Can LLM Already Serve as A Database Interface? A Big Bench for Large-Scale Database Grounded Text-to-SQLs

Anukriti Singh, Bharath Somasundaram, Praneeth Kumar Thummalapalli,
Shreya Sudhakaran
as18692@nyu.edu, bs4852@nyu.edu, pt2427@nyu.edu, ss18292@nyu.edu

Abstract

In this work, we undertake a study of text-to-SQL translation using the Spider dataset and three different Large Language Models (LLMs). The workflow is cut into three stages: Information Determination, Classification, and Hint, and SQL Generation. Both contribute to the end-to-end conversion of natural language instructions into SQL queries that can be executed. The method of evaluation uses the spider with the evaluation.py script that comes with the spider dataset. Unlike conventional string comparison methods, our evaluation approach decomposes SQL queries into clauses and conducts set comparison, enabling a more nuanced assessment of performance.

We have leveraged the scalability and parallel processing capability of Hadoop MapReduce to make the analysis and processing of this large-scale and cross-domain-based semantic parsing and text-to-SQL dataset efficient and effective. This was possible through the power of the distributed computing framework which Hadoop provides, in order to fully cater for the enormity and intricate nature that comes with the Spider dataset. By data partitioning, the data processing and analysis tasks enable effective extraction of insights and evaluation of text-to-SQL translation models This is intended to provide valuable insight into the effectiveness of LLMs for text-to-SQL translation and push towards the improvement in execution accuracy for future research.

Keywords

Spider dataset, Text-to-SQL translation, Large Language Models (LLMs), Hadoop, MapReduce, Semantic parsing, Cross-domain dataset, Natural language processing, Database management, Query generation, Evaluation methodology

Introduction

This report analyzes a project of advanced Text-to-SQL Translation Systems using an Architecture with Integration of three different Large Language Models (LLMs). All LLMs are essential to constitute a multi-stage process comprising Information Determination, Classification, Hint, and SQL Generation for translating natural language instructions into precise, formally structured, and concrete executable SQL queries. This study will test the performance of this integrated system by coming up with six different approaches to feed LLM and compare the results that use components such important in increasing accuracy and relevance in the context in which queries may be generated, such as Hints, Schema, Shots.

Architecture leverages the strong distributed computing capabilities of Hadoop MapReduce for parallelizing the call to our pipeline. The dataset contains more than 10,000 questions and over 5,693 unique SQL queries across 200 databases. Due to budget and time constraints, we have currently worked on 200 SQL queries across 4 databases. This is intended to allow the widest range of execution accuracy comparisons between the integrated LLM system and GPT-4.0 standalone. This comparison of LLMs points to the fact that their operational effectiveness is accurate and reliable, both in semantic parsing and database querying, able to give powerful insights into practical applications and limitations of LLMs, all while pointing the way for future advancements in this field.

Problem Statement

The text-to-SQL translation process suffers from a number of challenges. The process of translating natural language (NL) to structured query language (SQL) mainly in a complex, multi-table database environment, despite improving natural language processing is one. The present work attends to bringing about an intrinsic difficulty—a forefront issue seen in text-to-SQL translation, which inherently involves interpretations of the semantics of natural language instructions accurately mapped to executable SQL queries. It involves the following:

Understanding the schema: The understanding of the target database schema is one of the prime requisites in text-to-SQL translation, including table names and relations between tables, and constraints in it. This is required to produce contextually appropriate SQL queries reflecting the intent of the input in natural language.

Variability and Complexity in SQL: The SQL queries may range from very simple ones requiring only a single table, to highly complex ones referring to many tables, where nesting structures are followed, and functions and aggregations used are of an advanced nature. This level of variability puts great complexity into the translation process, so that very sophisticated mechanisms need to be developed to deal with the diversity of query structures.

Generalization Across Databases: The serious limitation, to be sure, of Large Language Models (LLMs) is that they have the problem of generalizing across databases. This highly limits the LLMs from being employed in scenarios where very robust performances are desired for varied domains and, in general, throttles their effectiveness for broader text-to-SQL applications.

The study will leverage Hadoop MapReduce distributed computing capabilities to parallelize the call to our pipeline. One objective is to develop methodologies that improve the translational accuracy and are also more scalable and generalizable for LLMs across varied database environments, thereby pushing the envelope further for semantic parsing and database querying.

Related Work

In [1], Yuanzhen Xie et al. propose a new method for text-to-SQL conversion that aims at substantially improving the model handling questions of a greater variety. This method tries to make the model handling questions of a greater variety more effective and efficient.

The information determination contains step filters: it identifies and focuses on the key needed data in building the SQL query, while the rest are filtered out. This is important in one way: to focus the concentration of a model to be deployed on a few relevant things within the question to be asked. The Classification Hint Module classifies the problems into different categories with respect to their needs: requirements for table joining or requirements for nested query execution and supplies relevant hints to lead the model along its best problem-solving path.

The most important part of our approach is SQL generation. Using templates and hints from previous steps, the given step should accurately generate an optimized SQL query.

The Self-Correction process further identifies and refines these queries to improve accuracy and efficiency, being sensitive to the point that the automation brings errors. Active Learning further improves the model's capability from learning from previous errors.

Finally, it prepares itself not only for the future challenge alterations of query complexity and performance feedback but indeed improves the current performance. This will allow for techniques to be integrated at the submodule level, therefore allowing increased precision in the adjustment and better detection of errors, in particular through the use of advanced algorithms for processing natural language. We do so with active learning, which integrates user feedback to ensure that the model keeps growing well-aligned with real-world application, and it keeps improving in accuracy and efficiency in generating the SQL query. All this offers a fine method for stronger immediate output and adaptability, of relevance towards evolving needs in data querying.

In [2], Shuaichen Chang and colleagues present the effective approaches for on-time engineering that will boost the performance of Large Language Models (LLMs) in converting the natural language of questions under different scenarios into SQL queries. We empirically investigate various prompt design methods for database representation and example set demonstration in the zero-shot, single-domain, and cross-domain settings. It underscores the importance of LLMs in-context learning ability for the text-to-SQL task—from prompt to database description—by comparing prompt designs and database portrayal with examples in each of the settings.

The research shows that there is variability in the use of various strategies to prompt construction, most specifically in the database schema and the content and examples displayed within it. This complicates comparing the two approaches in such a way that there is no uniformity in the construction of the prompt or even the effect it has on the LLM performance. This points out that, while presenting in various examples, helps reduce LLM within certain domains, presenting in various formats does not reduce the detailed representation of table contents.

Researchers Chang and Fosler-Lussier evaluated their model over the zero-shot, single-domain, and cross-domain settings with the Spider dataset, which is a cross-domain benchmark for the text-to-SQL task. They found that table relationships in the prompts and the way content was presented had a significant influence on LLM performance. It further points to an optimal prompt length that renders improved model performance in cross-domain scenarios, hence suggesting that maybe this "sweet spot" of 40 tokens is due to intrinsic characteristics in the LLM architecture or the training data.

This study, therefore, brings out insightful observations with reference to the development of effective prompts in the text-to-SQL task where the schema and content of the database to use demonstration examples. With this work, we effectively identified a few key determinants of impact on LLMs performance when generating SQL from text and thereby lay down the foundation for future work on how one might do better modification and apply LLMs to database querying.

In [3] Zhenwen Li and et.al development the technology of Large Language Models (LLMs) has real impacts on the advancement made in the text-to-SQL technologies, bringing new methodologies toward converting natural language instruction to an executable SQL query. This is an essential translation, so that users without special technical skills can carry out database querying. Good examples of such invaluable work in this domain are those contributed to by Li and Xie, aimed at finding the most appropriate SQL query from a candidate list created by the text-to-SQL models. They show an automatic generation of test case approach that uses LLM in predicting the expected execution result; hence, they benefit further when candidate SQL required to be queries written for execution by the underlying database are more. That is indeed a great stride in furthering the accuracy and usefulness in the text-to-SQL translations field, using the LLMs ability to understand and produce database queries.

Notably, the method presented by Li and Xie is of a high critic value to the present scope of research in this field of text-to-SQL parsing. It models the database and uses LLMs to find the ground truth or what is expected from the output, resulting from executing the given SQL query over this database. The novelty here lies in the fact that, hence, this approach tackles a relatively less-explored problem of test case generation for Text-to-SQL, which is paramount to the activity of benchmarking and improving the quality of Text-to-SQL systems. Through experiments on ways of generating easily predictable databases for LLMs and designing intuitive prompts, they lay a systematic vb foundation on how the performance of the text-to-SQL model can be improved through reranking based on test cases. The results of their experiments on the Spider validation dataset returned a really big jump in model performance, indicating their approach has great potential to push forward the broader goals of making data querying more intuitive and effective for a wide range of users.

Methods Architecture and Design

Methodology:

Spider dataset's 200 databases with multiple tables, encompassing 138 diverse domains are sent as input to the 3 LLMs to be processed which then return suitable SQL queries at the end. This process generates 6 SQL documents while using a combination of at least three of the following: Shots, Hints, Schema and Short Schema as follows:

Evaluation Metrics:

Our evaluation metrics include Exact Matching and Execution Accuracy. In addition, we measure the system's accuracy as a function of the difficulty of a query. Since our task definition does not predict value strings, our evaluation metrics do not take value strings into account. We have used the official evaluation script available on the spider dataset to measure the correctness of the algorithm.

Exact Matching:

We measure whether the predicted query as a whole is equivalent to the gold query. The predicted query is correct only if all of the components are correct. Because we conduct a set comparison in each clause, this exact matching metric can handle the "ordering issue".

Execution Accuracy:

Since Exact Matching is possible to provide false negative evaluation when the semantic parser is able to generate novel syntax structures, we also consider Execution Accuracy. For Execution Accuracy, the value is a must in order to execute SQL queries. Instead of generating these values, a list of gold values for each question is given. Models need to select them and fill them into the right slots in their predicted SQL. We exclude value prediction in Component and Exact Matching evaluations and do not provide Execution Accuracy in the current version. However, it is also important to note that Execution Accuracy can create false positive evaluations when a predicted SQL returns the same result (for example, 'NULL') as the gold SQL while they are semantically different. So we can use both to complement each other.

SQL Hardness Criteria:

To better understand the model performance on different queries, we divide SQL queries into 4 levels: easy, medium, hard, extra hard. We define the difficulty based on the number of SQL components, selections, and conditions, so that queries that contain more SQL keywords (GROUP BY, ORDER BY, INTERSECT, nested subqueries, column selections and aggregators, etc) are considered to be harder. For example, a query is considered as hard if it includes more than two SELECT columns, more than two WHERE conditions, and GROUP BY two columns, or contains EXCEPT or nested queries. A SQL with more additions on top of that is considered as extra hard.

SQL query examples in 4 hardness levels:

Easy:

easy pred:

```
SELECT COUNT(*) as Number_of_Farms FROM farm.
```

easy gold:

```
SELECT count(*) FROM farm
```

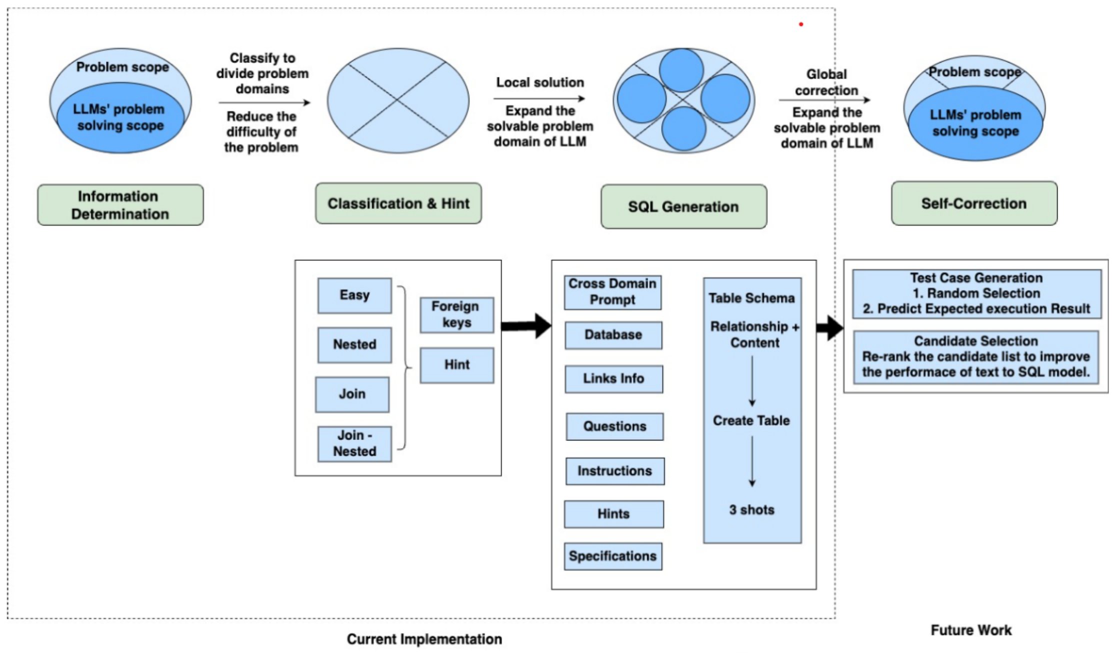
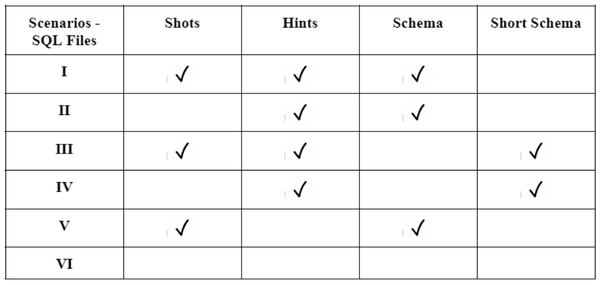


Figure 1: The overall structure of the SQL model



Scenarios - SQL Files	Shots	Hints	Schema	Short Schema
I	✓	✓	✓	
II		✓	✓	
III	✓	✓		✓
IV		✓		✓
V	✓		✓	
VI				

Figure 2: 6 SQL documents combinations

Medium:
medium pred:

```
SELECT SUM(duration) AS total_duration,
MAX(duration) AS max_duration FROM trip
WHERE bike_id = 636;
```

medium gold:

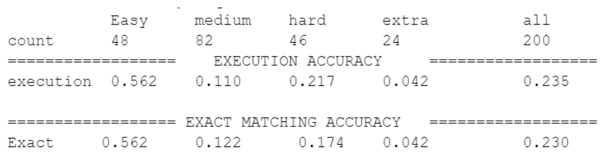
```
SELECT sum(duration) ,  max(duration)
FROM trip WHERE bike_id  =  636
```

Hard:
hard pred:

```
SELECT s.id FROM station s WHERE
s.lat > 37.4 AND NOT EXISTS
(SELECT 1 FROM status st WHERE
st.station_id = s.id AND
st.bikes_available < 7);
```

hard gold:

```
SELECT id FROM station WHERE
```



```
              Easy    medium    hard     extra          all
count         48      82        46       24             200
==================    EXECUTION ACCURACY     ==================
execution     0.562   0.110     0.217    0.042          0.235

================== EXACT MATCHING ACCURACY   ==================
Exact        0.562    0.122      0.174   0.042          0.230
```

Figure 3: Scenario - I: Generated by using shots, hint and schema

```
lat  >  37.4 EXCEPT SELECT
station_id FROM status GROUP
BY station_id HAVING
min(bikes_available)  <  7
```

Extra Hard:
extra pred:

```
SELECT s.name FROM station s
JOIN status st ON s.id = st.station_id
WHERE s.city != 'San Jose'
GROUP BY s.name HAVING
AVG(st.bikes_available) > 10;
```

extra gold:

```
SELECT T1.name FROM station AS T1
JOIN status AS T2 ON
T1.id  =  T2.station_id GROUP BY
T2.station_id
HAVING avg(bikes_available)  >  10
EXCEPT SELECT name FROM station
WHERE city  =  "San Jose"
```

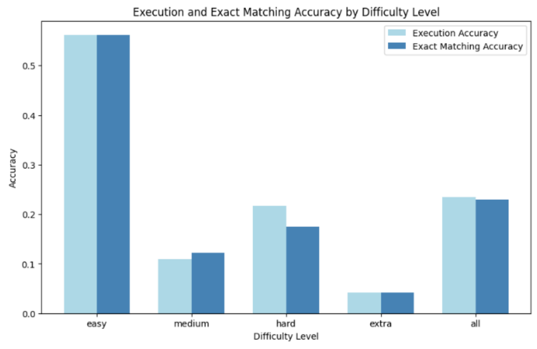


Figure 4: Scenario 1: Execution and Exact Matching Accuracy by difficulty level

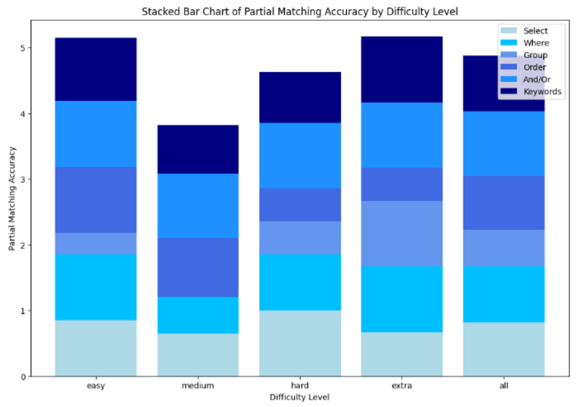


Figure 5: Scenario 1: Partial Matching accuracy by difficulty level

Result

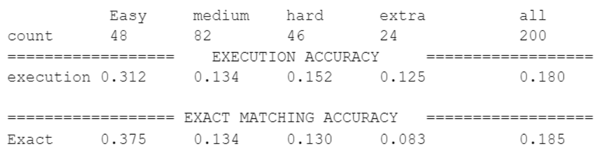
Scenario - I: Generated by using shots, hint and schema

Generated by using shots, hint, and schema:

- **Execution accuracy vs. Exact Match Accuracy with respect to the degree of difficulty of the queries:** Utilizing shots, hints, and schema, exhibits moderate execution accuracy ranging from 11.0% to 56.2% across different difficulty levels, with an exact matching accuracy ranging from 11.2% to 56.2%.
- Partial Accuracy with respect to the degree of difficulty of the queries and query groups like SELECT, WHERE, GROUP BY, etc.

Scenario - II: Generated by using hints and schema without any shots:

Generated by using hints and schema without any shots:
Execution accuracy v/s Exact Match Accuracy with respect to the degree of difficulty of the queries: Generated without shots but with hints and schema, shows lower execution accuracy



```
           Easy      medium    hard      extra          all
count      48        82        46        24             200
==================    EXECUTION ACCURACY     ==================
execution 0.312      0.134     0.152     0.125          0.180

================== EXACT MATCHING ACCURACY   ==================
Exact      0.375      0.134     0.130     0.083          0.185
```

Figure 6: Scenario - II: Generated by using hints and schema without any shots:

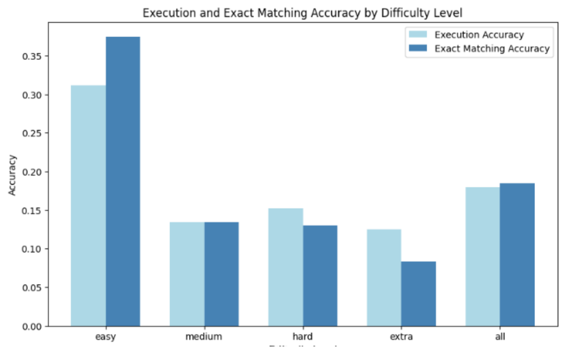


Figure 7: Scenario 2: Execution and Exact Matching Accuracy by difficulty level

(13.4% to 31.2%) and exact matching accuracy (13.0% to 37.5%). Partial Accuracy with respect to the degree of difficulty of the queries and query groups like SELECT, WHERE, GROUP BY, etc.

Scenario - III: Generated by using shots, hints and short schema:

Generated by using shots, hints and short schema:
Execution accuracy v/s Exact Match Accuracy with respect to the degree of difficulty of the queries:
Partial Accuracy with respect to the degree of difficulty of the queries and query groups like SELECT, WHERE, GROUP BY, etc.

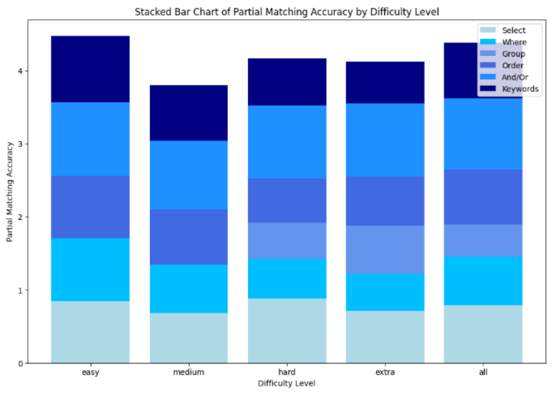
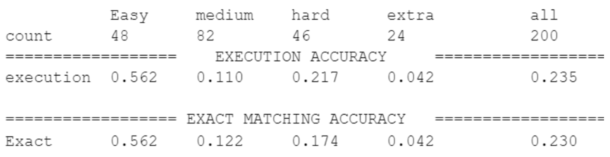


Figure 8: Scenario 2: Partial Matching accuracy by difficulty level



```
           Easy      medium    hard      extra          all
count      48        82        46        24             200
==================    EXECUTION ACCURACY     ==================
execution  0.562     0.110     0.217     0.042          0.235

================== EXACT MATCHING ACCURACY   ==================
Exact      0.562     0.122     0.174     0.042          0.230
```

Figure 9: Scenario - III: Generated by using shots, hints and short schema:

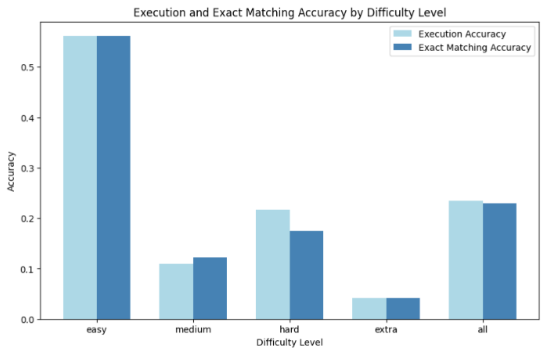


Figure 10: Scenario 3: Execution and Exact Matching Accuracy by difficulty level

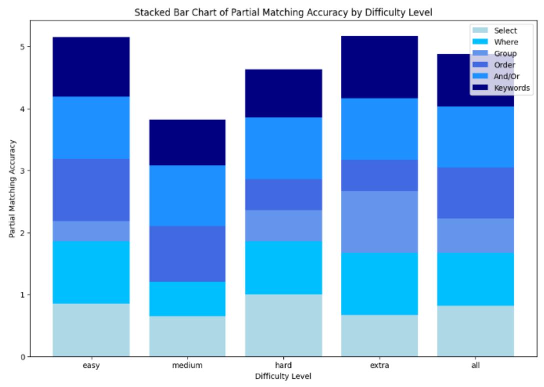


Figure 11: Scenario 3: Partial Matching accuracy by difficulty level

```
          Easy    medium   hard     extra          all
count     48      82       46       24             200
==================    EXECUTION ACCURACY     ==================
execution  0.604   0.183    0.217    0.042          0.275

================== EXACT MATCHING ACCURACY   ==================
Exact      0.604   0.146    0.196    0.042          0.255
```

Figure 12: Scenario - IV: Generated by using hints and short schema without any shots:

Scenario - IV: Generated by using hints and short schema without any shots:

Generated by using hints and short schema without any shots:

Execution accuracy v/s Exact Match Accuracy with respect to the degree of difficulty of the queries: Generated without shots but with hints and short schema, demonstrates improved execution accuracy (18.3% to 60.4%) and exact matching accuracy (14.6% to 60.4%).

Partial Accuracy with respect to the degree of difficulty of the queries and query groups like SELECT, WHERE, GROUP BY, etc.

Scenario - V: Generated by using shots and schema without any hints:

Generated by using shots and schema without any hints:

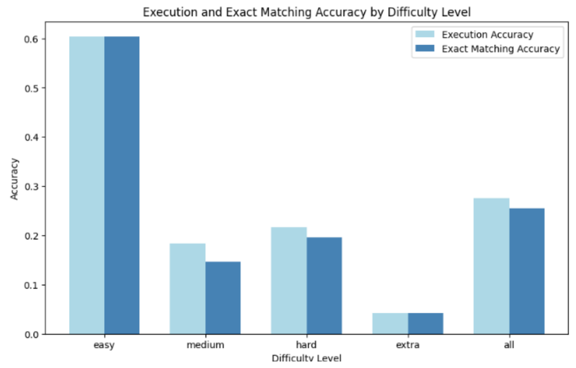


Figure 13: Scenario 4: Execution and Exact Matching Accuracy by difficulty level

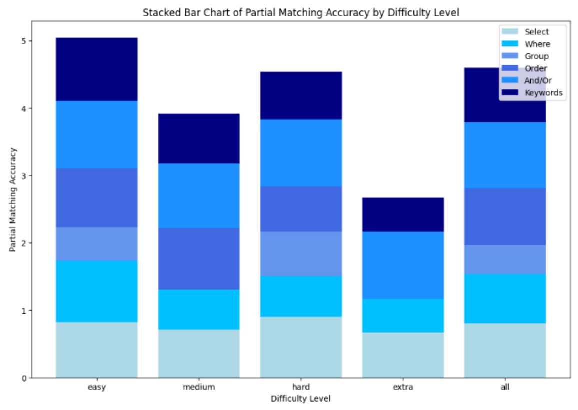
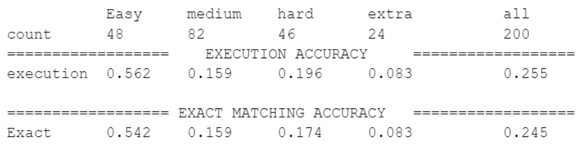


Figure 14: Scenario 4: Partial Matching accuracy by difficulty level



```
           Easy     medium     hard       extra          all
count      48       82         46         24             200
==================    EXECUTION ACCURACY      ==================
execution  0.562    0.159      0.196      0.083          0.255

================== EXACT MATCHING ACCURACY    ==================
Exact      0.542    0.159      0.174      0.083          0.245
```

Figure 15: Scenario - V: Generated by using shots and schema without any hints

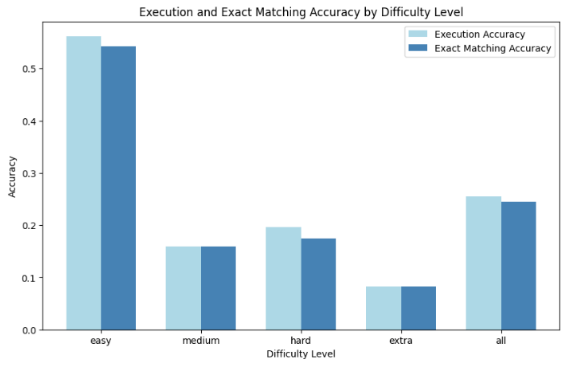


Figure 16: Scenario 5: Execution and Exact Matching Accuracy by difficulty level

Execution accuracy v/s Exact Match Accuracy with respect to the degree of difficulty of the queries: Without hints but with shots and schema shows execution accuracy ranging from 15.9% to 56.2% and exact matching accuracy ranging from 15.9% to 54.2%.
Partial Accuracy with respect to the degree of difficulty of the queries and query groups like SELECT, WHERE, GROUP BY, etc.

Scenario - VI:Generated using GPT-4 pipeline:

The results suggest that the inclusion of hints and a short schema, even without shots, can significantly enhance the execution and exact matching accuracies of the generated SQL queries, particularly evident in case 4. However, the presence of shots alongside hints and schema does not notably improve performance compared to scenarios utilizing hints and

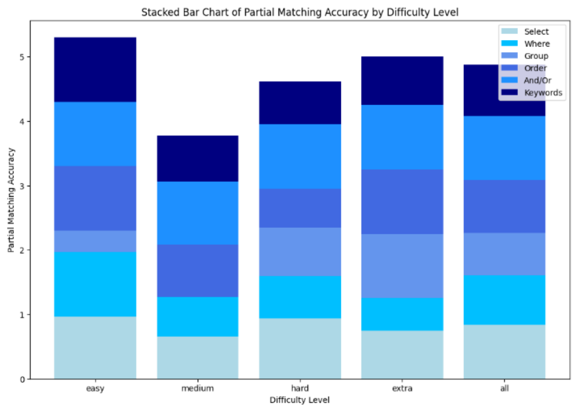
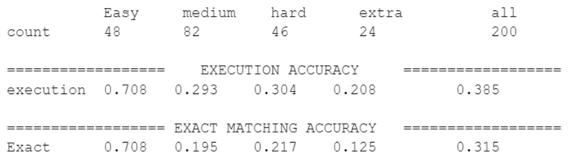


Figure 17: Scenario 5: Partial Matching accuracy by difficulty level



```
           Easy     medium     hard       extra          all
count      48       82         46         24             200

==================    EXECUTION ACCURACY      ==================
execution  0.708    0.293      0.304      0.208          0.385

================== EXACT MATCHING ACCURACY    ==================
Exact      0.708    0.195      0.217      0.125          0.315
```

Figure 18: Scenario - VI:Generated using GPT-4 pipeline

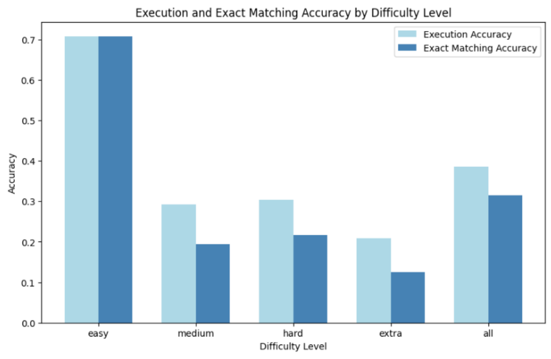


Figure 19: Scenario 6: Execution and Exact Matching Accuracy by difficulty level

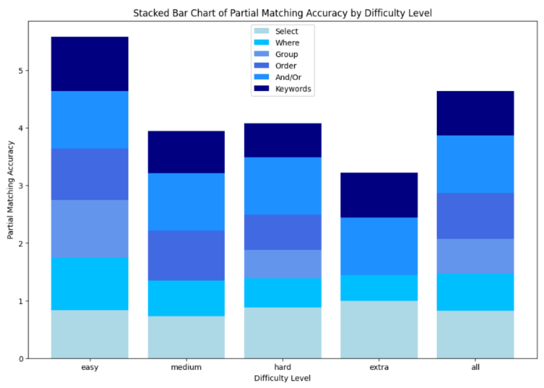


Figure 20: Scenario 6: Partial Matching accuracy by difficulty level

schema alone. This analysis underscores the importance of input methods and their combinations in improving the effectiveness of SQL query generation.

Conclusion:

This project allowed sophisticated design and development of an architecture that can effectively combine three Large Language Models (LLMs) in producing SQL queries. We evaluated the effectiveness of this integrated pipeline by comparing its performance against the standard results from the Spider dataset and against the outcomes that were derived from using only GPT-4.0. From these findings, we get the execution accuracy of 60.4% by the integrated approach. But, the use of GPT-4.0 had an execution accuracy of 70.8%. These observations, therefore, tend to imply that under such test conditions, though our model introduces complexity and can have potential robustness, the GPT-4.0 standalone provides much better accuracy in text to SQL query generation.

Future Work

Our research aims to realize the proposed framework by implementing and refining the Self-Correction module and integrating it seamlessly with Large Language Models (LLMs) for enhanced decision-making in text-to-SQL translation. The focus will be on the practical implementation of the Self-Correction module's functionality, starting with the generation of candidate SQL queries from natural language instructions. This will involve fine-tuning the text-to-SQL model to produce accurate and diverse candidate queries that capture the nuances of the given instructions and database schema. Subsequently, the test case generation and evaluation process will be optimized to ensure comprehensive coverage of potential scenarios, enabling the module to effectively assess the performance of candidate queries. Moreover, the integration of a feedback loop mechanism will be a key area of focus to enable continuous improvement and error correction within the text-to-SQL translation system. By running evaluation scripts for error detection and correction, the system will iteratively refine its performance, leading to enhanced accuracy and reliability over time. Additionally, exploring the scalability of the proposed framework and addressing potential resource constraints will be essential for its practical deployment in real-world scenarios. This may involve optimizations in computational efficiency and resource utilization to ensure the system's feasibility across various platforms and environments.

Our research will implement and refine the self-correction module and integrate it seamlessly with the large language model to aid in better decisions during the course of text-to-SQL translation. The main point will be practical implementation of the functionalities of the Self-Correction, which will range from generating candidate SQL queries out of the natural language instructions. This will be done by fine tuning the text-to-SQL model, producing an accurate and diverse set of candidate queries that capture nuances from the given set of instructions and schema. Following that, the process of test case generation and evaluation will be further fine-tuned to an extent where the module-based test is able to provide complete coverage of all the possible scenarios through which the performance of candidate queries can be tested. Attention will also be given to the feedback loop mechanism to be incorporated, which would enable continual improvement and error correction of the text-to-SQL translation system. The accuracy and reliability of the system will, therefore, be improved by running the scripts for evaluation of errors in detection and correction. For it to actually be of use in practical deployment, the scalability of the system based on resource considerations will also be investigated. This will particularly include optimal computational and resource efficiency through different settings of the platforms.

References

1. Chang S. & Fosler-Lussier E. (2023). *How to Prompt LLMs for Text-to-SQL: A Study in Zero-shot Single-domain and Cross-domain Settings.* https://arxiv.org/abs/2306.12345

2. Li Y. & Xie S. (2024). *Self-Correction - Testing and selecting the best SQL results from the candidates - Using LLM to select the right SQL Query from candidates.* https://arxiv.org/pdf/2305.11853.pdf

3. Li Y. & Xie S. (2024). *SQL Generation - How to Prompt LLMs for Text-to-SQL: A Study in Zero-shot Single-domain and Cross-domain Settings.* https://arxiv.org/pdf/2401.02115.pdf

4. Samuel Arcadinho, David Aparício, Hugo Veiga & António Alegria *T5QL: Taming language models for SQL generation.* https://arxiv.org/pdf/2209.10254.pdf

5. Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding & Jingren Zhou *Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation.* https://arxiv.org/pdf/2308.15363.pdf

6. Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding & Jingren Zhou *Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation.* https://arxiv.org/pdf/2308.15363.pdf

7. Han Wang, Archiki Prasad, Elias Stengel-Eskin, Xiuyu Sun, Yichen Qian & Mohit Bansal *Soft Self-Consistency Improves Language Model Agents.* https://arxiv.org/pdf/2402.13212v1.pdf

8. Xiang Lisa Li & Percy Liang *Prefix-Tuning: Optimizing Continuous Prompts for Generation.* https://arxiv.org/pdf/2101.00190.pdf

9. Yuanzhen Xie, Xinzhou Jin, Tao Xie, Mingxiong Lin, Liang Chen, Chenyun Yu, Lei Cheng, Chengxiang Zhuo, Bo Hu & Zang Li *Decomposition for Enhancing Attention: Improving LLM-based Text-to-SQL through Workflow Paradigm.* https://arxiv.org/pdf/2402.10671v1.pdf

10. Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer & Christopher Ré1 *LANGUAGE MODELS ENABLE SIMPLE SYSTEMS FOR GENERATING STRUCTURED VIEWS OF HETEROGENEOUS DATA LAKES.* https://arxiv.org/pdf/2304.09433.pdf

11. Zhenwen Li & Tao Xie *Using LLM to select the right SQL Query from candidates.* https://arxiv.org/pdf/2401.02115.pdf