UDC:
004.8:811.163.41'322.2
004.655.3
10.62907/juuntics260101001s

# Generating SQL Queries In The Serbian Language: A Comparative Analysis of Open-Source and Commercial Language Models

*Slaviša Sovilj[1*], Jovo Marković[1], Igor Dugonjić[2]*

[1]Ministry of the Interior of the Republic of Srpska, Banja Luka, Bosnia and Herzegovina; e-mail: slavisa.sovilj@proton.me; jovo.markovic@fbn.unibl.org
[2]Pan-European University "APEIRON", Faculty of Information Technology, Banja Luka, Bosnia and Herzegovina; e-mail: igor.r.dugonjic@apeiron-edu.eu



**Abstract:** This paper investigates the ability of large language models (LLMs) to generate SQL queries based on questions formulated in natural language (Text-to-SQL), with a particular focus on the Serbian language. The aim of the research is to empirically determine the extent to which the query language and task complexity affect the accuracy of the generated SQL code, as well as to identify the dominant sources of errors in the generation process. For this purpose, an experimental environment was developed, comprising a relational database that simulates an employee management system, an evaluation set of 200 tasks divided into four categories according to complexity level, natural-language formulations that the LLM translates into SQL code, and an automated system for executing evaluations in the selected language. The results suggest that the key challenge of Text-to-SQL systems is no longer syntactic validity, but semantic precision and the correct modeling of relationships between entities in the database, while the choice of natural language in which the question is formulated has a negligible impact on the final accuracy.

***Keywords:*** *Large language models (LLMs), Text-to-SQL.*

## Introduction

The development of large language models (LLMs) has significantly improved the capabilities of automatic natural language processing, including tasks that require the generation of formal languages such as SQL. Such systems have the potential to significantly reduce the need for technical knowledge of SQL and enable a wider range of users to access complex data [8]. Although modern models have demonstrated a high level of performance, there are still significant limitations in their practical application. Most existing studies focus on the English language and commercial models, while the performance of open-source, locally executable models in other linguistic contexts remains insufficiently explored. Furthermore, despite the high syntactic correctness of generated queries, semantic accuracy remains a key challenge [11].

This paper aims to empirically evaluate the ability of LLMs to generate SQL queries from questions formulated in the Serbian language, with an analysis of the impact of language and task complexity on the accuracy of the generated code. For this purpose, an experimental environment was developed that includes a relational database from the domain of employee management, as well as an evaluation set of 200 tasks divided into four categories of complexity. The experimental part of the paper is based on a comparison of the performance of several models, including both open-source and commercial models. The evaluation includes several levels, such as syntactic correctness, execution success, and result accuracy, with an additional analysis of error types and partial accuracy. Although the results provide significant insights, the study is limited to one domain and a relatively small evaluation set.

The research results indicate that the impact of language on model performance is limited, while the dominant causes of errors arise from problems of semantic understanding and modeling of relationships between database entities. Errors in query logic and SQL dialect incompatibility in open-source models are particularly emphasized. The structure of the paper is organized as follows: after the review of relevant literature, the research methodology and experimental framework are presented, followed by the results

*Corresponding author: slavisa.sovilj@proton.me

and their analysis, discussion, and conclusion.

The contribution of this paper is reflected in several aspects. First, the paper presents an empirical evaluation of the performance of large language models in Text-to-SQL tasks in the Serbian language, which has been insufficiently investigated in the existing literature. Second, an evaluation set of 200 tasks of different complexity levels was developed, based on a realistic relational database from the employee management domain. All code required for reproducing the research is available in an open repository [10]. Third, a comparative analysis of open-source and commercial models was conducted using several metrics, including syntactic correctness, execution success, and result accuracy. Finally, the paper provides a detailed analysis of error types, with a special focus on problems of semantic interpretation and SQL dialects.

## Literature review

The first phase in the development of Text-to-SQL systems was characterized by rule-based approaches, which rely on predefined patterns and heuristics for mapping natural language into SQL queries. The next significant step in the evolution of this field was the introduction of methods based on neural networks. In this context, the Seq2SQL model was proposed, using deep neural networks to learn relationships between natural language and SQL structures, thereby enabling automatic query generation without explicitly defined rules [1].

IRNet is a model that introduces the intermediate representation SemQL between natural language and SQL [3]. Instead of directly translating questions into SQL, the model first builds an abstract query structure that bridges the semantic gap between the user question and the target SQL code. It uses a bidirectional recurrent architecture with an attention mechanism to simultaneously understand the question and the database structure, achieving 46.7% accuracy on the Spider [2] dataset.

RAT-SQL [4] then introduces a relation-aware self-attention mechanism that uniformly encodes relational information between the question and the database schema, reaching an accuracy of 65.6%. LGESQL and SADGA [6] further improve this approach by applying graph neural networks to model structural relationships between questions and schemas. This approach enables more precise understanding of complex relational structures, and both models achieve accuracy above 75% on the Spider dataset, confirming the advantage of this approach over sequential architectures.

The emergence of large language models introduces a new paradigm in which the ability to generate SQL queries becomes an emergent capability of models trained on large corpora of text and code. DIN-SQL [7] decomposes the task into phases of schema linking, query classification, and SQL generation with self-correction, achieving 85.3% execution accuracy using GPT-4. DAIL-SQL [8] systematically compares prompting strategies and achieves 86.6% execution accuracy on the Spider dataset.

In parallel with model development, evaluation datasets were also developed. WikiSQL [11] opened the way for systematic model comparison, but it is limited to simple queries over single-dimensional tables. Spider [2] introduced cross-domain evaluation with 10,181 question-SQL pairs across 200 databases, requiring models to generalize to unseen databases. BIRD [9] introduces more realistic scenarios with large databases and questions requiring external knowledge, where the gap between automated systems and experts remains significant. Spider 2.0 [10] introduces evaluation based on enterprise scenarios and shows a dramatic drop in performance to only 6% accuracy, revealing model limitations under real-world conditions.

Although significant research exists in the field of Text-to-SQL, most studies focus on the English language and commercial models. The performance of open-source, locally executable models in other languages, including Serbian, remains insufficiently explored.

## Methodology

In this paper, an experimental approach was applied in order to evaluate the ability of LLMs to generate SQL queries from natural language. The methodology is based on a comparative analysis of execution results obtained by generating SQL queries and comparing them with a reference set of results. The primary performance measure is the execution accuracy of the generated SQL code (Execution Accuracy — EA), defined as the degree of correspondence between the results of the generated and reference SQL queries [2]:

$$\mathrm{Acc} = \frac{1}{N}\sum_{i=1}^{N}(Ri = \hat{R}i) \qquad (1)$$

where:

- N — total number of evaluated SQL queries (N = 200 per model and strategy),
- Ri — the result set obtained by executing the i-th generated SQL query over the database,
- Ri — the result set obtained by executing the i-th reference SQL query over the database,
- 1(Ri = Ri) denotes an indicator function that takes the value 1 if the result sets are identical, and 0 otherwise.

In order to analyze the causes of failure, an error taxonomy based on three categories was introduced. A dialect error denotes the use of SQL functions characteristic of the MySQL or SQLite dialect in a PostgreSQL context. A projection error occurs when the generated query returns the correct number of rows but the wrong set of columns or content. A logic error denotes semantically incorrect filtering logic, table joining, or aggregation, resulting in an incorrect number or order of returned rows.

Since EA can be modeled as a Bernoulli variable, confidence intervals were calculated using the binomial approximation to the normal distribution according to the formulas:

$$SE = \sqrt{\frac{p(1-p)}{N}} \qquad (2)$$

and

$$CI = p \pm z * \sqrt{\frac{p(1-p)}{N}} \qquad (3)$$

where p is the proportion of correct queries, N is the total number of queries per model and strategy (N = 200), and z = 1.96 corresponds to a 95% confidence level.

The evaluation was conducted on a set of 200 natural-language tasks of different complexity levels, enabling analysis of the impact of task complexity submitted to the LLM on the generated SQL code.

Additionally, the impact of the assigned task language was examined by comparing the results for tasks formulated in Serbian and English. This methodological framework enables a systematic and reproducible assessment of LLM performance, as well as the identification of key limitations in the process of SQL query generation.

## Research

In order to systematically evaluate the ability of large language models to generate SQL queries from natural language, an integrated experimental environment was developed, consisting of four components: a relational database, an evaluation task set, a module for executing reference queries, and a module for evaluating LLM models.

The relational database was implemented in PostgreSQL and models an information system for human resource management. The database includes eleven interconnected tables covering the domains of employee management, organizational structure, payroll system, leave requests, benefits, and employee development. The data were generated synthetically while preserving realistic statistical distributions and referential integrity, enabling queries of different semantic and structural complexity to be posed.

The evaluation set consists of 200 tasks organized into four categories according to SQL complexity level, with 50 tasks per category:

- simple queries (S),
- medium-complexity queries with multiple table joins (M),
- queries using window functions (W), and
- recursive queries over hierarchical structures (R).

Each task contains a question formulated in Serbian, the corresponding English translation, and a reference SQL query validated by execution over the database. The evaluation set was constructed by two database administrators with the aim of reflecting realistic information needs in a business environment. The process of generating SQL queries is formalized as an approximation of the function:

$$f: Q \times D \rightarrow S \qquad (4)$$

where:
- Q — the space of questions formulated in natural language,
- D — the space of relational database schemas,
- S — the space of syntactically valid SQL expressions.

For each task, the model receives a prompt containing the database schema and the question formulated in the selected language, and then generates an SQL query that is executed over the database. The result is compared with the reference solution by comparing the MD5 checksum of the returned rows and columns.

The evaluation was conducted using two generation strategies:
- generation from the Serbian question,
- generation from the English question.

Each strategy includes 200 queries per model, meaning that each model was expected to generate 400 queries. Five models of different architectures and sizes were tested: three open-source models available for local execution and two commercial models:

1. **DeepSeek Coder 1.3B**,open-source
2. **DeepSeek Coder 6.7B**,open-source
3. **Mistral 7B**, open-source
4. **GPT-4o-mini**, commercial
5. **GPT-4o**, commercial.

Since a total of five models were tested with 400 queries each, the research sample consisted of 2,000 queries distributed across different natural languages and difficulty levels. The complete experimental environment was implemented in the Python programming language using containerization technology, thereby ensuring reproducibility of the experimental results, with the entire code available in a public repository [12]

## Research results

### Syntactic Correctness of the Generated Code

Checking the syntactic correctness of generated SQL queries represents the first level of evaluation. However, syntactic correctness does not guarantee semantic accuracy; a model may generate a syntactically correct query that does not return the expected results. The results shown in Table 1 indicate significant differences between models, but not as much between languages.

| Model | EN(%) | SR(%) | Difference |
|---|---|---|---|
| DSC 1.3B | 79.5% | 58.0% | 21.5% |
| DSC 6.7B | 80.0% | 74.0% | 6.0% |
| Mistral 7B | 60.5% | 57.0% | 3.5% |
| GPT-4o-mini | 100.0% | 99.0% | 1.0% |
| GPT-4o | 100.0% | 99.0% | 1.0% |

Table 1: Overview of query syntactic correctness by model

Commercial models GPT-4o and GPT-4o-mini achieve almost perfect syntactic correctness in both languages, while open-source models show significantly lower values.

The dominant cause of syntactic errors in open-source models is the mixing of SQL dialects. The models insert functions characteristic of MySQL or SQLite, such as DATEDIFF, DATEADD, YEAR(), MONTH(), and julianday, despite the explicit instruction in the prompt to use PostgreSQL syntax exclusively. Table 2 shows the share of errors caused by incorrect SQL dialect and the total errors in generating syntactically correct code.

| Model | Language | Gd (%) | Gu (%) |
|---|---|---|---|
| DSC 1.3B | SR | 24.3% | 14.0% |
| DSC 1.3B | EN | 17.6% | 12.0% |
| DSC 6.7B | SR | 15.2% | 7.0% |
| DSC 6.7B | EN | 24.0% | 14.5% |
| Mistral 7B | SR | 41.1% | 30.0% |
| Mistral 7B | EN | 41.8% | 30.5% |
| GPT-4o-mini | SR | 0.0% | 0.0% |
| GPT-4o-mini | EN | 0.0% | 0.0% |
| GPT-4o | SR | 0.0% | 0.0% |
| GPT-4o | EN | 0.0% | 0.0% |

Table 2: Overview of dialect errors in relation to total errors by models and execution strategies

In the table header, Gu denotes total errors, while Gd denotes dialect errors. It is particularly significant that in the case of Mistral 7B, every second execution error, approximately 41%, originates from the wrong SQL dialect, equally in Serbian and English. This confirms that it is a systemic limitation of the model independent of the query language, indicating that this is a technical limitation rather than a linguistic factor, and represents a potential direction for improvement through model specialization. In contrast, in commercial models GPT-4o and GPT-4o-mini, dialect errors are completely absent, indicating that these models have acquired robust understanding of PostgreSQL syntax during training, independently of the question language.

**Execution Success of the Generated Code**

Executing the generated SQL query over the database represents the second level of evaluation. A query may be syntactically correct but fail to execute due to semantic errors, incorrect column names, nonexistent tables, or invalid GROUP BY clauses. The following table shows the levels of successfully executed SQL queries.

| **Model** | **EN(%)** | **SR(%)** | **Difference** |
|---|---|---|---|
| DSC 1.3B | 42.5% | 32.0% | +10.5 % |
| DSC 6.7B | 54.0% | 39.5% | +14.5 % |
| Mistral 7B, | 27.0% | 27.0% | 0.0% |
| GPT-4o-mini | 92.5% | 92.5% | 0.0 % |
| GPT-4o | 94.0% | 94.0% | 0.0 % |

Table 3: Overview of successfully executed SQL queries by model

Commercial models execute over 92% of generated queries in both languages. Open-source models show significantly lower values, with Mistral 7B achieving only 27% successful executions. The significant gap between syntactic correctness and execution success in open-source models — for example, in DeepSeek Coder 1.3B the difference is 10.5 percentage points in English — indicates that the models generate syntactically valid queries that are nevertheless not semantically correct.

**Execution Accuracy**

Execution accuracy is the central and strictest evaluation metric. A generated SQL query is considered correct only if the set of returned rows and columns is identical to the reference result, which is verified by comparing the MD5 checksum of the results. This measure is stricter than checking the number of rows because it also requires correct column projection and order.

| Model | EN(%) | SR(%) | Difference |
|---|---|---|---|
| DSC 1.3B | 2.0% | 2.0% | 0.0 % |
| DSC 6.7B | 2.5% | 1.5% | 1.0 % |
| Mistral 7B, | 1.0% | 0.0% | 1.0 % |
| GPT-4o-mini | 6.0% | 5.9% | 0.1 % |
| GPT-4o | 6.7% | 6.3% | 0.2 % |

Table 4: Overview of the EA metric for results obtained by query execution

EA values are extremely low across all models, regardless of architecture or size, ranging from 0% to 6.7%. This indicates that high syntactic correctness and execution success do not guarantee the semantic correctness of the generated query. The central finding of the research is that there is no statistically significant difference in EA between Serbian and English — differences from 0 to 1 percentage point do not show a consistent pattern that would indicate a systematic advantage of English, confirming that the language barrier is not the dominant cause of low accuracy.

In order to assess the stability of the obtained results, 95% confidence intervals were calculated for each execution accuracy value. The results show that the confidence intervals between Serbian and English overlap significantly for all tested models, which further confirms that the differences in accuracy are not statistically significant.

| Model | Language | EA (%) | 95% CI |
|---|---|---|---|
| GPT-4o | EN | 6.7 | 3.2 – 10.2 |
| GPT-4o | SR | 6.3 | 2.9 – 9.7 |
| GPT-4o-mini | EN | 6.0 | 2.7 – 9.3 |
| GPT-4o-mini | SR | 5.9 | 2.6 – 9.2 |

Table 5: Assessment of execution accuracy with 95% confidence intervals

For example, for the GPT-4o model, execution accuracy is 6.7% with a 95% CI of 3.2%–10.2% in English, and 6.3% with a 95% CI of 2.9%–9.7% in Serbian. Given the significant overlap of the intervals, it cannot be concluded that there is a statistically significant difference between these two cases.

**Partial Accuracy — Analysis of the Number of Rows and Columns**

Since the strict EA checksum does not distinguish between query logic errors and column projection errors, two additional measures were introduced into the analysis:

- row count match, and
- column count match.

These measures enable a better understanding of the causes of failure. Out of a total of 2,000 evaluated queries, 1,190, or 59.5%, were successfully executed over the database, with commercial models achieving over 92% execution success, while open-source models achieved between 27% and 54%.

| Metric | Number of queries | | % of executed |
|---|---|---|---|
| Successfully executed | 1,190 | | 59.5% |
| Correct number of rows / (Row match) | 501 | | 15.5% |
| Correct columns (Col match) | 789 | | 24.4% |
| Correct checksum (EA = 1) | 66 | | 2.0% |

Table 6: Overview of additional accuracy metrics

When examining the EA metric in relation to complexity level, the results are predictable — for less complex tasks, LLMs generate more accurate code. Regarding the types of errors that occur, three categories are distinguished:

Gd — dialect error, which implies the use of MySQL/SQLite functions in a PostgreSQL context,

Gp — projection error, where the model returns the correct number of rows but the wrong columns or content,

Gl — logic error, representing the most significant category of errors, because they directly affect the accuracy of the results returned by the SQL query. Unlike syntactic or dialect errors, such queries are often executable, but they generate the wrong number of rows due to incorrect filtering conditions, table joins, or aggregation.

The following table shows the distribution of errors by model

| **Model** | **Gd (%)** | **Gp(%)** | **Gl(%)** |
|---|---|---|---|
| DSC 1.3B | 11.8% | 8.1% | 28.9% |
| DSC 6.7B | 10.7% | 7.5% | 37.7% |
| Mistral 7B, | 30.1% | 0.6% | 25.7% |
| GPT-4o-mini | 0.0% | 11.1% | 80.1% |
| GPT-4o | 0.0% | 12.9% | 80.0% |

Table 7: Overview of the distribution of error types by model

**Example of a Dialect Error**

In the example with the DeepSeek Coder 1.3B model, for the question “Show employees without leave in the last 2 years”, the model generates the following query :

```
SELECT e.id,
       e.first_name,
       e.last_name
FROM employees AS e
WHERE NOT EXISTS
    (SELECT *
     FROM leave_requests lr
     WHERE lr.employee_id = e.id)
  AND TIMESTAMPDIFF('year', e.hire_date, CURRENT_DATE) > 2;
```

However, the TIMESTAMPDIFF function is not part of the PostgreSQL dialect, but belongs to the MySQL/MariaDB environment, which makes the query non-executable. The reference solution uses the correct PostgreSQL expression:

```
SELECT e.first_name,
       e.last_name
FROM employees e
WHERE NOT EXISTS
    (SELECT 1
```

```
    FROM leave_requests lr
    WHERE lr.employee_id = e.id
      AND lr.start_date >= CURRENT_DATE - interval '2 years');
```

This example clearly illustrates the problem of SQL dialect incompatibility as a significant source of errors.

**Example of a Projection Error**

In the example with the DeepSeek Coder 1.3B model, for the question "Show employees and their current salaries", the reference query returns first name, last name, and salary amount:

```
SELECT e.id,
    d.name AS department_name
FROM employees e
JOIN departments d ON e.department_id = d.id
WHERE is_active = TRUE;
```

Instead of salary information, data about departments are returned, which indicates an incorrect interpretation of the projection. The reference query that corresponds to the question requirement is:

```
SELECT e.first_name,
    e.last_name,
    s.amount
FROM employees e
JOIN salaries s ON s.employee_id = e.id
WHERE s.effective_to IS NULL
  AND e.is_active = TRUE
ORDER BY s.amount DESC;
```

Another example of this type is an error of incomplete aggregation of results. For the question "Show total benefits by type", the DeepSeek Coder 1.3B model generates the following SQL query:

```
SELECT benefit_type,
    SUM(amount) AS total_benefits
FROM benefits
GROUP BY benefit_type;
```

In this case, the model correctly identifies the need for aggregation and grouping, but omits an additional column implicitly required by the question, leaving the result incomplete.

The reference query is:

```
SELECT benefit_type,
    SUM(amount) AS ukupno,
    COUNT(*) AS broj
FROM benefits
GROUP BY benefit_type
ORDER BY ukupno DESC;
```

**Examples of Logic Errors**

For the question "Which active employees do not receive a transportation allowance?", the GPT-4o model generates the following SQL query:

```
SELECT e.id,
    e.first_name,
    e.last_name,
    e.email
FROM employees e
LEFT JOIN benefits b ON e.id = b.employee_id
AND b.benefit_type = 'transport'
WHERE e.is_active = TRUE
  AND b.id IS NULL;
```

At first glance, the generated query uses the standard LEFT JOIN ... IS NULL pattern for finding records without corresponding relations. However, in this context, this approach is not semantically equivalent to the requested condition, because it may lead to row duplication or incorrect filtering in the presence of multiple related records.

The reference query uses the NOT EXISTS construction:

```
SELECT e.first_name,
      e.last_name
FROM employees e
WHERE e.is_active = TRUE
  AND NOT EXISTS
    (SELECT 1
     FROM benefits b
     WHERE b.employee_id = e.id
       AND b.benefit_type = 'transport');
```

This approach guarantees the correct semantics of eliminating all employees who have at least one transportation allowance record, without the risk of duplicates or incorrect interpretation of relations. This example illustrates that models often choose syntactically valid patterns, but fail to recognize their exact semantic applicability.

For the question "Show departments where all active employees had at least one positive review (>3)", the GPT-4o-mini model generates the following query:

```
SELECT d.id,
      d.name
FROM departments d
LEFT JOIN employees e ON e.department_id = d.id
LEFT JOIN performance_reviews pr ON e.id = pr.employee_id
AND pr.score > 3.0
WHERE pr.id IS NULL;
```

while the GPT-4o model generates the following query:

```
SELECT d.id,
      d.name
FROM departments d
JOIN employees e ON d.id = e.department_id
JOIN performance_reviews pr ON e.id = pr.employee_id
AND pr.score > 3
GROUP BY d.id
HAVING COUNT(e.id) = COUNT(pr.employee_id);
```

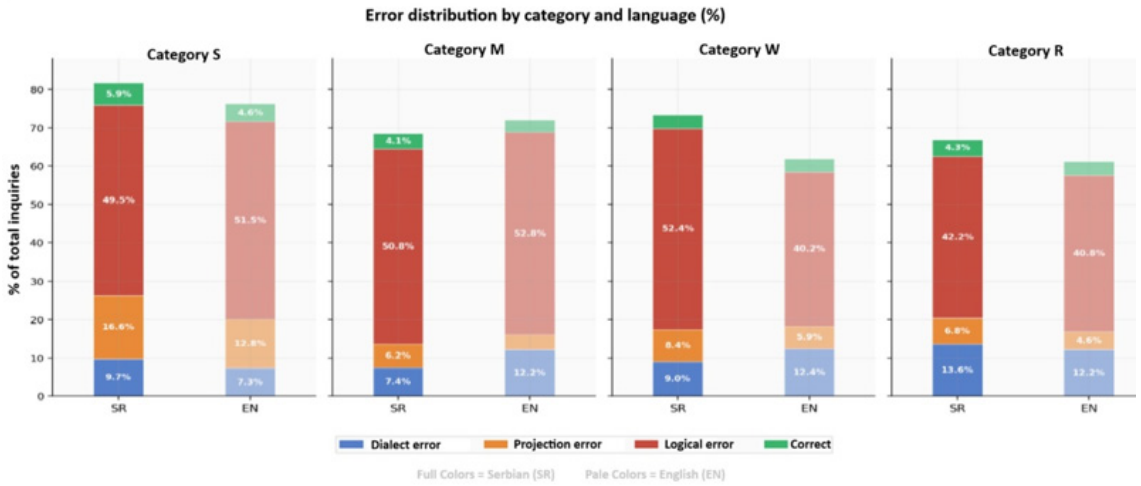


Figure 1. Distribution of errors by category and language

In both cases, the models attempt to solve a condition in which all employees must satisfy a certain criterion, using aggregation or LEFT JOIN patterns. However, such approaches do not properly cover all relevant cases, especially when employees have no records in the review table. As a result, the generated queries include records that should not be part of the result, leading to an incorrect number of returned rows.

The reference query uses the so-called double NOT EXISTS construction:

```
 SELECT d.name
FROM departments d
WHERE NOT EXISTS
   (SELECT 1
    FROM employees e
    WHERE e.department_id = d.id
     AND e.is_active = TRUE
     AND NOT EXISTS
       (SELECT 1
        FROM performance_reviews pr
        WHERE pr.employee_id = e.id
          AND pr.score > 3));
```

The analysis of error distribution reveals two distinctive model profiles. In open-source models, dialect errors dominate; for example, Mistral 7B records 30.1% dialect errors, while DeepSeek models show somewhat lower values, ranging from 10.7% to 11.8%, alongside a significant share of logic errors, indicating that semantic query understanding is an additional limitation of these models. In contrast, commercial models GPT-4o and GPT-4o-mini do not have a single dialect error, but record an extremely high share of logic errors of around 80%, confirming that these models have excellent command of PostgreSQL syntax but fail at the level of semantic understanding of the domain data model.

From the perspective of the impact of the Serbian language, dialect errors occur equally in both languages, confirming that this is a systemic limitation of the model independent of the query language, rather than a consequence of a language barrier.

Observed by complexity category and language, logic errors dominate in all combinations, ranging from 43% to 56%, confirming that semantic query understanding is the primary limitation of all tested models. Projection errors are most pronounced in the S category in Serbian, at 16.6%, which is explained by the fact that simple queries more often have an undefined requirement regarding columns — the model guesses the logic but not the exact projection. When Serbian and English are compared, there is no consistent degradation pattern that would indicate a language barrier. In the U, M, and W categories, dialect errors are higher in English than in Serbian, while in the R category the difference is negligible, 13.6% versus 12.2%. Overall, Serbian records a slightly lower share of dialect errors, 9.9%, compared to English, 11.1%, while projection errors are higher in Serbian, 9.4% versus 6.7%.

## DISCUSSION

The central finding of this paper is that the language barrier between Serbian and English is not the dominant cause of low accuracy in generated SQL queries. The difference in EA ranges between 0% and 6.7% across all tested models, which is a statistically negligible difference.

The introduction of confidence intervals further confirms that the observed differences between languages are not statistically significant, and that result variation primarily arises from model limitations rather than from the linguistic factor.

The analysis of errors by category and language does not show a consistent degradation pattern that would indicate a systematic advantage of English. The dominant cause of failure differs by model type and represents the second key finding of the study. In open-source models, the primary limitation is insufficient specialization for the PostgreSQL dialect — DeepSeek and Mistral systematically use MySQL and SQLite functions despite explicit instructions in the prompt, with Mistral recording as much as 30.1% dialect errors. This limitation is technical in nature and potentially solvable through prompt improvement or fine-tuning on PostgreSQL examples.

In commercial models GPT-4o and GPT-4o-mini, the situation is fundamentally different — dialect errors are zero, but logic errors account for about 80% of all failed queries. These models have excellent command of PostgreSQL syntax, but fail to generate semantically correct filtering, table joining, and aggregation logic in the context of a specific domain data model. Low EA across all models, ranging from 0% to 6.7%, does not mean that the models are unusable in practice, but rather that they could potentially be improved through prompting strategies.

Partial analysis shows that 15.5% of executed queries return the correct number of rows, and 24.4% return the correct number of columns. In a real business environment, such queries may represent a useful starting point that a domain expert can correct, which still significantly reduces the required technical knowledge of SQL compared to writing queries from scratch.

Viewed through the prism of SQL construct complexity, logic errors dominate in all categories, ranging from 43% to 56%, regardless of query language. This confirms that semantic understanding of the domain context represents the primary limitation of all tested models, both open-source and commercial. Recursive queries show a somewhat lower rate of logic errors, around 45%, compared with other categories, which is seemingly paradoxical, but can be explained by the fact that the model more often fails completely at earlier levels of evaluation — syntactic or dialect-related — and does not even reach the execution phase.

The limitations of the study include the use of only the HR domain, which limits the generalizability of the findings to other business contexts. The evaluation set of 200 tasks, although carefully constructed, represents a relatively small sample compared with established datasets such as Spider and BIRD.

## References

[1] V. Zhong, C. Xiong, and R. Socher, “Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning,” arXiv:1709.00103, 2017. Available: https://arxiv.org/abs/1709.00103
[2] Yu, T., Zhang, R., Yang, K. i dr. (2018). Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. Proceedings of EMNLP 2018. https://arxiv.org/abs/1809.08887
[3] Guo, J., Zhan, Z., Gao, Y. i dr. (2019). Towards complex text-to-SQL in cross-domain database with intermediate representation. Proceedings of ACL 2019. https://arxiv.org/abs/1905.08205
[4] Wang, B., Shin, R., Liu, X. i dr. (2020). RAT-SQL: Relation-aware schema encoding and linking for text- to-SQL parsers. Proceedings of ACL 2020. https://arxiv.org/abs/1911.04942
[5] Lei, F., Li, S., Liu, J. i dr. (2020). Re-examining the role of schema linking in text-to-SQL. Proceedings of EMNLP 2020. https://aclanthology.org/2020.emnlp-main.564
[6] Cao, R., Chen, L., Chen, Z. i dr. (2021). LGESQL: Line graph enhanced text-to-SQL model with mixed local and non-local relations. Proceedings of ACL 2021. https://arxiv.org/abs/2106.01093
[7] Pourreza, M. i Rafiei, D. (2023). DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. NeurIPS 2023. https://arxiv.org/abs/2304.11015
[8] Gao, D., Wang, H., Li, Y. i dr. (2024). Text-to-SQL empowered by large language models: A benchmark evaluation. PVLDB, 17(5), 1132–1145. https://arxiv.org/abs/2308.15363
[9] Li, J., Hui, B., Qu, G. i dr. (2024). Can LLM already serve as a database interface? NeurIPS 2024. https://arxiv.org/abs/2305.03111
[10] Lei, F. i dr. (2025). Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. ICLR 2025. https://arxiv.org/abs/2411.07763
[11] Z. Hong, Z. Yuan, Q. Zhang, H. Chen, J. Dong, F. Huang, and X. Huang, “Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL,” arXiv:2406.08426, 2024. Available: https://arxiv.org/abs/2406.08426
[12] http://github.com//adresa-repozitorija