Scientific Research Publishing

Journal of Computer and Communications, 2025, 13(2), 236-249
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Comparing Large Language Models for Generating Complex Queries

Limin Ma[1], Ken Pu[1], Ying Zhu[2], Wesley Taylor[3]

[1]Faculty of Science, Ontario Tech University, Oshawa, Canada
[2]Faculty of Business and IT, Ontario Tech University, Oshawa, Canada
[3]Legion Development Group, Oshawa, Canada
Email: ken.pu@ontariotechu.net

How to cite this paper: Ma, L.M., Pu, K., Zhu, Y. and Taylor, W. (2025) Comparing Large Language Models for Generating Complex Queries. *Journal of Computer and Communications*, **13**, 236-249.
https://doi.org/10.4236/jcc.2025.132015

Received: December 8, 2024
Accepted: February 25, 2025
Published: February 28, 2025

Abstract

This study presents a comparative analysis of a complex SQL benchmark, TPC-DS, with two existing text-to-SQL benchmarks, BIRD and Spider. Our findings reveal that TPC-DS queries exhibit a significantly higher level of structural complexity compared to the other two benchmarks. This underscores the need for more intricate benchmarks to simulate realistic scenarios effectively. To facilitate this comparison, we devised several measures of structural complexity and applied them across all three benchmarks. The results of this study can guide future research in the development of more sophisticated text-to-SQL benchmarks. We utilized 11 distinct Language Models (LLMs) to generate SQL queries based on the query descriptions provided by the TPC-DS benchmark. The prompt engineering process incorporated both the query description as outlined in the TPC-DS specification and the database schema of TPC-DS. Our findings indicate that the current state-of-the-art generative AI models fall short in generating accurate decision-making queries. We conducted a comparison of the generated queries with the TPC-DS gold standard queries using a series of fuzzy structure matching techniques based on query features. The results demonstrated that the accuracy of the generated queries is insufficient for practical real-world application.

Keywords

Text-to-SQL, Evaluation, LLM, Generative AI

1. Introduction

The task of generating SQL queries from natural language (NL) has been a long-standing problem in the field of natural language processing (NLP) and databases.

The task is important as it can enable non-expert users to interact with databases without having to learn SQL. The task is also important for database administrators, who can use the generated SQL queries as a starting point for further optimization. The task is also important for data scientists, who can use the generated SQL queries to analyze data and generate insights.

In recent years, large language models (LLMs) have shown impressive performance on a wide range of NLP tasks. LLMs are pre-trained on large amounts of text data and fine-tuned on specific tasks. LLMs have been shown to achieve state-of-the-art performance on a wide range of NLP tasks, including text-to-SQL generation. However, the task of generating SQL queries from NL is challenging due to the complex structure of SQL queries and the need for accurate decision making.

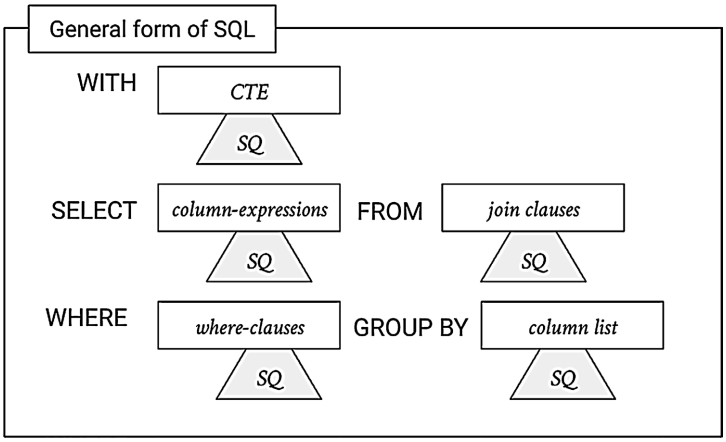

Figure 1. The general form of a SQL query. Note all triangles labeled as *SQ* are possible occurrences of sub-queries.

In an effort to evaluate the performance of LLMs on the task of generating SQL queries from NL, two major benchmarks have been proposed: BIRD and Spider. BIRD is a benchmark for text-to-SQL generation that consists of over 12,000 NL questions and their corresponding SQL queries. Spider is a benchmark for text-to-SQL generation that consists of 10,000 NL questions corresponding to 5693 SQL queries. Both benchmarks have been widely used to evaluate the performance of LLMs on the task of generating SQL queries from NL.

In this study, we present a comparative analysis of the TPC-DS benchmark [1] with the BIRD [2] and Spider [3] benchmarks. The TPC-DS benchmark is a widely used benchmark for evaluating the performance of database systems. The benchmark consists of a set of complex SQL queries that simulate real-world decision-making queries. Our findings reveal that TPC-DS queries exhibit a significantly higher level of structural complexity compared to the BIRD and Spider benchmarks. This underscores the need for more intricate benchmarks to simulate realistic scenarios effectively. Furthermore, we used 11 distinct LLMs to generate SQL queries based on the query descriptions provided by the TPC-DS benchmark. The prompt engineering process incorporated both the query description as outlined in the TPC-DS specification and the database schema of TPC-DS. Our

findings indicate that current state-of-the-art generative AI models fail to generate accurate decision-making queries. We conducted a comparison of the generated queries with the TPC-DS gold standard queries using a series of fuzzy structure matching techniques based on query features. The results demonstrated that the accuracy of the generated queries is insufficient for practical applications in the real world.

2. Related Work

The field of text-to-SQL has seen significant progress due to the integration of large language models (LLMs) [4] [5]. Innovative techniques have been proposed for improving the accuracy and efficiency of SQL query generation from natural language inputs. We discuss several key works in this domain, and present their contributions and how they relate to our work.

Owda *et al.* [6] present an early system that incorporates Information Extraction (IE) techniques into an Enhanced Conversation-Based Interface of Relational Databases (C-BIRD) to generate dynamic SQL queries. Their approach allows conversational agents to interact with users in natural language, generating SQL queries without any user requirement of prior SQL knowledge. Since then, LLM-based solutions have dominated the field of text-to-SQL. Furthermore, two major benchmarks, BIRD and Spider, have been proposed to evaluate the performance of LLMs on the task of generating SQL queries from NL. Yu *et al.* [3] introduce Spider, a large-scale, complex, and cross-domain semantic parsing and text-to-SQL dataset. Spider includes 10,181 questions and 5,693 unique complex SQL queries across 200 databases covering 138 different domains, requiring models to generalize well to both new SQL queries and database schemas. This work is foundational in the text-to-SQL field, demonstrating the challenges in handling complex and diverse SQL queries. Li *et al.* [2] introduce BIRD, a large-scale benchmark for text-to-SQL tasks that aims to bridge the gap between academic research and real-world applications. BIRD benchmark measures both the accuracy and efficiency of text-to-SQL models, providing a comprehensive evaluation of model performance.

A number of text-to-SQL models and techniques have been proposed and were heavily based on the BIRD and Spider benchmarks. Li *et al.* [7] introduce the Fuzzy Semantic to Structured Query Language (F-SemtoSql) neural approach, designed to tackle the complex and cross-domain task of text-to-SQL generation. Pourreza and Rafiei [8] propose DIN-SQL, a method that improves text-to-SQL performance by decomposing the task into smaller subtasks and using in-context learning with self-correction. Chen *et al.* [9] present the Open-SQL Framework, designed to enhance text-to-SQL performance on open-source LLMs. They introduce novel strategies for supervised fine-tuning and the *openprompt* strategy for effective question representation. Li *et al.* [10] propose ResdSQL, a framework that decouples schema linking and skeleton parsing for text-to-SQL tasks. By addressing the complexity of parsing schema items and SQL skeletons separately, their method improves the accuracy of SQL query generation.

3. Features and Measures of Target SQL Queries

We are interested in scenarios of complex analytical needs involving SQL queries of complex structures. Existing research on SQL query generation by generative AI as discussed in Section 1, they focus on queries with limited complexity. Practical queries often exhibit greater structural complexity for several reasons. As the need for data analytics grows, the corresponding query also becomes increasingly complex in structure [11]. In addition, as shown in the study by Taipalus [12], SQL becomes increasingly complex for more sophisticated database designs.

Evaluating the text-to-SQL capability of the general-purpose LLMs would require:

- A collection of query complexity features that captures the query structural complexity: these features can be used to distinguish *simple* scenarios to more complex ones.
- A widely accepted benchmark with a well defined database schema, and a collection of examples natural language description and gold standard SQL query. We also need the SQL queries to be reasonably complex according to the query complexity features.

Towards the first goal, we will quantify the structural complexity between simple queries and complex queries using multidimensional query features. Using these features, we can perform comparative studies of different query workloads.

Towards the second goal, we utilize the TPC-DS [1] benchmark. The TPC-DS benchmark has been adopted by all major relational database vendors for performance evaluation and comparison. The salient properties of TPC-DS queries are that the SQL queries are well formed according to idiomatic practices, and are specified by natural language descriptions. This makes TPC-DS benchmark an ideal choice for our needs.

Example 1. *In the TPC-DS* [1], *consider a query with the following description:*

Find customers who have returned items more than 20% more often than the average customer returns for a store in a given state for a given year.

Corresponding workload query as specified in TPC-DS is a complex nested SQL query Q_{tpcds}.

```
WITH customer_total_return AS (SELECT sr_customer_sk AS ctr_customer_sk
                                    , sr_store_sk AS ctr_store_sk
                                    , sum(sr_fee) AS ctr_total_return
                               FROM store_returns, date_dim
                               WHERE sr_returned_date_sk = d_date_sk
                                 AND d_year = 2000
                               GROUP BY sr_customer_sk, sr_store_sk)

  SELECT c_customer_id
  FROM customer_total_return AS ctr1, store, customer
  WHERE ctr1.ctr_total_return > (SELECT avg(ctr_total_return) * 1.2
                                   FROM customer_total_return AS ctr2
                                   WHERE ctr1.ctr_store_sk = ctr2.ctr_store_sk)
    AND s_store_sk = ctr1.ctr_store_sk
    AND s_state = 'TN'
    AND ctr1.ctr_customer_sk = c_customer_sk
  ORDER BY c_customer_id
```

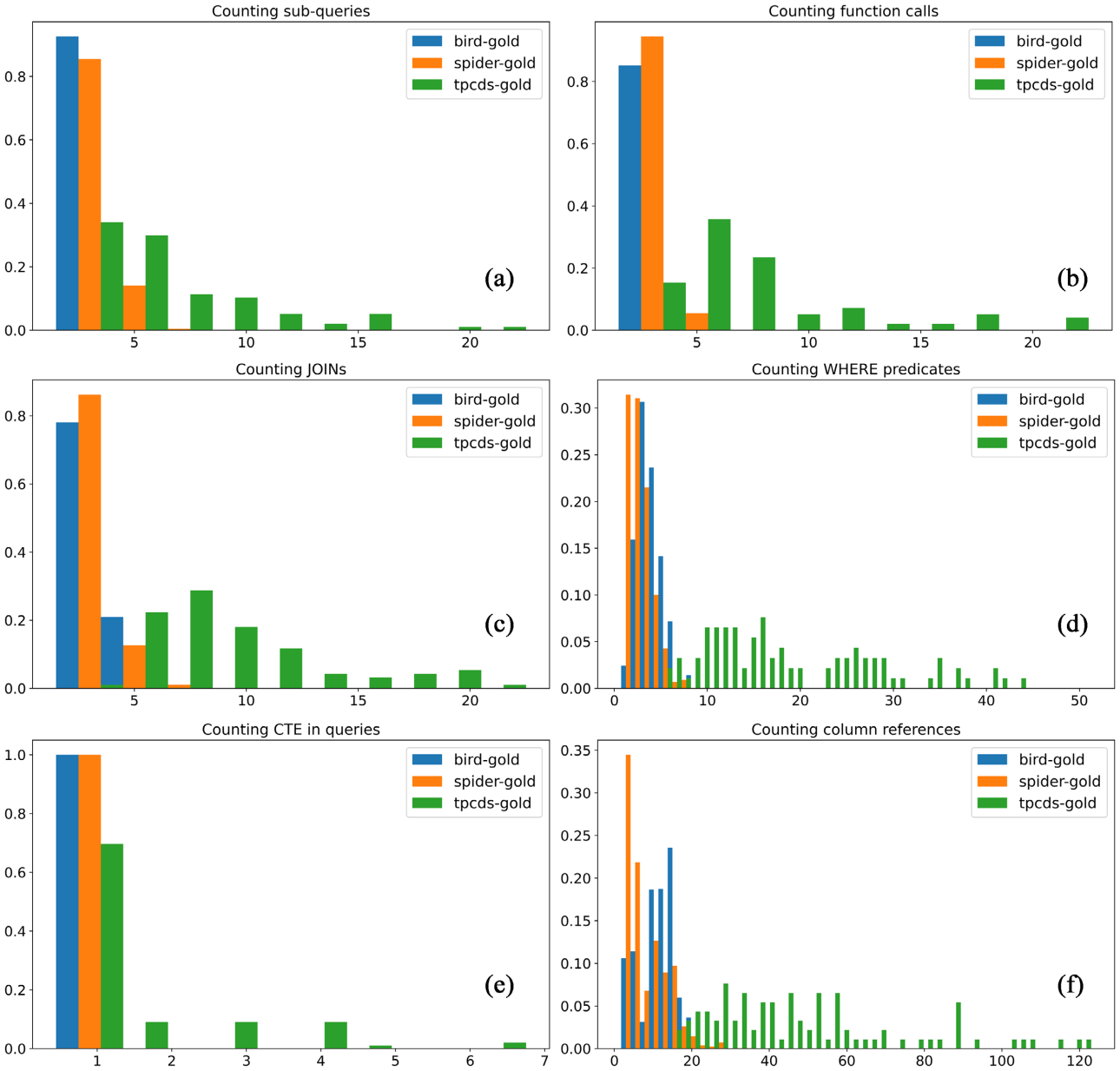

Figure 2. Feature based comparison of benchmarks. (a) $\mathrm{SQ}(Q)$; (b) $\mathrm{JOIN}(Q)$; (c) $\mathrm{Func}(Q)$; (d) $\mathrm{WHERE}(Q)$; (e) $\mathrm{CTE}(Q)$; (f) $\mathrm{Cols}(Q)$.

This is significantly different from typical queries in the standard Text-to-SQL benchmarks. Here is a query given in the SPIDER benchmark:

What are the ids of the students who either registered or attended a course?

The corresponding SQL query is given as Q_{spider}:

```
SELECT student_id FROM student_course_registrations
UNION
SELECT student_id FROM student_course_attendance
```

□

The two queries in Example 1 are quite different in structure. This type of structural difference is commonly observed between queries in TPC-DS and existing text-to-SQL benchmarks. The features we define in section help us to quantify these differences, and highlight the necessity of a comparative study of the performance of LLMs for SQL generation from texts when the intended queries are more complex.

The general structure of a SQL query is shown in Figure 1. Given the nested structures of SQL, Q can be a complex self-referencing tree of SQL queries.

We will define a number of features that can be used to characterize the structural complexity of SQL queries. We argue that the bag-valued features are useful in performing semantic comparison of pairs of SQL queries.

Let $\boldsymbol{SQ}(Q)$ be all sub-queries in Q regardless of where it appears in Q. We define a collection of features derived from Q and its sub-queries. These features are divided into two categories: bag-valued features and numeric features. Namely, a feature F is a mapping from SQL query to either a bag of elements or an integer.

$$F_{\text{bag}} : \boldsymbol{SQL} \to \boldsymbol{Bags} \tag{1}$$

$$F_{\text{count}} : \boldsymbol{SQL} \to \mathbb{N} \tag{2}$$

We note that any bag feature F is also naturally a numeric feature $F^{\#}$: $F^{\#}(Q) = \left|F(Q)\right|$.

3.1. Bag-Valued Features

We will consider the following bag-valued features. These features are measures of the structural complexity of the SQL query. Namely, the more complex a query is, the more elements the feature (bag) will contain.

Columns: the set of distinct columns included in the select *column expressions* of Q and its subqueries.

$$\text{Cols}(Q) = \left\{c \in \text{Columns} : c \in \text{SelectExpr}(Q)\right\} \cup \bigcup_{Q' \in \text{SQ}(Q)} \text{Cols}(Q') \tag{3}$$

Example 2. *Consider the queries in Example 1, the columns of the two queries are:*

$$\text{Cols}\left(Q_{\text{tpcds}}\right)$$
$$= \left\{\text{c\_customer\_id}, \text{ctr\_total\_return}, \text{sr\_customer\_sk}, \text{sr\_fee}, \text{sr\_store\_sk}\right\}$$

$$\text{Cols}\left(Q_{\text{spider}}\right) = \left\{\text{student\_id}\right\}$$

We can measure the distinction based on the size of the bag features.

- $\text{Col}^{\#}\left(Q_{\text{tcpds}}\right) = 5$
- $\text{Col}^{\#}\left(Q_{\text{spider}}\right) = 1$

(1) Relations: $\text{Tables}(Q)$ the set of distinct relations in the SQL query. This is computed as:

$$\text{Relations}(Q) = \left\{r \in \text{Relations} : r \in \text{FromClause}(Q)\right\} \cup \bigcup_{Q' \in \text{SQ}(Q)} \text{Relations}(Q') \tag{4}$$

(2) Where predicates: $\mathrm{WHERE}(Q)$ the set of distinct where basic predicates in the SQL query and its subqueries. Each basic predicate is given in the form of

$$\langle \mathrm{pred}, \mathrm{expr}_i, \cdots \rangle$$

where pred is the boolean operator supported by SQL, and expr_i is an expression over column names, constants, and functions. All columns aliases are normalized to the canonical form.

(3) JOINs: $\mathrm{JOIN}(Q)$ is the set of joins of Q is defined to be pairs of physical relations (t, t') that participate in a JOIN in the *logical* query plan of Q and its subqueries.

(4) Aggregation: $\mathrm{Agg}(Q)$ is the set of aggregated columns in Q and its subqueries. Each aggregated column is given in the form of $\langle \mathrm{agg}, \mathrm{expr}, \mathrm{groupby\ columns} \rangle$ where agg is the aggregation function (e.g., SUM, AVG, COUNT, etc.) and expr is an expression over column names, constants, and functions.

(5) Functions: $\mathrm{Func}(Q)$ is the set of SQL functions that appear in Q and its subqueries.

3.2. Feature Invariance for SQL Generation

An important property to note is the *feature invariance* of the generated SQL queries. We say that a feature F is invariant to it remains the same with two equivalent queries.

Definition 1 Given two SQL queries Q_1 and Q_2 are equivalent and without redundancy. Then a feature $F : \mathbf{SQL} \to \circ$ is an invariant feature if $F(Q_1) = F(Q_2)$.

Given that generative AI can be highly nondeterministic, e.g. temperature setting [13], it may generate functional equivalent SQL queries in many different forms. We want to ensure that our features remain an accurate metric with respect to all *reasonably* generated queries. By reasonable queries, we mean that the SQL does not contain any redundancy.

Claim 1. *The bag-valued features in Section 3.1 are invariant features.*

The comparison of the two major Text-to-SQL benchmarks of SPIDER [3] and BIRD [2] with TPC-DS is shown in **Figure 2**. We can note the *long tail* distribution that is unique to the TPC-DS workload. Namely, bag features associated with the TPC-DS queries are much larger, indicating a higher level of complexity in the analytical tasks.

3.3. Numeric Features

In addition to the bag features, we introduce to numeric features: counts of common table expressions (CTE) and count of subqueries.

Common Table Expressions: $\mathrm{CTE}(Q)$ is the number of common-table-expressions defined by Q. We use CTE count as an indicator of the structural complexity and readability of the query Q.

Sub-queries: we measure $\|\mathrm{SQ}(Q)\|$, the number of subqueries of Q as an indicator of the structural complexity of Q.

These numeric features are not invariant in the sense that two equivalent queries may have different number of CTE and subqueries. Nonetheless, these features are good indicators of the structural complexity of the queries, allowing as to compare the LLM's capability of generating complex SQL structures, especially for queries whose standard form given in TPC-DS benchmark are highly nested and with many CTE.

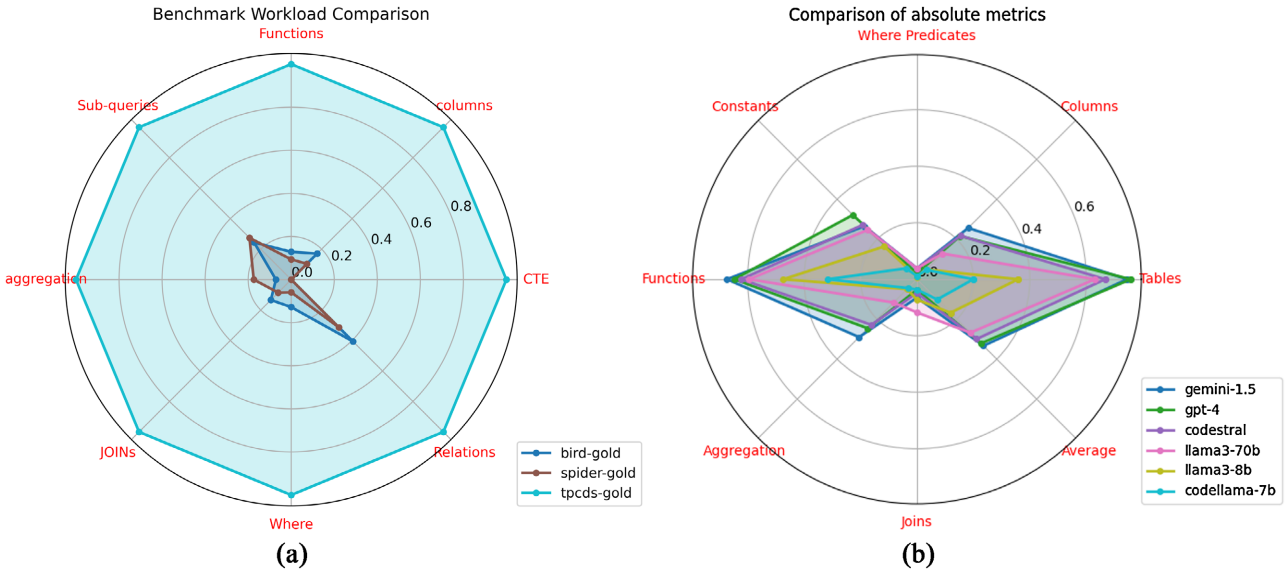

Figure 3. Feature comparisons. (a) Comparison of SQL benchmarks; (b) generated queries vs TPC-DS gold queries.

4. TPC-DS as a Complex Text-to-SQL Benchmark

We compare the complexity of WHERE clauses in the SQL queries from the three SQL benchmarks. Complexity of the WHERE clause is measured by the number of basic predicates in the WHERE clause. A basic predicate is one that has a single condition, such as year = 1998. Compound predicates comprising multiple conditions, such as year = 1998 AND ss_quantity BETWEEN 81 AND 100, are counted as multiple basic predicates. The counts of basic predicates in WHERE clauses for the three benchmarks are plotted in a histogram in **Figure 2(d)**. While the BIRD and SPIDER benchmarks are similar in their distribution of WHERE clause predicate counts, the TPC-DS benchmark clearly exhibits substantially greater WHERE clause complexity in its SQL queries. TPC-DS has such greater WHERE clause complexity than the other two benchmarks that its distribution of the counts hardly overlaps with the others. Almost all TPC-DS queries have multiple times the WHERE predicate counts than all the queries from the other two benchmarks.

Another metric used to measure the complexity of SQL queries is the number of Common Table Expressions (CTE). The role of a CTE is to minimize repetition of subqueries as well as making the overall SQL statement more structured and

thus easier to maintain. We count the number of CTE in each query from the three benchmarks, and plot the histogram in Figure 2(e). It can be seen that both BIRD and SPIDER workloads do not require CTE due to their low degree of semantic complexity. In contrast, TPC-DS includes heavy usage of CTE in the gold queries.

We also count the number of distinct columns referenced in SQL queries. The number of distinct columns involved in a query is an indicator of the semantic complexity of that query. A column is included in the count whether it appears in SELECT projections, in JOIN conditions, or in WHERE predicates. The distributions of the column counts for the benchmarks are shown in Figure 2(f). This graph is similar to the one for the count of WHERE clause predicates. Again, the number of columns used in the gold queries of TPC-DS is, in most cases, an order of magnitude larger than the number of columns in BIRD and SPIDER queries. The overlap is minimal, which means that the vast majority of TPC-DS workload reference a much larger set of columns than BIRD and SPIDER workloads. This is another indication of the significantly higher query complexity of TPC-DS.

In addition to the above, we also consider three more metrics towards our evaluation of overall query complexity. SQL queries often include calls to functions that perform data transformation. These functions include scalar and aggregation functions. They contribute to the overall query complexity. We therefore count the number of expressions that contain function calls, whether they are part of a SELECT projection, a WHERE clause, or an aggregation. Also, subqueries are undoubtedly one of the signs of the semantic complexity of queries, therefore we count the number of subqueries in each query. We further consider the presence of JOINs since a JOIN means the need for additional source tables and that contributes to query complexity. By counting the number of JOINS, we get a measure of how many data sources have to be combined to generate the final result.

The histograms for these three metrics applied to the three benchmarks are shown, respectively, in Figure 2(a)-(c). The distributions of function calls, subqueries and joins are all quite similar. The gap between TPC-DS workload and the workloads of BIRD and SPIDER is even greater than that in the CTE metric above. Most of TPC-DS queries have more than 5 function calls, some of them have more than 10. In contrast, most of BIRD and SPIDER have 3 or fewer function calls, and never more than 5. TPC-DS stands out as the only one among the three that makes regular use of subqueries. The other two workloads appear to make minimal use of subqueries. Just like subqueries and function calls, TPC-DS clearly outstrips BIRD and SPIDER in the number of JOINs. This means that queries from TPC-DS almost always require data from a much larger number of tables in order to obtain the result.

Overall, TPC-DS queries are significantly more complex than BIRD and SPIDER queries in every metric we have considered as shown in the radar plot in Figure 3(a).

The comparison results are aggregated in Figure 3(a) that shows the mean count of each metric of query complexity for the three benchmarks, normalized by the

mean counts for TPC-DS. TPC-DS clearly exhibits much greater query complexity in every metric. Not only is its mean metrics higher than BIRD and SPIDER, they are multiple times higher in every metric. This indicates that TPC-DS is a much more complex benchmark than BIRD and SPIDER, and that it is likely to be more challenging for AI models to generate queries based on TPC-DS workload.

The query features defined in Section 3 allow us to perform comparative analysis of different LLMs in terms of their respective generated queries. Table 1 compares the generated queries from different LLMs with the gold standard TPC-DS queries using a feature based cosine similarity measure. Table 2 compares the generated queries by different LLMs with the TPC-DS gold standard using the Jaccard similarity between the respective bag-valued features. Details of the query generation methodology and similarity measures are given in Section 5.

Table 1. Feature based comparison of generated SQL queries.

LLM	CTE	Selected Columns	Functions	Nested Select	Aggregation	joinCount	Where Predicates	Number of Joins
tpcds-gold	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
gemini-1.5	2.38	0.79	0.62	1.01	0.95	0.91	0.55	0.87
gpt-4	1.11	0.77	0.63	0.69	0.80	0.73	0.53	0.90
codestral	0.19	0.67	0.69	0.53	0.63	0.63	0.53	0.90
mixtral-8x22b	1.99	0.96	0.70	1.04	1.06	0.96	0.67	0.90
llama3-70b	2.03	0.81	0.75	1.37	1.02	1.04	0.67	0.94
llama3-8b	0.51	0.63	0.63	0.67	0.65	0.59	0.52	0.84
codellama-7b	0.03	0.39	0.29	0.28	0.32	0.24	0.25	0.55

Table 2. Similarity comparison of generated queries with TPC-DS gold queries.

Features LLM	Tables	Columns	Where Predicates	Constants	Functions	Aggregation	Joins	Average
gemini-1.5	0.74	0.26	0.04	0.27	0.67	0.29	0.06	0.33
gpt-4	0.76	0.22	0.02	0.32	0.65	0.25	0.04	0.32
codestral	0.67	0.22	0.04	0.27	0.62	0.23	0.05	0.30
mixtral-8x22b	0.69	0.17	0.02	0.30	0.60	0.24	0.06	0.30
mistral-large	0.66	0.19	0.06	0.22	0.54	0.24	0.08	0.28
llama3-70b	0.63	0.13	0.04	0.25	0.60	0.12	0.12	0.27
codellama-70b	0.36	0.12	0.03	0.13	0.44	0.11	0.11	0.18
llama3-8b	0.36	0.05	0.01	0.16	0.47	0.05	0.07	0.17
codellama-13b	0.28	0.08	0.02	0.10	0.34	0.05	0.07	0.13
codellama-35b	0.24	0.06	0.01	0.09	0.32	0.05	0.07	0.12
codellama-7b	0.20	0.05	0.01	0.05	0.32	0.04	0.04	0.10

5. Evaluating LLMs Using TPC-DS Workload

We have used 11 different LLMs to generate SQL queries based on the TPC-DS workload. The LLMs are: gemini-1.5, gpt-4, codestral, mixtral-8x22b, mistral-large, llama3-70b, codellama-70b, llama3-8b, codellama-13b, codellama-35b, and codellama-7b. We have used the following prompt to generate the queries.

System Prompt:

```
You are a skilled PostgreSQL
relational database developer.
Your task is to generate a PostgreSQL
query to answer user's question based
on given database schema without
further explanations.

The database schema is:
{{ DDL_statements }}
```

User Prompt:

```
User question:
{{ question }}
```

For all the LLMs, we have encountered invalid queries generated by LLM, either with syntax error or containing incorrect schema information due to hallucination. Our implementation utilizes the PostgreSQL database system to validate the generated queries. In case of invalid queries, we retry the generation process with a new user message containing the error message reported by PostgreSQL.

Additional User Prompt:

```
Correct the query based on the error
message.

Query:
{{ sql }}

Error message:
{{ error_message }}
```

The generation process is limited to maximum of three retries. LLMs have different success rate after three retries. The number of successful queries generated by each LLM is shown in Table 3. The LLMs gpt-4, gemini-1.5, and mistral-large have the highest success rate in generating queries based on the TPC-DS workload. We can observe that the smaller models such as llama3-8b and codellama-34b or smaller have a very low success rate in generating queries.

5.1. Structural Complexity of Generated Queries

In order to compare the structural complexity of the queries generated by the different LLMs, we again look at several count measures, such as number of

Table 3. Number of syntactically correct queries generated based on TPC-DS query description.

LLM	Success rate
gpt-4	94%
gemini-1.5	86%
mistral-large	75%
codestral	77%
mistral-8x22b	67%
llama3-70b	74%
llama4-8b	8%
codellama-70b	56%
codellama-34b	29%
codellama-13b	18%
codellama-7b	17%

columns and joins. For each type of measure and each LLM, the mean of the counts in the queries is calculated, and then normalized over the mean count for TPC-DS queries. These normalized mean counts are given in Table 1. We can see that gemini 1.5, gpt-4, mixtral and llama3-70b match the closest the structural complexity of TPC-DS. The other LLMs generate markedly less complex queries.

It is encouraging that large LLMs are capable of generating queries that can match the complexity of TPC-DS workload. But more importantly we want to evaluate the quality of these queries with respect to the gold queries given by TPC-DS. In the next section, we will evaluate each of the generated queries using the bag-valued features as defined in Section3.

5.2. Accuracy of Generated Queries

We have defined 7 bag-valued features defined in Section 8. Each feature is a function mapping SQL to a bag of discrete values. To compare two queries: Q the generated query by a LLM, and Q^* the corresponding gold query given by TPC-DS benchmark, we use the Jaccard similarity coefficient between the two bags of values for each feature given by:

$$\text{sim}_F\left(Q, Q^*\right) = \frac{\left|F\left(Q\right) \cap F\left(Q^*\right)\right|}{\left|F\left(Q\right) \cup F\left(Q^*\right)\right|}$$

Table 2 shows the mean similarity between the queries generated by 11 LLMs and the TPC-DS queries. All LLMs are instructed with the instructions given in Section 5. Five of the top performing LLMs are visualized in a radar plot in Figure 3(b) and Figure 4(a). One can see that the accuracy of the generated queries is insufficient for practical real-world application. In particular, none of the LLMs are able to generate queries that match the **WHERE** predicates and **JOIN** pairs used by TPC-DS. This highlights the unique challenges posed by the TPC-DS workload in comparison with the BIRD and SPIDER benchmarks.

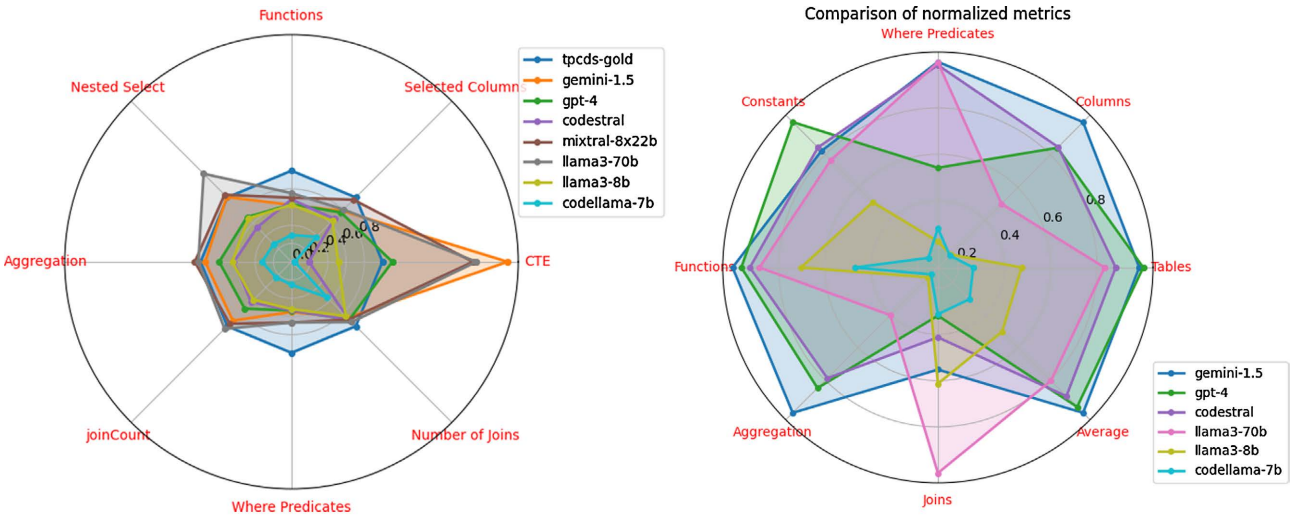

Figure 4. (a) Comparison of generated queries; (b) Comparison using normalized metrics.

6. Conclusion and Future Work

We have compared TPC-DS with existing text-to-SQL benchmarks, BIRD and SPIDER, and found that TPC-DS queries exhibit a significantly higher level of structural complexity compared to the other two benchmarks. We have also evaluated the performance of 11 LLMs in generating SQL queries based on the TPC-DS workload. Our findings indicate that the current state-of-the-art generative AI models fall short in generating accurate decision-making queries. The accuracy of the generated queries is insufficient for practical real-world application.

Towards a satisfactory solution to complex SQL generation, we identify the following areas of future work.

Evaluating incremental SQL generation: our experiments show the need of incremental SQL generation to improve the accuracy of generated queries. This motivates us to investigate novel prompt strategies. Based on the observation in Figure 3(b), we propose to generate WHERE clause and JOIN pairs as a separate LLM prompt, and use the results to prompt LLMs for the rest of the query.

Fine-tuning smaller models: our experiments show that smaller models are not able to match the performance of larger models in the context of complex SQL generation. In situations where cloud based LLMs are not feasible (due to cost or privacy concerns), we propose to investigate fine-tuning of smaller models to improve their performance. In particular, smaller models can certainly be improved in terms of syntax and schema accuracy during generation. Also, ensemble methods involving multiple fine-tuned LLMs can be used to combine the outputs of multiple smaller models to improve the overall accuracy.

Human-in-the-loop: complex SQL generation is a challenging task, and the burden of SQL generation should not be entirely on the AI model. We propose to develop a novel human-in-the-loop workflow in which the AI model can identify the parts of the query that are difficult to generate, and prompt the user to provide

additional information.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Poess, M. and Floyd, C. (2000) New TPC Benchmarks for Decision Support and Web Commerce. *ACM SIGMOD Record*, **29**, 64-71. https://doi.org/10.1145/369275.369291

[2] Li, J.Y., Hui, B.Y., Qu, G., Yang, J.X., Li, B.H., Li, B.W., Wang, B.L., Qin, B.W., Geng, R.Y. and Huo, N. (2024) Can LLM Already Serve as a Database Interface? A Big Bench for Large-Scale Database Grounded Text-to-SQLs. arXiv: 2305.03111.

[3] Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., *et al.* (2018) Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-To-SQL Task. *Proceedings of the* 2018 *Conference on Empirical Methods in Natural Language Processing*, Brussels, 31 October-4 November 2018, 3911-3921. https://doi.org/10.18653/v1/d18-1425

[4] Naveed, H., Khan, A.U., Qiu, S., Saqib, M., Anwar, S., Usman, M., Barnes, N. and Mian, A. (2023) A Comprehensive Overview of Large Language Models. arXiv: 2307.06435.

[5] Bae, S., Kyung, D., Ryu, J., Cho, E., Lee, G., Kweon, S., Oh, J., Ji, L., Chang, E., Kim, T., *et al.* (2024) EHRXQA: A Multi-Modal Question Answering Dataset for Electronic Health Records with Chest X-Ray Images. arXiv: 2310.18652.

[6] Owda, M., Bandar, Z. and Crockett, K. (2011) Information Extraction for SQL Query Generation in the Conversation-Based Interfaces to Relational Databases (C-BIRD). In: O'Shea, J., Nguyen, N.T., Crockett, K., Howlett, R.J. and Jain, L.C., Eds., *Agent and Multi-Agent Systems: Technologies and Applications. KES-AMSTA* 2011, Springer, 44-53. https://doi.org/10.1007/978-3-642-22000-5_6

[7] Li, Q., Li, L., Li, Q. and Zhong, J. (2020) A Comprehensive Exploration on Spider with Fuzzy Decision Text-To-SQL Model. *IEEE Transactions on Industrial Informatics*, **16**, 2542-2550. https://doi.org/10.1109/tii.2019.2952929

[8] Pourreza, M. and Rafiei, D. (2024) DIN-SQL: Decomposed in-Context Learning of Text-to-SQL with Self-Correction. arXiv: 2304.11015.

[9] Chen, X.J., Wang, T., Qiu, T.H., Qin, J.B. and Yang, M. (2024) Open-SQL Framework: Enhancing Text-to-SQL on Open-Source Large Language Models. arXiv: 2405.06674.

[10] Li, H., Zhang, J., Li, C. and Chen, H. (2023) RESDSQL: Decoupling Schema Linking and Skeleton Parsing for Text-to-SQL. *Proceedings of the AAAI Conference on Artificial Intelligence*, **37**, 13067-13075. https://doi.org/10.1609/aaai.v37i11.26535

[11] Muniswamaiah, M., Agerwala, T. and Tappert, C.C. (2019) Federated Query Processing for Big Data in Data Science. 2019 *IEEE International Conference on Big Data* (*Big Data*), Los Angeles, 9-12 December 2019, 6145-6147. https://doi.org/10.1109/bigdata47090.2019.9005530

[12] Taipalus, T. (2020) The Effects of Database Complexity on SQL Query Formulation. *Journal of Systems and Software*, **165**, Article ID: 110576. https://doi.org/10.1016/j.jss.2020.110576

[13] Peeperkorn, M., Kouwenhoven, T., Brown, D. and Jordanous, A. (2024) Is Temperature the Creativity Parameter of Large Language Models? arXiv: 2405.00492.