underscores the critical role semantic errors play in NL2SQL systems.

Given the subtle nature of semantic errors, they are more challenging to detect, as they can result in incorrect outputs without any visible errors during execution. This makes semantic error detection particularly important, as accurately identifying these errors is the *prerequisite* for any subsequent correction. Currently, detecting these errors is often carried out manually by experts. However, to scale this process and reduce human reliance, we need models that can assist or even replace experts in detecting these errors. Just as modern DBMSs include SQL syntax checkers to detect and correct syntax errors, **NL2SQL systems require a similar mechanism – an NL2SQL semantic detector – to identify errors in the meaning of generated SQL queries.** As shown in Figure 2, the task of **NL2SQL semantic errors detection** aims to detect semantic discrepancies between the natural language query, the generated SQL query, and the database schema. Once semantic errors are detected, these discrepancies can be corrected by triggering error alarms for further inspection, revising the query generation process, or applying automated correction mechanisms.

To evaluate the capabilities of NL2SQL semantic errors detection, we introduce NL2SQL-BUGs, *the first* benchmark designed to identify and classify semantic errors in NL2SQL translation. NL2SQL-BUGs adopts a two-level taxonomy to classify semantic errors. It covers nine main categories and numerous subcategories. NL2SQL-BUGs consists of 2,018 expert-annotated instances. Each instance contains a natural language query, a database schema, and the corresponding SQL query, along with detailed error annotations for semantically incorrect SQL queries.

Contributions. We make the following contributions.

(1) Problem Definition. We formally define the task of semantic errors detection in NL2SQL translation (Section 2.1).

(2) NL2SQL Semantic Errors Taxonomy. We introduce a comprehensive two-level taxonomy to classify NL2SQL semantic errors into 9 main categories and 31 subcategories (Section 3).

(3) New Benchmark. We present NL2SQL-BUGs, *the first* benchmark designed for evaluating the capabilities of NL2SQL semantic errors detection. The dataset consists of 2,018 expert-annotated instances, each with a natural language query, database schema, and SQL query, along with detailed error annotations for semantically incorrect SQL queries (Section 4).

(4) Finding Errors in Existing NL2SQL Benchmarks. We leverage the insights from our error taxonomy and demonstration cases in NL2SQL-BUGs to prompt GPT-4o to detect semantic errors in widely-used NL2SQL benchmarks – Spider and BIRD. Our evaluation revealed that 16 SQL queries in Spider (**1.55%** of the dev set) and **106** SQL queries in BIRD (**6.91%** of the dev set) contained semantic errors that had not been previously identified (Section 2.2).

(5) Extensive Experiments. We evaluate various LLM-based solutions on the NL2SQL-BUGs benchmark to assess their capabilities in detecting NL2SQL semantic errors. While some methods perform reasonably well on specific error types (e.g., attribute-related or table-related), overall performance remains limited. These findings highlight the need for further research into more robust semantic error detection methods (Section 5).

We contend that NL2SQL-BUGs will play an important role in grounding NL2SQL techniques for practical applications. We call for cross-community collaboration: database engineers, data mining practitioners, and NLP researchers, to integrate formal logic into training paradigms, fostering ethical, deployable NL2SQL systems.

2 Problem and Real-world Cases

2.1 Problem Formulation

NL2SQL Semantic Error Detection. Given a triple (NL, DB, SQL) consisting of a natural language question (NL), a relational database (DB), and an SQL query (SQL) generated by models or provided by humans, the task of semantic error detection aims to verify whether SQL is semantically equivalent to the NL over DB. There are two types of semantic errors: (1) *Incorrect results*: The query returns results that do not match the expected intent of the natural language query. (2) *Correct results on a specific instance*: The query may produce correct results, but only for the current database instance; the results may not hold for all possible database instances.

The Taxonomy of NL2SQL Translation Semantic Errors

Category	Error Type	Description
Attribute-related Errors (§3.2.1)	Attribute Mismatch	The attribute [A] may be wrong.
	Attribute Redundancy	The attribute [A] may not be mentioned in the NL.
	Attribute Missing	The attribute [A] may be missing.
Table-related Errors (§3.2.2)	Table Mismatch	The table [T] may be wrong.
	Table Redundancy	The table [T] may be unnecessary.
	Table Missing	The table [T] may be missing.
	Join Condition Mismatch	The join condition between table [T] and table [T] is incorrect.
	Join Type Mismatch	The join type [K] (e.g., LEFT JOIN) is inconsistent with the NL.
Value-related Errors (§3.2.3)	Value Mismatch	The value [V] in condition [C] may be wrong.
	Data Format Mismatch	The data format of value [V] in attribute [A] may be wrong.
Operator-related Errors (§3.2.4)	Comparison Operator	The comparison operator [O] in condition [C] may be wrong.
	Logical Operator	The boolean operator [O] or the logical operator precedence may be wrong.
Condition-related Errors (§3.2.5)	Explicit Condition Missing	The condition [C] in NL may be missing.
	Explicit Condition Mismatch	The condition [C] may be wrong.
	Explicit Condition Redundancy	The condition [C] which not mentioned in NL.
	Implicit Condition Missing	The SQL fails to include implicit conditions [C] (e.g., IS NOT NULL).
Function-related Errors (§3.2.6)	Aggregate Functions	The usage of aggregate functions [F] (e.g., SUM, AVG) is incorrect.
	Window Functions	The usage of window functions [F] (e.g., OVER, PARTITION BY) is incorrect.
	Date/Time Functions	The usage of date/time functions [F] (e.g., JULIANDAY, strftime) is incorrect.
	Conversion Functions	The usage of conversion functions [F] (e.g., CAST) is incorrect.
	Math Functions	The usage of math functions [F] (e.g., ROUND) is incorrect.
	String Functions	The usage of string functions [F] (e.g., SUBSTR) is incorrect.
	Conditional Functions	The usage of conditional functions [F] (e.g., IIF, CASE WHEN) is incorrect.
Clause-related Errors (§3.2.7)	Clause Missing	The clause [K] (e.g., GROUP BY) is missing.
	Clause Redundancy	The clause [K] (e.g., GROUP BY) is redundancy.
Subquery-related Errors (§3.2.8)	Subquery Missing	The subquery [Q] is missing.
	Subquery Mismatch	The subquery [Q] is mismatch with the logic with NL.
	Partial Query	The query [Q] is a partial query that contributes to the complete SQL.
Other Errors (§3.2.9)	ASC/DESC	The usage of ASC/DESC is incorrect.
	DISTINCT	The usage of DISTINCT is either omitted or incorrectly applied.
	Other	The SQL generated by the model almost necessitates a complete rewrite.

Figure 3: A Taxonomy of NL2SQL Translation Semantic Errors.

Formally, the task can be defined as a function $\mathcal{F}$ that maps the triple to a binary decision: $\mathcal{F}(\text{NL}, \text{DB}, \text{SQL}) \rightarrow \{True, Fasle\}$, where $True$ (resp. $False$) indicates that SQL is semantically correct (resp. wrong) w.r.t. the given question NL.

If a semantic error is detected, *i.e.*, $\mathcal{F}(\text{NL}, \text{DB}, \text{SQL}) = False$, the next task is to classify the type of semantic error. This can be formally defined as follows:

NL2SQL Semantic Error Type Detection. Given a triple (NL, DB, SQL) consisting of a natural language question (NL), a relational database (DB), and a generated SQL query (SQL), semantic error classification aims to identify the specific category of semantic error when SQL fails to capture NL's intent over DB: $\mathcal{F}(\text{NL}, \text{DB}, \text{SQL}) \rightarrow T$, where T is the predefined set of semantic error types, $\mathcal{F}(\text{NL}, \text{DB}, \text{SQL}) = t \in T$ represents the specific type of semantic error identified in the SQL query.

2.2 Detected Errors in Popular Benchmarks

To demonstrate the utility of the semantic error detection task, we applied GPT-4o [7] to detect semantic errors in two widely used benchmarks: Spider [45] and BIRD [15]. For each dataset, we used natural language queries (NL), database schemas (DB), and corresponding SQL queries (SQL) as input to GPT-4o [7]. The model then determined whether each SQL query was semantically correct with respect to the natural language query and the database schema.

After the automated semantic error detection, we manually validated the LLM's predictions to ensure accuracy. Our validation revealed that 16 SQL queries in Spider (1.55% of the development set) and **106** SQL queries in BIRD (**6.91%** of the development set) *contained semantic errors* that had not been previously identified. The real errors identified in both the Spider and BIRD datasets are provided in Appendix A.

These findings demonstrate the utility of the semantic error detection model and highlight the importance of devising semantic error detection models to find hidden errors in NL2SQL datasets.

3 NL2SQL Semantic Errors Taxonomy

3.1 Overview

The task of detecting semantic errors in NL2SQL translation requires a clear and structured understanding of the types of errors that can arise. Based on the extensive analysis of NL2SQL systems [13, 14, 29, 34, 35], we propose a comprehensive two-level taxonomy to categorize semantic errors in NL2SQL translation.

The classification of semantic errors in NL2SQL is based on the structure of SQL queries, common translation mistakes, and their impact on query semantics. This approach allows for systematic error identification at various stages of query generation, helping to pinpoint where and why translation mistakes occur. Therefore, as shown in Figure 3, we classify semantic errors into 9 main categories such as `Attribute-related Errors`. Each category contains multiple subcategories that identify specific types of errors, allowing for a detailed understanding of how and where translation mistakes occur. For example, errors in mapping attributes (columns) from the natural language query to the corresponding attributes in the database are classified under `Attribute-related Errors`.

3.2 Two-Level Taxonomy

3.2.1 Attribute-related Errors. Attribute-related Errors occur when NL2SQL models incorrectly map the required columns in natural language queries to corresponding attributes in the database schema. These errors manifest in three primary forms:

Attribute Mismatch: The model selects incorrect attributes from the schema, indicating a misunderstanding between natural language expressions and their corresponding database fields.

Attribute Redundancy: The model includes unnecessary attributes not mentioned or implied in the natural language query, suggesting over-interpretation of the input.

Attribute Missing: The model fails to identify and include attributes that are essential to fulfilling the query intent, indicating an incomplete understanding of the natural language requirements.

These errors primarily stem from the semantic gap between natural language expressions and database schema representations, where models struggle to establish accurate mappings between user intent and the formal database structure.

3.2.2 Table-related Errors. Table-related Errors occur when NL2SQL models fail to correctly identify, link, or utilize the required tables from the database schema. These errors encompass both basic table selection issues and more complex `JOIN` operations, manifesting in five distinct forms:

Table Mismatch: The model selects incorrect tables from the schema, indicating a misalignment between natural language query intent and table identification.

Table Redundancy: The model includes unnecessary tables not required by the query intent, which occurs particularly frequently and suggests over-complication of the query structure.

Table Missing: The model fails to include tables that are required, leading to an incomplete understanding of the schema.

Join Condition Mismatch: The model exhibits errors in constructing `JOIN` conditions between tables, creating invalid `JOIN` between unrelated attributes, which demonstrates difficulties in understanding the relationships between tables in the database.

Join Type Mismatch: The model selects inappropriate `JOIN` types (e.g., `LEFT JOIN`, `INNER JOIN`) that do not align with the natural language query intent, showing a misinterpretation of the desired data inclusion/exclusion patterns.

Table-Related Example

Query: List all students and their course grades, including students who haven't taken any courses.

Incorrect SQL:	**Correct SQL:**
SELECT s.name, e.grade	SELECT s.name, e.grade
FROM student s	FROM student s
INNER JOIN enrollment e	LEFT JOIN enrollment e
ON `s.id = e.id`	ON `s.id = e.student_id`

The example demonstrates both `JOIN` Type Mismatch and `JOIN` Condition Mismatch errors: Using `INNER JOIN` excludes students without courses, contrary to the requirement. The connection condition between the student table and the enrollment table is wrong (*i.e.,* `student.id → enrollment.student_id`).

3.2.3 Value-related Errors. Value-related Errors arise when NL2SQL models incorrectly parse or interpret the required attribute values from natural language expressions. These errors manifest in two primary forms:

Value Mismatch: The model fails to map values from natural language descriptions to their actual representations in the database, occurring frequently and reflecting a misunderstanding of how values are actually stored in the database.

Data Format Mismatch: The model fails to properly format values according to the attribute's data type requirements, showing difficulties in handling type-specific representations.

Please refer to Appendix B.1 for an example.

3.2.4 Operator-related Errors. Operator-related Errors occur when NL2SQL models fail to select appropriate operators for SQL conditions. These errors manifest in two primary forms:

Comparison Operator: The model selects incorrect comparison operators in query conditions, such as using '>' when '>=' is required or confusing 'LIKE' with '=', indicating a semantic gap between natural language intent and SQL comparison operations.

Logical Operator: The model misinterprets logical relationships expressed in natural language, resulting in the incorrect use of boolean operators (e.g., `AND`, `OR`, `NOT`) in SQL conditions or the logical operator precedence may be wrong.

Operator-Related Example

Query: Find all courses that started after January 1st, 2023 and have more than 30 students.

Incorrect SQL:	**Correct SQL:**
SELECT Student_ID	SELECT Student_ID
FROM Students	FROM Students
WHERE Grade = 3 AND	WHERE Grade = 3 AND
Math_Score > 90 OR	(Math_Score > 90 OR
English_Score > 90	English_Score > 90)

This example demonstrates Logical Operator errors: By default, the `AND` operator has higher precedence than the `OR` operator. So,

the incorrect query is interpreted by the database as: WHERE (Grade = 3 AND Math_Score > 90) OR English_Score > 90.

3.2.5 Condition-related Errors. Condition-related Errors occur when NL2SQL models fail to properly handle query conditions, particularly in cases where multiple condition-related issues co-exist. These errors are categorized into explicit and implicit conditions:

Explicit Condition Missing: The model fails to include conditions that are explicitly stated in the natural language query, indicating an incomplete interpretation of the user requirements.

Explicit Condition Mismatch: The model generates incorrect conditions that deviate from the natural language requirements, showing a misunderstanding of the query intent. A condition typically consists of three parts (attribute, operator, and value), and we classify it as a condition error only when errors occur in two or more components.

Explicit Condition Redundancy: The model adds unnecessary conditions not mentioned in the natural language query, suggesting an over-interpretation of the query requirements.

Implicit Condition Missing: The model fails to include necessary conditions that are implied but not explicitly stated in the natural language query, such as NULL value handling.

Please refer to Appendix B.2 for an example.

3.2.6 Function-related Errors. Function-related Errors occur when NL2SQL models fail to correctly use SQL functions or misunderstand their purposes. We consider the following cases.

Aggregate Functions: The model incorrectly applies aggregate functions (*e.g.,* SUM), such as using them without proper GROUP BY clauses or misunderstanding their effect on result sets.

Window Functions: The model misuses window functions (*e.g.,* OVER, PARTITION BY), often failing to properly specify window frames or partition criteria.

Date/Time Functions: The model incorrectly handles date and time manipulations (*e.g.,* STRFTIME), such as wrong format strings or improper date arithmetic.

Conversion Functions: The model fails to properly convert between data types (*e.g.,* CAST, CONVERT), leading to type mismatch errors or incorrect data transformations.

Math Functions: The model misapplies mathematical functions (*e.g.,* ROUND, CEIL, FLOOR), such as incorrect precision specifications or inappropriate numerical operations.

String Functions: The model incorrectly uses string manipulation functions (*e.g.,* SUBSTR, CONCAT), such as wrong substring positions or improper string concatenations.

Conditional Functions: The model misuses conditional functions (*e.g.,* IIF, CASE WHEN), such as incorrect condition logic or improper result expressions.

Please refer to Appendix B.3 for an example.

3.2.7 Clause-related Errors. Clause-related Errors occur when NL2SQL models fail to correctly include, omit, or structure SQL clauses. These errors primarily involve mishandling GROUP BY, ORDER BY, and HAVING clauses, which are crucial for aggregation, sorting, and filtering grouped results. The errors manifest in two main categories:

Clause Missing: The model fails to include necessary clauses such as GROUP BY, ORDER BY, or HAVING, resulting in incomplete or incorrect SQL queries that do not fully capture the intent of the natural language.

Clause Redundancy: The model includes extraneous clauses like GROUP BY, ORDER BY, or HAVING that are not required by the natural language query, indicating an over-interpretation or misunderstanding of the input requirements.

3.2.8 Subquery-related Errors. Subquery-related Errors occur when NL2SQL models fail to correctly use or generate subqueries. These errors manifest in three distinct categories:

Subquery Missing: The model fails to generate a necessary subquery, leading to an incomplete or incorrect SQL statement.

Subquery Mismatch: The model generates a subquery that does not match the context or requirements of the outer query, resulting in logical errors or unexpected results.

Partial Query: The model generates an SQL query that is only a part of the complete query required to fully address the natural language request. The generated SQL needs to be wrapped inside another query or combined with additional SQL constructs to produce the desired output.

Subquery-Related Example

Query: Find the names of students who have enrolled in the maximum number of courses.

Incorrect SQL:	Correct SQL:
SELECT COUNT(*) as count, student_id	SELECT s.name FROM students s
FROM enrollments	JOIN (SELECT COUNT(*) as count, student_id
GROUP BY student_id	...
ORDER BY count DESC	LIMIT 1)
LIMIT 1	AS e ON s.id = e.student_id

This example demonstrates a "Partial Query" error, where the incorrect SQL query only returns the student_id and course count, but not the students' names. To retrieve the names, an external join with the students table is required, making this query a part of a complete query.

3.2.9 Other Errors. This category is mainly for errors caused by incorrect use of some special keywords in SQL and queries where the SQL intent seriously deviates from the original intent of NL.

ASC/DESC: This error occurs when the model improperly handles the ordering of query results by either failing to include the correct sorting order (ASC/DESC) or by incorrectly specifying the order.

DISTINCT: This error occurs in SQL queries, typically due to the failure to properly apply deduplication operations (e.g., missing DISTINCT), resulting in the presence of duplicate records or unnecessary deduplication, which impacts the accuracy and performance of the query results.

Others: The SQL generated by the model deviates significantly from the required SQL, almost necessitating a complete rewrite. It is difficult to achieve the expected results through corrective modifications. We categorize such errors under this type.

4 NL2SQL-BUGs Overview

Next, we first overview the characteristics of NL2SQL-BUGs (Section 4.1) and then introduce how NL2SQL-BUGs is curated based on real running examples generated by various NL2SQL models over popular NL2SQL benchmark – BIRD [15] (Section 4.2).
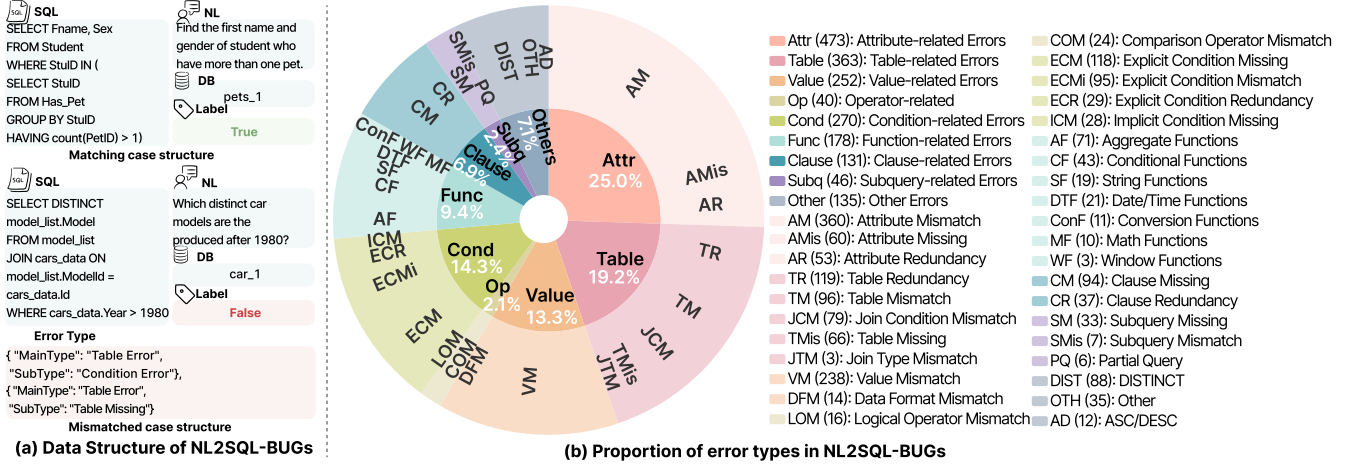
Figure 4: Data Structure and Proportion of Error Types in NL2SQL-BUGs Benchmark.

4.1 NL2SQL-BUGs Statistics

The NL2SQL-BUGs benchmark is designed to detect semantic errors in NL2SQL translations by distinguishing between correct and incorrect NL2SQL translations.

Examples of Errors. Figure 4(a) illustrates two examples in NL2SQL-BUGs that highlight different types of semantic errors in NL2SQL translation. The first example (top-left) represents a correctly generated SQL query, where the natural language query and the SQL query are semantically aligned, resulting in no errors. In NL2SQL-BUGs, there are a total of 1,019 correct examples, where the natural language query and the SQL query match semantically and produce correct results. The second example (bottom-left) highlights a mismatched case, where the SQL query fails to correctly represent the intent of the natural language query, leading to a semantic error. This example is classified as a "Table Error" and "Condition Error", demonstrating how mismatched table structures or incorrect conditions in the query can produce incorrect results, even when the SQL syntax itself is valid. In NL2SQL-BUGs, there are 999 incorrect examples (semantic errors), where the SQL queries do not match the intent of the natural language queries, and the discrepancies are categorized into different error types.

Error Type Distribution. Figure 4(b) shows the distribution of error types within the incorrect translation examples (*i.e.,* the NL2SQL semantic errors) in the NL2SQL-BUGs benchmark. The errors are classified into nine main categories, each capturing a different aspect of semantic mistakes in NL2SQL translation.

The distribution of errors provides several important insights. First, attribute-related and table-related errors are the most frequent, highlighting the difficulty models face in correctly identifying and mapping database schema components from natural language queries. Second, condition-related errors and value-related errors also appear frequently, suggesting that correctly interpreting query conditions and handling values are key areas of challenge. Third, although function-related and clause-related errors occur

less frequently, their impact on NL2SQL translation quality is significant, as even a few mistakes in these complex SQL constructs can lead to major semantic discrepancies.

4.2 NL2SQL-BUGs Construction

In this section, we describe how the NL2SQL-Bugs dataset from BIRD is curated to support the detection and classification of semantic errors in NL2SQL translations.

Dataset Components. Each instance in NL2SQL-BUGs contains the following components: (1) *Natural Language Query* (NL): Expressing various database-related user questions in natural language; (2) *Database* (DB): The database related to the user question; (3) *SQL Query* (SQL): Generated translations of NL, including both correct and incorrect queries; (4) *Correctness Label*: True or False labels indicating whether the SQL query is semantically correct; (5) *Error Types*: For incorrect SQL queries, detailed error categorizations are provided to facilitate error detection research.

Obtaining Reliable Correctness Labels. To establish a robust evaluation benchmark for NL2SQL models, obtaining clean correct labels is crucial. The labeling process primarily focuses on execution result matching - whether the SQL query returns identical results as the gold SQL query. For instance in Figure 4(a), consider an SQL query "SELECT Fname, Sex FROM Student WHERE StuID IN (SELECT StuID FROM Has_Pet GROUP BY StuID HAVING COUNT(PetID) > 1)" paired with the question "Find the first name and gender of students who have more than one pet." This pair is annotated with a True label since its execution results exactly match the gold SQL query. In contrast, for the question "Which distinct car models are produced after 1980?" paired with "SELECT DISTINCT model_list.Model FROM model_list JOIN cars_data ON model_list.ModelId = cars_data.Id WHERE cars_data.Year > 1980", the label would be False as its result set deviates from the gold SQL query due to the incorrect join condition. Through this result-based validation approach, we can establish reliable correctness labels for evaluating NL2SQL models.
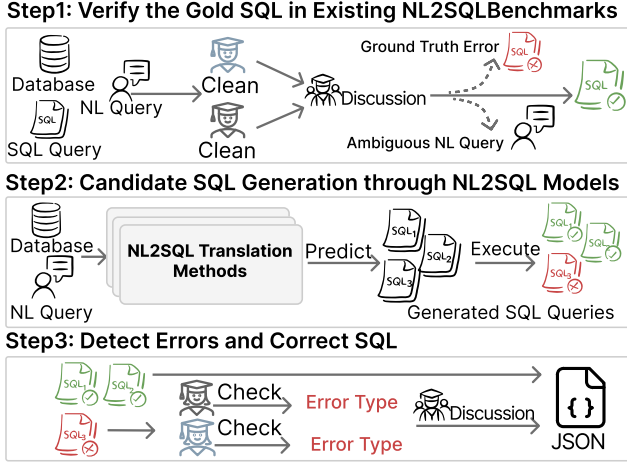
Step1: Verify the Gold SQL in Existing NL2SQLBenchmarks

Step2: Candidate SQL Generation through NL2SQL Models

Step3: Detect Errors and Correct SQL

Figure 5: The Construction Pipeline of NL2SQL-BUGs.

Determining Error Types. To annotate error types for incorrect SQL queries, we employ a two-step approach. First, we examine whether the SQL query executes successfully in the database. If it fails to execute, it is classified as a syntax error. Second, for executable queries that produce incorrect results, we proceed with semantic error classification. Errors are hierarchically classified into main types and subtypes, as detailed in Figure 3. For example, under the main type "Table-related Error", we identify subtypes such as "Join Condition Mismatch" (when join conditions do not align with the query intent) and "Table Missing" (when essential tables are omitted from the query). This hierarchical classification allows for more precise identification of model weaknesses and guides targeted improvements for error correction mechanisms.

To ensure SQL correctness and accurate error annotation, we construct NL2SQL-BUGs through three steps, as shown in Figure 5.

Step 1: Verify the Gold SQL in Existing NL2SQL Benchmarks. As discussed in Section 2.2, existing NL2SQL benchmarks may contain semantically incorrect SQL queries, which can compromise the effectiveness of model evaluation. Therefore, the first step in curating NL2SQL-BUGs is to verify the ground truth SQL queries from existing benchmarks and correct any errors.

We began by collecting test data from the BIRD [15] development dataset, which includes queries of varying difficulty, such as those involving nested queries, aggregate functions, and subqueries. During our inspection, we found that some SQL queries were semantically incorrect, and some NL queries lacked a clear corresponding SQL query. To address these issues, we conducted a secondary manual cleaning process.

Specifically, two PhD students were tasked with verifying each SQL query in the dataset, ensuring that they were consistent with both the database schema and the corresponding NL. After their individual assessments, they identified errors in SQL queries as well as ambiguous NL queries. These cases were then discussed collaboratively until a consensus was reached on the necessary modifications. The finalized erroneous queries were removed, ensuring a high-quality ground truth dataset.

Table 1: Semantic Error Detection of Different LLMs (NP: Negative Precision; NR: Negative Recall; PP: Positive Precision; Positive Recall: PR)

Models	Semantic Error Detection				
	Accuracy	**NP**	**NR**	**PP**	**PR**
GPT-4o-mini	71.56	67.43	82.28	77.85	61.04
GPT-4o	**76.81**	72.63	85.29	82.6	68.5
Claude-3.5-Sonnet	76.36	71.75	86.19	83.13	66.73
Gemini-2.0-Flash	**76.81**	**82.66**	67.27	72.86	**86.16**
DeepSeek-V3	75.02	70.44	85.39	81.91	64.87
Qwen2.5-72B	74.38	69.28	**86.69**	**82.68**	62.32

Step 2: Candidate SQL Generation through NL2SQL Models. After cleaning the dataset, we used multiple NL2SQL models from the BIRD [15] leaderboard to generate SQL queries. The models used included RESDSQL [13] (a pre-trained language model), CodeS [14] (a fine-tuned large language model), and CHESS [35] (an LLM agent with GPT-4o). These models generated SQL queries based on the database schema and the corresponding NL query.

Since the first step ensured the correctness of the ground truth data, we directly compared the execution results of the model-generated SQL with the ground truth to determine correctness. In cases where identical SQL queries were generated by multiple models, only one instance was retained to avoid redundant analysis.

Step 3: Detect Errors and Correct SQL. In the final step, we categorized and annotated the erroneous SQL queries. The annotation labels consisted of 9 major categories and 31 subcategories (see Figure 3 for details). During the annotation process, we provided each database's ER diagram, requiring the PhD students to first familiarize themselves with it before annotating the SQL errors. Two PhD students independently annotated the errors, and their annotations were aggregated. The Cohen's Kappa coefficient reached 0.78, indicating a high level of agreement between the annotators. In cases of disagreement about specific error types, discussions were held, and a third SQL expert was consulted to resolve the differences until a consensus was reached. Both correct samples and incorrect samples with corresponding error types were added to the dataset.

Finally, NL2SQL-BUGs contains 1,019 correct examples and 999 incorrect examples with semantic errors.

5 Experiment

In this section, we evaluate the capabilities of NL2SQL semantic error detection. Specifically, we try to answer the following questions: **Q1:** How effective are different LLMs in detecting NL2SQL semantic errors? **Q2:** What are the strengths and weaknesses of various LLMs in detecting different types of semantic errors?

5.1 Experimental Settings

Models. We consider state-of-the-art LLMs, including open-source models such as Qwen2.5-72B [31], DeepSeek-V3 [17], as well as closed-source models like GPT-4o [7], GPT-4o-mini [27], Claude-3.5-Sonnet [1], and Gemini-2.0-Flash [4]. For all models, we set the temperature to 0 to ensure deterministic output.

Table 2: The Comparison of SQL Error Detection Abilities Across Different Models for Different Error Categories (TSA_t).

Error Types	Sub Error Types	Qwen2.5-72B	GPT-4o	GPT-4o-mini	Claude-3.5-Sonnet	Deepseek-V3	Gemini-2.0-Flash
Attribute-related Errors	Attribute Mismatch	26.94%	32.78%	18.61%	28.61%	24.17%	20.83%
	Attribute Redundancy	5.66%	13.21%	9.43%	7.55%	1.89%	0.00%
	Attribute Missing	40.00%	36.67%	30.00%	35.00%	43.33%	28.33%
Table-related Errors	Table Mismatch	6.25%	14.58%	18.75%	16.67%	11.46%	11.46%
	Table Redundancy	36.13%	36.97%	14.29%	37.82%	35.29%	28.57%
	Table Missing	50.00%	46.97%	22.73%	40.91%	48.48%	36.36%
	Join Type Mismatch	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	Join Condition Mismatch	24.05%	58.23%	46.84%	48.10%	46.84%	34.18%
Value-related Errors	Value Mismatch	44.12%	48.74%	40.76%	40.76%	42.44%	34.45%
	Data Format Mismatch	14.29%	21.43%	14.29%	21.43%	14.29%	7.14%
Operator-related Errors	Comparison Operator Mismatch	8.33%	25.00%	8.33%	25.00%	0.00%	4.17%
	Logical Operator Mismatch	43.75%	43.75%	18.75%	50.00%	31.25%	25.00%
Condition-related Errors	Explicit Condition Missing	62.71%	66.95%	69.49%	71.19%	57.63%	35.59%
	Explicit Condition Mismatch	29.47%	37.89%	45.26%	35.79%	33.68%	24.21%
	Explicit Condition Redundancy	27.59%	31.03%	10.34%	10.34%	10.34%	10.34%
	Implicit Condition Missing	0.00%	14.29%	3.57%	0.00%	14.29%	3.57%
Function-related Errors	Aggregate Functions	28.17%	35.21%	15.49%	36.62%	30.99%	18.31%
	Window Functions	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	Date/Time Functions	14.29%	9.52%	4.76%	14.29%	28.57%	9.52%
	Conversion Functions	0.00%	0.00%	0.00%	0.00%	9.09%	0.00%
	Math Functions	30.00%	40.00%	0.00%	30.00%	0.00%	0.00%
	String Functions	5.26%	5.26%	0.00%	0.00%	10.53%	15.79%
	Conditional Functions	6.98%	0.00%	0.00%	2.33%	2.33%	0.00%
Clause-related Errors	Clause Missing	26.60%	19.15%	22.34%	18.09%	25.53%	17.02%
	Clause Redundancy	24.32%	8.11%	13.51%	5.41%	16.22%	13.51%
Subquery-related Errors	Subquery Missing	9.09%	12.12%	0.00%	3.03%	6.06%	9.09%
	Subquery Mismatch	42.86%	14.29%	28.57%	28.57%	28.57%	14.29%
	Partial Query	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Other Errors	ASC/DESC	58.33%	83.33%	58.33%	66.67%	58.33%	66.67%
	DISTINCT	0.00%	6.82%	1.14%	4.55%	0.00%	0.00%
	Other	0.00%	2.86%	0.00%	8.57%	0.00%	8.57%

Datasets. We use our curated benchmark NL2SQL-BUGs to evaluate LLMs' ability to detect NL2SQL semantic errors. It includes 1,019 correct cases and 999 cases with semantic errors.

5.2 Experiment for Q1

The NL2SQL Semantic Error Detection task is formulated as a binary classification task. To evaluate the capability of LLMs in this task, we adopt the classic metrics for classification tasks.

Evaluation Metrics. We evaluate performance using Overall Accuracy, along with a refined breakdown of Precision and Recall. Precision measures the proportion of correctly predicted True cases (correct SQLs) out of all cases predicted as True, while Recall assesses the proportion of correctly predicted True cases out of all actual True cases. For the Negative case, the metrics are defined similarly, focusing on correctly identifying incorrect queries.

Overall Semantic Error Detection Capability. In Table 1, the performance differences between the models in the NL2SQL semantic error detection task are not highly pronounced. GPT-4o and Claude-3.5-Sonnet are the most balanced models, performing well across various types of semantic errors and outperforming the other models. Despite achieving a solid overall accuracy of 76.81%, Gemini-2.0-Flash exhibits imbalanced performance across metrics, as evidenced by its higher positive recall (86.16%) compared to negative recall (67.27%), suggesting a greater sensitivity to positives but also a higher risk of false negatives. DeepSeek-V3 and Qwen2.5-72B still have room for improvement, particularly in enhancing their positive recall and negative precision.

5.3 Experiment for Q2

The NL2SQL Semantic Error Type Detection focuses on identifying the specific type of error present in an incorrect SQL query.

Evaluation Metrics. We report a Type-Specific Accuracy (TSA_t) for each error type t. Given a dataset D of instances and a set T of all error types, we denote the set of true error types present in an instance $i \in D$ as $E_i \subseteq T$, and the set of error types predicted by the model as $P_i \subseteq T$. The Type-Specific Accuracy (TSA_t) for a given error type $t \in T$ is then defined as:

$$TSA_t = \frac{|\{i \in D \mid t \in E_i \land t \in P_i\}|}{|\{i \in D \mid t \in E_i\}|}$$

We calculated the metrics TSA_t for specific categories, as shown in Table 2, which shows the performance of different models across subtypes. The radar chart (see Figure 6) further demonstrates the performance differences of the models at the main category level.

Overall Semantic Type Error Detection Capability. Despite achieving relatively high accuracy (around 75%) in distinguishing correct and incorrect SQL queries(Table 1), the models fail to reach 50% accuracy in identifying specific error types (Figure 6). This discrepancy suggests that while the models may correctly flag a query as erroneous, they often struggle to pinpoint the underlying cause or fail to recognize all existing errors.

The empirical evidence presented in Figure 6 demonstrates that the models exhibit pronounced proficiency in identifying Condition-related errors and Value errors. Conversely, the models display
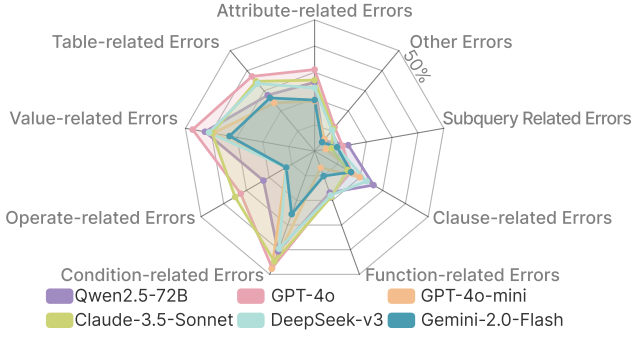
Figure 6: Performance of NL2SQL models in error type detection across main error types (TSA_t).

markedly diminished capability in detecting subquery-related errors and those categorized as "Other Errors", indicating a significant performance differential across error typologies.

In comparative analysis, GPT-4o and Gemini-2.0-Flash demonstrate identical accuracy metrics (76.81% in Table 1); however, regarding error categorization capabilities, GPT-4o exhibits substantially superior performance compared to Gemini-2.0-Flash (Figure 6). This observation illuminates a critical distinction in language model evaluation methodologies: equivalent accuracy metrics do not necessarily reflect equivalent capabilities. GPT-4o demonstrates a deeper understanding of SQL semantics and structure, enabling it not only to detect anomalies (Error Detection), but also to analytically identify the underlying error taxonomies (Error Type Detection).

Fine-grained Evaluation of Semantic Error Detection Capability. We conducted a fine-grained evaluation of semantic error detection capabilities, with results summarized in Table 2. The model exhibits deficiencies in fine-grained detection capabilities. For example, Claude-3.5-Sonnet outperforms GPT-4o-mini in many subtypes of error detection, yet underperforms in the detection of Explicit Condition Mismatch, highlighting certain limitations of the model. It demonstrates greater proficiency in handling tasks involving substantial semantic variations. However, its performance is suboptimal, particularly in tasks such as Explicit Condition Missing, which require less complex semantic processing. Moreover, the model struggles with errors that necessitate DB knowledge, such as Join Type Mismatch and Function-Related Errors, revealing considerable potential for improvement.

6 Related Work

NL2SQL Translation Methods. NL2SQL translation has been an active research area in recent years. Early research relied mainly on rule-based templates and semantic parsing approaches [8, 11], which often struggled with complex queries. With the advancement of deep learning, neural network-based methods [2, 16, 40] substantially improved translation accuracy. The introduction of pre-trained language models made a particular breakthrough [13]. More recently, Large Language Models (LLMs), which have advanced many fields [18, 37–39, 44, 48, 49], have made significant strides in NL2SQL [9, 19, 51], pushing accuracy close to 90% on

Spider while achieving 70% on the more challenging BIRD benchmark [15]. While these advancements are impressive, the 30% error rate in complex scenarios in BIRD highlights significant challenges in real-world deployments, where near-perfect accuracy is crucial. For instance, although models such as Alpha-SQL [10] and CHASE-SQL [28] utilize LLMs to achieve notable performance in the NL2SQL domain, they are still prone to significant issues with hallucination and errors. In critical fields like finance and healthcare, even small query errors can have serious consequences. Despite this, current research mainly focuses on improving overall accuracy, neglecting systematic error analysis and correction.

NL2SQL Errors Detection. NL2SQL faces challenges in understanding user intent, handling complex database schemas, and generating intricate SQL [19]. Prior approaches to error analysis in NL2SQL systems can be broadly categorized into three main types: NL explanation, human-machine collaboration, and visual analytics. DIY [26] employs templates to provide step-by-step NL explanations of SQL, helping users to comprehend them incrementally. NaLIR [12], on the other hand, enables SQL explanation and error detection by creating entity mappings between NL and SQL. MISP [43] adopts a human-machine collaborative approach, leveraging parsed states and incorporating human feedback for SQL error detection. SQLVis [25] applies visualization techniques to SQL error detection by transforming SQL into graphical representations, helping users understand complex SQL and detect errors. Prior research has faced challenges in reliable semantic error detection and explanation of SQL queries, often relying heavily on manual effort. There has been a preliminary exploration of automated error detection systems [42]. To facilitate this research direction, we propose a two-level taxonomy with 9 main categories and 31 subcategories to categorize semantic errors in NL2SQL translation. Based on this, we also developed the first benchmark for evaluating NL2SQL semantic error detection, containing 2,018 expert-annotated examples.

7 Conclusion

In this paper, we introduced the task of semantic error detection in NL2SQL translation and demonstrated its importance by uncovering 106 errors (6.91%) in the BIRD benchmark and 16 errors (1.55%) in the Spider benchmark. We proposed a two-level taxonomy categorizing semantic errors into 9 main categories and 31 subcategories. Based on this taxonomy, we developed NL2SQL-BUGs, the first benchmark designed specifically for evaluating NL2SQL semantic error detection, containing 2,018 expert-annotated examples. Through extensive experiments, we highlighted the limitations of current LLM-based models for detecting NL2SQL semantic errors. This study aims to advance the development of techniques for the automatic detection of semantic errors.

Acknowledgments

References

[1] Anthropic. 2024. Claude 3.5 sonnet. https://www.anthropic.com/news/claude-3-5-sonnet

[2] Ben Bogin, Matt Gardner, and Jonathan Berant. 2019. Representing schema structure with graph neural networks for text-to-SQL parsing. arXiv preprint arXiv:1905.06241 (2019).

[3] Yeounoh Chung, Gaurav T. Kakkar, Yu Gan, Brenton Milne, and Fatma Ozcan. 2025. Is Long Context All You Need? Leveraging LLM's Extended Context for NL2SQL. arXiv:2501.12372 [cs.DB] https://arxiv.org/abs/2501.12372

[4] Google Deepmind. 2025. Gemini 2.0 Flash Thinking Mode. https://ai.google.dev/gemini-api/docs/thinking-mode

[5] Jonathan Fürst, Catherine Kosten, Farhad Nooralahzadeh, Yi Zhang, and Kurt Stockinger. 2024. Evaluating the Data Model Robustness of Text-to-SQL Systems Based on Real User Queries. arXiv:2402.08349 [cs.DB] https://arxiv.org/abs/2402.08349

[6] Zezhou Huang, Pavan Kalyan Damalapati, and Eugene Wu. 2023. Data Ambiguity Strikes Back: How Documentation Improves GPT's Text-to-SQL. arXiv:2310.18742 [cs.DB] https://arxiv.org/abs/2310.18742

[7] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. arXiv preprint arXiv:2410.21276 (2024).

[8] George Katsogiannis-Meimarakis and Georgia Koutrika. 2021. A deep dive into deep learning approaches for text-to-sql systems. In Proceedings of the 2021 International Conference on Management of Data. 2846–2851.

[9] Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024. The Dawn of Natural Language to SQL: Are We Fully Ready? Proc. VLDB Endow. 17, 11 (aug 2024), 3318–3331. https://doi.org/10.14778/3681954.3682003

[10] Boyan Li, Jiayi Zhang, Ju Fan, Yanwei Xu, Chong Chen, Nan Tang, and Yuyu Luo. 2025. Alpha-SQL: Zero-Shot Text-to-SQL using Monte Carlo Tree Search. arXiv:2502.17248 [cs.DB] https://arxiv.org/abs/2502.17248

[11] Fei Li and Hosagrahar V Jagadish. 2014. NaLIR: an interactive natural language interface for querying relational databases. In ACM SIGMOD.

[12] Fei Li and H. V. Jagadish. 2014. NaLIR: an interactive natural language interface for querying relational databases. In SIGMOD Conference. ACM, 709–712.

[13] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023. RESDSQL: Decoupling Schema Linking and Skeleton Parsing for Text-to-SQL. arXiv:2302.05965 [cs.CL]

[14] Haoyang Li, Jing Zhang, Hanbing Liu, and et al. 2024. Codes: Towards building open-source language models for text-to-sql. SIGMOD (2024).

[15] 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. 2023. 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 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.). http://papers.nips.cc/paper_files/paper/2023/hash/83fc8fab1710363050bbd1d4b8cc0021-Abstract-Datasets_and_Benchmarks.html

[16] Kevin Lin, Ben Bogin, Mark Neumann, Jonathan Berant, and Matt Gardner. 2019. Grammar-based neural text-to-sql generation. arXiv preprint arXiv:1905.13326 (2019).

[17] 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).

[18] Bang Liu, Xinfeng Li, Jiayi Zhang, Jinlin Wang, Tanjin He, Sirui Hong, Hongzhang Liu, Shaokun Zhang, Kaitao Song, Kunlun Zhu, Yuheng Cheng, Suyuchen Wang, Xiaoqiang Wang, Yuyu Luo, Haibo Jin, Peiyan Zhang, Ollie Liu, Jiaqi Chen, Huan Zhang, Zhaoyang Yu, Haochen Shi, Boyan Li, Dekun Wu, Fengwei Teng, Xiaojun Jia, Jiawei Xu, Jinyu Xiang, Yizhang Lin, Tianming Liu, Tongliang Liu, Yu Su, Huan Sun, Glen Berseth, Jianyun Nie, Ian Foster, Logan T. Ward, Qingyun Wu, Yu Gu, Mingchen Zhuge, Xiangru Tang, Haohan Wang, Jiaxuan You, Chi Wang, Jian Pei, Qiang Yang, Xiaoliang Qi, and Chenglin Wu. 2025. Advances and Challenges in Foundation Agents: From Brain-Inspired Intelligence to Evolutionary, Collaborative, and Safe Systems. CoRR abs/2504.01990 (2025).

[19] Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuyu Luo, Yuxin Zhang, Ju Fan, Guoliang Li, and Nan Tang. 2024. A Survey of NL2SQL with Large Language Models: Where are we, and where are we going? arXiv:2408.05109 [cs.DB] https://arxiv.org/abs/2408.05109

[20] Yuyu Luo, Xuedi Qin, Nan Tang, and Guoliang Li. 2018. DeepEye: Towards Automatic Data Visualization. In ICDE. IEEE Computer Society, 101–112.

[21] Yuyu Luo, Xuedi Qin, Nan Tang, Guoliang Li, and Xinran Wang. 2018. DeepEye: Creating Good Data Visualizations by Keyword Search. In SIGMOD Conference. ACM, 1733–1736.

[22] Yuyu Luo, Nan Tang, Guoliang Li, Chengliang Chai, Wenbo Li, and Xuedi Qin. 2021. Synthesizing Natural Language to Visualization (NL2VIS) Benchmarks from NL2SQL Benchmarks. In SIGMOD Conference. ACM, 1235–1247.

[23] Yuyu Luo, Nan Tang, Guoliang Li, Jiawei Tang, Chengliang Chai, and Xuedi Qin. 2022. Natural Language to Visualization by Neural Machine Translation. IEEE

[24] Peixian Ma, Boyan Li, Runzhi Jiang, Ju Fan, Nan Tang, and Yuyu Luo. 2024. A Plug-and-Play Natural Language Rewriter for Natural Language to SQL. CoRR abs/2412.17068 (2024). https://doi.org/10.48550/ARXIV.2412.17068 arXiv:2412.17068

[25] Daphne Miedema and George Fletcher. 2021. SQLVis: Visual Query Representations for Supporting SQL Learners. In IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2021, St Louis, MO, USA, October 10-13, 2021, Kyle J. Harms, Jácome Cunha, Steve Oney, and Caitlin Kelleher (Eds.). IEEE, 1–9. https://doi.org/10.1109/VL/HCC51201.2021.9576431

[26] Arpit Narechania, Adam Fourney, Bongshin Lee, and Gonzalo A. Ramos. 2021. DIY: Assessing the Correctness of Natural Language to SQL Systems. In IUI. ACM, 597–607.

[27] OpenAI. 2024. OpenAI o1-mini. https://openai.com/index/openai-o1-mini-advancing-cost-efficient-reasoning/

[28] Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O. Arik. 2024. CHASE-SQL: Multi-Path Reasoning and Preference Optimized Candidate Selection in Text-to-SQL. arXiv:2410.01943 [cs.LG] https://arxiv.org/abs/2410.01943

[29] Mohammadreza Pourreza and Davood Rafiei. 2024. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. NeurIPS (2024).

[30] Xuedi Qin, Yuyu Luo, Nan Tang, and Guoliang Li. 2020. Making data visualization more efficient and effective: a survey. VLDB J. 29, 1 (2020), 93–117.

[31] Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 Technical Report. arXiv:2412.15115 [cs.CL] https://arxiv.org/abs/2412.15115

[32] Chen Shen, Jin Wang, Sajjadur Rahman, and Eser Kandogan. 2024. Demonstration of a Multi-agent Framework for Text to SQL Applications with Large Language Models. In Proceedings of the 33rd ACM International Conference on Information and Knowledge Management, CIKM 2024, Boise, ID, USA, October 21-25, 2024, Edoardo Serra and Francesca Spezzano (Eds.). ACM, 5280–5283. https://doi.org/10.1145/3627673.3679216

[33] Sithursan Sivasubramaniam, Cedric Osei-Akoto, Yi Zhang, Kurt Stockinger, and Jonathan Fürst. 2024. SM3-Text-to-Query: Synthetic Multi-Model Medical Text-to-Query Benchmark. In Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024, Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (Eds.). http://papers.nips.cc/paper_files/paper/2024/hash/a182a8e6ebc91728b6e6b6382c9f7b1e-Abstract-Datasets_and_Benchmarks_Track.html

[34] Ruoxi Sun, Sercan O Arik, Hootan Nakhost, and et al. 2023. Sql-palm: Improved large language modeladaptation for text-to-sql. arXiv:2306.00739 (2023).

[35] Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, and et al. 2024. CHESS: Contextual Harnessing for Efficient SQL Synthesis. arXiv:2405.16755 (2024).

[36] Jiawei Tang, Yuyu Luo, Mourad Ouzzani, Guoliang Li, and Hongyang Chen. 2022. Sevi: Speech-to-Visualization through Neural Machine Translation. In SIGMOD Conference. ACM, 2353–2356.

[37] Nan Tang, Chenyu Yang, Ju Fan, Lei Cao, Yuyu Luo, and Alon Y. Halevy. 2024. VerifAI: Verified Generative AI. In CIDR. www.cidrdb.org.

[38] Nan Tang, Chenyu Yang, Zhengxuan Zhang, Yuyu Luo, Ju Fan, Lei Cao, Sam Madden, and Alon Y. Halevy. 2024. Symphony: Towards Trustworthy Question Answering and Verification using RAG over Multimodal Data Lakes. IEEE Data Eng. Bull. 48, 4 (2024), 135–146.

[39] Jinyu Xiang, Jiayi Zhang, Zhaoyang Yu, Fengwei Teng, Jinhao Tu, Xinbing Liang, Sirui Hong, Chenglin Wu, and Yuyu Luo. 2025. Self-Supervised Prompt Optimization. CoRR abs/2502.06855 (2025).

[40] Chunyang Xiao, Marc Dymetman, and Claire Gardent. 2016. Sequence-based structured prediction for semantic parsing. In ACL. 1341–1350.

[41] Yupeng Xie, Yuyu Luo, Guoliang Li, and Nan Tang. 2024. HAIChart: Human and AI Paired Visualization System. Proc. VLDB Endow. 17, 11 (2024), 3178–3191.

[42] Yicun Yang, Zhaoguo Wang, Yu Xia, and Zhuoran Wei. 2025. Automated Validation and Fixing of Text-to-SQL Translation with Execution Consistency.

[43] Ziyu Yao, Yu Su, Huan Sun, and Wen-tau Yih. 2019. Model-based Interactive Semantic Parsing: A Unified Framework and A Text-to-SQL Case Study. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, 5446–5457. https://doi.org/10.18653/V1/D19-1547

[44] Yilin Ye, Jianing Hao, Yihan Hou, Zhan Wang, Shishi Xiao, Yuyu Luo, and Wei Zeng. 2024. Generative AI for visualization: State of the art and future directions.

Trans. Vis. Comput. Graph. 28, 1 (2022), 217–226.

Vis. Informatics 8, 1 (2024), 43–66.

[45] 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. 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, Brussels, Belgium, October 31 - November 4, 2018, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (Eds.). Association for Computational Linguistics, 3911–3921. https://doi.org/10.18653/V1/D18-1425

[46] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. 2025. AFlow: Automating Agentic Workflow Generation. In The Thirteenth International Conference on Learning Representations. https://openreview.net/forum?id=z5uVAKwmjf

[47] Yi Zhang, Jan Deriu, George Katsogiannis-Meimarakis, Catherine Kosten, Georgia Koutrika, and Kurt Stockinger. 2023. ScienceBenchmark: A Complex Real-World Benchmark for Evaluating Natural Language to SQL Systems. Proc. VLDB Endow. 17, 4 (2023), 685–698. https://doi.org/10.14778/3636218.3636225

[48] Zhengxuan Zhang, Zhuowen Liang, Yin Wu, Teng Lin, Yuyu Luo, and Nan Tang. 2025. DataMosaic: Explainable and Verifiable Multi-Modal Data Analytics through Extract-Reason-Verify. CoRR abs/2504.10036 (2025).

[49] Zhengxuan Zhang, Yin Wu, Yuyu Luo, and Nan Tang. 2024. MAR: Matching-Augmented Reasoning for Enhancing Visual-based Entity Question Answering. In Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 1520–1530. https://doi.org/10.18653/v1/2024.emnlp-main.91

[50] Yizhang Zhu, Shiyin Du, Boyan Li, Yuyu Luo, and Nan Tang. 2024. Are Large Language Models Good Statisticians?. In NeurIPS.

[51] Yizhang Zhu, Runzhi Jiang, Boyan Li, Nan Tang, and Yuyu Luo. 2025. EllieSQL: Cost-Efficient Text-to-SQL with Complexity-Aware Routing. arXiv:2503.22402 [cs.DB] https://arxiv.org/abs/2503.22402

A Detected Errors in NL2SQL Benchmarks

In Table 3 and Table 4, we list SQL queries with semantic errors found in the BIRD and Spider benchmarks.

- Table 3 lists the indices of SQL queries from the BIRD [15] and Spider [45] development benchmarks that we identified as containing semantic errors. The presence of such errors can cause query results to deviate from the expected answers derived from the natural language question (NL).
- Table 4 illustrates specific instances of the semantic errors identified, providing concrete examples. Each example comprises the NL, the incorrect SQL statement, and color-coded annotations that highlight the erroneous segments alongside the correct query intent. The showcased errors exemplify various categories, such as redundant fields, incorrect sorting directions, and improper filtering conditions.

B Examples in Our Error Taxonomy

B.1 Value-related Errors

This example demonstrates both data format and value mismatch errors. The model incorrectly uses the MM/DD/YY format ('01/01/23') instead of the database's standard YYYY-MM-DD format ('2023-01-01'), while also treating numeric values as strings ('30' instead of 30).

Value-Related Example	
Query: Find all courses that started after January 1st, 2023 and have more than 30 students.	
Incorrect SQL:	**Correct SQL:**
SELECT c.name	SELECT c.name
FROM course c	FROM course c
WHERE c.start_date >'01/01/23'	WHERE c.start_date > '2023-01-01'
AND c.enrollment > '30'	AND c.enrollment > 30

B.2 Condition-related Errors

This example demonstrates explicit and implicit condition errors. The explicit condition mismatch appears in using the wrong attribute (dept) and wrong value (CS) instead of matching the course name, while the implicit condition error manifests in missing the grade IS NOT NULL check for completed courses.

Condition-Related Example	
Query: Find the students' names and their grades in the Database course.	
Incorrect SQL:	**Correct SQL:**
SELECT s.name, e.grade	SELECT s.name, e.grade
FROM student AS s	FROM student AS s
JOIN enrollment AS e	JOIN enrollment AS e
ON s.id = e.student_id	ON s.id = e.student_id
JOIN course AS c	JOIN course AS c
ON e.course_id = c.id	ON e.course_id = c.id
WHERE c.dept = 'CS'	WHERE c.name = 'Database'
	AND e.grade IS NOT NULL

B.3 Function-related Errors

This example illustrates two function-related errors: using the AVG aggregate function in the WHERE clause instead of the appropriate HAVING clause, and neglecting to use the ROUND math function to define decimal precision.

Function-Related Example	
Query: Calculate the average grade (with two decimal places) for each course, and list courses with averages above 85.	
Incorrect SQL:	**Correct SQL:**
SELECT c.name,	SELECT c.name,
AVG(e.grade)	ROUND(AVG(e.grade), 2)
FROM course c	FROM course c
JOIN enrollment e	JOIN enrollment e
ON c.id = e.course_id	ON c.id = e.course_id
WHERE AVG(e.grade) > 85	GROUP BY c.name
GROUP BY c.name	HAVING AVG(e.grade) > 85

Table 3: Semantic Errors in BIRD and Spider Benchmarks.

Benchmark	Index of (NL, SQL) Examples with Semantic Errors
Spider	67, 101, 494, 555, 579, 773, 774, 811, 812, 819, 820, 128, 129, 961, 962, 177
BIRD	1027, 1029, 519, 523, 530, 23, 70, 72, 584, 1107, 600, 602, 603, 94, 1119, 1120, 1121, 631, 632, 635, 125, 639, 640, 129, 642, 646, 649, 144, 145, 656, 1170, 667, 679, 682, 1197, 686, 687, 1199, 1204, 693, 182, 186, 194, 1219, 709, 710, 1225, 1233, 1243, 221, 1247, 1248, 1256, 1265, 1269, 247, 1273, 1274, 252, 254, 1279, 1284, 271, 1300, 281, 1308, 296, 1322, 812, 309, 341, 342, 343, 855, 349, 360, 1388, 386, 387, 388, 389, 398, 406, 1450, 1454, 431, 1458, 441, 442, 443, 446, 447, 966, 458, 970, 1482, 973, 978, 1491, 986, 993, 484, 1000, 1004, 1530, 1531

Table 4: Samples of Detected Semantic Errors in Real-world Benchmarks.

Source	NL Query	SQL Query
Spider-494	List the name, date and result of each battle.	`SELECT name , date t1.age #Need result FROM battle;`

Spider-555	What is the mobile phone number of the student named Timmothy Ward?	`SELECT cell_mobile_number FROM students`
		`WHERE first_name = 'timmothy' #Need 'Timmothy'`
		`AND last_name = 'ward'; '#Need 'Ward'`

Spider-579	What are the different addresses that have students living there?	`SELECT count(DISTINCT current_address_id)` `FROM Students;#No Need count`

Spider-773	What are the countries that have greater surface area than any country in Europe?	`SELECT Name FROM country WHERE SurfaceArea > (`
		`SELECT min(SurfaceArea) #max(...)`
		`FROM country WHERE Continent = "Europe");`

BIRD-252	What are the atoms that can bond with the atom that has the element lead? Evidence: Atom that has the element lead refers to atomid where element = 'pb';	`SELECT T2.atomid, T2.atom_id2 #No Need`
		`FROM atom AS T1`
		`INNER JOIN connected AS T2 ON T1.atomid = T2.atomid`
		`WHERE T1.element = 'pb';`

BIRD-812	List down at least five full names of superheroes with blue eyes. Evidence: Blue eyes refers to colour.colour = 'Blue' WHERE eyecolourid = colour.id;	`SELECT T1.superheroname #Need T1.full_name`
		`FROM superhero AS T1`
		`INNER JOIN colour AS T2 ON T1.eyecolourid = T2.id`
		`WHERE T2.colour = 'Blue' LIMIT 5;`

BIRD-986	In which race did the fastest 1st lap time was recorded? Evidence: Please indicate the time in milliseconds. Fastest refers to Min(time);	`SELECT T1.milliseconds #Need races.name`
		`FROM lapTimes AS T1`
		`INNER JOIN races AS T2 ON T1.raceId = T2.raceId`
		`WHERE T1.lap = 1 ORDER BY T1.time LIMIT 1;`

BIRD-1029	What are the speed in which attacks are put together of the top 4 teams with the highest build Up Play Speed? Evidence:Speed in which attacks are put together refers to buildUpPlaySpeed;highest build up play speed refers to MAX(buildUpPlaySpeed);	`SELECT t1.buildUpPlaySpeed`
		`FROM Team_Attributes AS t1 INNER JOIN Team AS t2`
		`ON t1.team_api_id = t2.team_api_id`
		`ORDER BY t1.buildUpPlayDribbling ASC #Need DESC`
		`LIMIT 4;`

BIRD-1248	How many patients born after 1980/1/1 have an abnormal fibrinogen level? Evidence: Born after 1980/1/1 refers to Birthday > '1980-01-01'; normal fibrinogen level refers to FG between 150 and 450; Should return the number of distinct patients.	`SELECT COUNT(DISTINCT T1.ID) FROM Patient AS T1`
		`INNER JOIN Laboratory AS T2 ON T1.ID = T2.ID`
		`WHERE T2.FG <= 150 OR T2.FG >= 450`
		`#Not(>=150 and <=450) is <150 or >450`
		`AND T1.Birthday > '1980-01-01';`