

Enhancing Text-to-SQL Translation for Financial System Design

Yewei Song
University of Luxembourg
Luxembourg
yewei.song@uni.lu

Saad Ezzini
Lancaster University
United Kingdom
s.ezzini@lancaster.ac.uk

Xunzhu Tang
University of Luxembourg
Luxembourg
xunzhu.tang@uni.lu

Cedric Lothritz
University of Luxembourg
Luxembourg
cedric.lothritz@uni.lu

Jacques Klein
University of Luxembourg
Luxembourg
jacques.klein@uni.lu

Tegawendé Bissyandé
University of Luxembourg
Luxembourg
tegawende.bissyande@uni.lu

Andrey Boytsov
Banque BGL BNP Paribas
Luxembourg
andrey.boytsov@bgl.lu

Ulrick Ble
Banque BGL BNP Paribas
Luxembourg
ulrick.ble@bgl.lu

Anne Goujon
Banque BGL BNP Paribas
Luxembourg
anne.goujon@bgl.lu

ABSTRACT

Text-to-SQL, the task of translating natural language questions into SQL queries, is part of various business processes. Its automation, which is an emerging challenge, will empower software practitioners to seamlessly interact with relational databases using natural language, thereby bridging the gap between business needs and software capabilities.

In this paper, we consider Large Language Models (LLMs), which have achieved state of the art for various NLP tasks. Specifically, we benchmark Text-to-SQL performance, the evaluation methodologies, as well as input optimization (e.g., prompting). In light of the empirical observations that we have made, we propose two novel metrics that were designed to adequately measure the similarity between SQL queries.

Overall, we share with the community various findings, notably on how to select the right LLM on Text-to-SQL tasks. We further demonstrate that a tree-based edit distance constitutes a reliable metric for assessing the similarity between generated SQL queries and the oracle for benchmarking Text2SQL approaches. This metric is important as it relieves researchers from the need to perform computationally expensive experiments such as executing generated queries as done in prior works. Our work implements financial domain use cases and, therefore contributes to the advancement of Text2SQL systems and their practical adoption in this domain.

1 INTRODUCTION

Advances in natural language processing (NLP), notably with the advent of large language models (LLMs), have led to significant performance improvements in various Text-to-Code generation tasks. Among these, Text-to-SQL, i.e., the process of translating

natural language queries into SQL queries, is an emerging task, with various applications in businesses. Implementing reliable Text-to-SQL automation is expected to empower practitioners, i.e., users and non-tech agents, to seamlessly interact with business relational databases using their own natural language. Breaking through the challenge of translating text to SQL code will bridge an important gap between business needs and software capabilities, and unlock new possibilities for software integration.

Our work focuses on Text2SQL with a specific emphasis on the financial domain. Leveraging the promising capabilities of LLMs, which have garnered widespread adoption across various domains, including code generation, our research endeavors to showcase the

To this end, this paper contributes to the advancement of Text2SQL systems and their practical utility. We turn our attention to three primary areas: performance benchmarking, evaluation methodologies, and input optimization. Within these domains, we propose two novel metrics designed to measure SQL query similarity, demonstrating impressive efficacy. Additionally, we conduct rigorous benchmarking exercises to evaluate the performance of LLMs while also exploring the impact of rephrased questions and two distinct prompt types. Furthermore, we curate a specialized sub-dataset focused on bank operations, replete with complex questions that pose a formidable challenge to state-of-the-art models. In doing so, our work not only sheds light on the potential enhancements to Text2SQL systems but also offers valuable insights into the broader landscape of NLP applications in software engineering, especially in domains with complex and domain-specific requirements.

Contributions. In this paper, we present three primary contributions. Firstly, we perform an empirical study on a publicly available dataset for the Text2SQL task and benchmark several SOTA LLMs. We provide some insights into model selection in the financial domain. Secondly, we create a set of new and challenging test cases specifically tailored to transaction-related queries, filling a critical gap in the availability of realistic dataset resources for banking operations. Thirdly, we propose two innovative evaluation metrics designed to accurately assess the performance of Text2SQL models. This metric exhibits a high correlation with execution matching,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICSE 2024, April 2024, Lisbon, Portugal

© 2023 Association for Computing Machinery.

eliminating the need to execute code on the database and offering more efficient and practical means of model evaluation in the context of banking applications. Additionally, we propose question optimization methods that improve performance for Text2SQL models.

Findings. Our work holds 2 important findings for practice: First, we conclude that for Text-to-SQL applications in financial business scenarios, choosing LLM with more code training data has better performance, such as *nsql-6B* and *CodeGen2*. Second, we found that the new metric we proposed, TSED, is currently the best Text-to-SQL evaluation metric when the original database cannot be used for result evaluation.

2 BACKGROUND AND RELATED WORK

This section presents the necessary background for our solutions and further discusses the related literature in SE and NLP.

Large Language Models. LLMs, including models like GPT-4 and PaLM, represent a transformative advancement in the field of NLP. These models are characterized by their extensive scale, boasting millions to billions of parameters, enabling them to learn intricate linguistic patterns and relationships from vast amounts of text data. This capacity allows them to generate human-like text and perform a wide array of language-related tasks, from text generation and machine translation to sentiment analysis and question answering. Their pre-trained nature, combined with fine-tuning on specific tasks, has led to groundbreaking progress in NLP. As LLMs continue to evolve, they promise to redefine the landscape of NLP by fostering more nuanced understanding, interaction, and generation of text.

The development of LLMs finds a common ancestry in Transformer proposed in 2017 [31], as illustrated in Figure 1. These models have diversified due to differences in deep neural network structure and training strategies, with diverse optimization and supervised fine-tuning techniques like LoRA [8] and P-Tuning v2 [18] alleviating training complexities. The HuggingFace LLM leaderboard [9], for instance, featured approximately 75 models in May 2023, but by mid-August 2023, this number had surged to over 600, marking the "explosion of LLMs" in the field. A recurring trend in contemporary LLMs is the exponential growth in the number of model parameters, with models like GPT-4 (presumed) expanding their parameter count by up to 10^5 times when compared to the BERT model, as depicted in Figure 1. The question of whether this increase in model size translates to diminishing marginal returns in terms of performance remains an enigma, but some papers, such as LLaMA [29] and Chinchilla [7], have suggested that the pre-training corpus may need to scale proportionally to the number of parameters, a practice not always followed by modern LLMs.

The Text-to-SQL task, serving as a vital link between human language and structured databases, has gained prominence in the field of natural language processing. Some survey papers, including contributions by Qin [24] and Katsogiannis [10], introduce some of the latest advances up to the first half of 2023. It involves translating natural language queries into structured SQL queries for database execution, aligning with the imperative for AI systems to understand and process user queries in a human-like manner. This task has its roots in semantic parsing and question answering, initially

explored through rule-based and template-based approaches. However, the advent of deep learning, especially sequence-to-sequence models like Seq2Seq with attention mechanisms, marked a significant turning point, enabling more precise and adaptable conversion of natural language into SQL. Datasets such as WikiSQL and ATIS established standardized benchmarks for evaluating Text2SQL models [6, 34], while the SPIDER dataset [33], meticulously annotated by Yale students, further accelerated progress.

In the Text2SQL domain, cutting-edge models have emerged to address the challenge of converting natural language queries into SQL queries. PICARD, introduced by Scholak et al. [25], combines a beam-search-like processor with the T5 model to effectively understand and generate SQL queries. Some remarkable post-PICARD advancements in existing methodologies, exemplified by RESD-SQL [15] and Graphix-3B [16]. However, it remains essential to scrutinize the performance of LLMs, particularly in Zero-Shot settings. Anticipating ongoing optimization and fine-tuning on Text2SQL datasets, it is evident that this task continues to evolve and hold a significant position in the NLP landscape.

In the realm of natural language processing, several Text2SQL datasets have emerged as pivotal resources for advancing the capabilities of semantic parsing models. Datasets such as WikiSQL, ATIS, and the recently introduced CoSQL, each contribute unique challenges and linguistic nuances from various domains [3, 32, 34]. However, the SPIDER dataset stands out due to its expansive complexity and cross-domain nature, making it a preferred choice for research and development in this field [33]. Notably, the SPIDER dataset benefits from the involvement of 7 Yale students who meticulously annotated the data, enhancing its accuracy and quality.

In Text2SQL evaluation, two crucial metrics are employed from SPIDER paper [33]: Execution Match, which assesses query utility by executing it against a database; Exact Match, which measures both semantic and syntactic accuracy. Besides, the BLEU score, adapted from machine translation evaluation, gauges query fluency and relevance [21]. Additionally, the use of Abstract Syntax Trees (AST) aids in evaluating structural similarity [13], offering deeper insights into a system's ability to capture underlying query structures.

3 SCENARIO AND CHALLENGES

Together with our industrial partner from the financial domain, we aimed to enhance a question-answering chatbot. To that end, we have incorporated a Text2SQL module. This module serves as a crucial component, enabling smooth interactions and ensuring our chatbot's ability to accurately retrieve data from the relational database and construct precise responses. The Text2SQL module acts as an intermediary, converting natural language queries from users into SQL queries that can be directly executed on the bank's database. Then we need to add an evaluation part to the module to check if the results are correct or not. This allows the company can easily integrate it into its chatbot system and keep polishing it.

3.1 Text to SQL Pipeline

This paper aims to assess the performance of several LLMs on the Text2SQL task by considering practical constraints. These constraints lead to several challenges that are elicited in Section 3.2. In this section, we detail the Text2SQL pipeline used to assess the

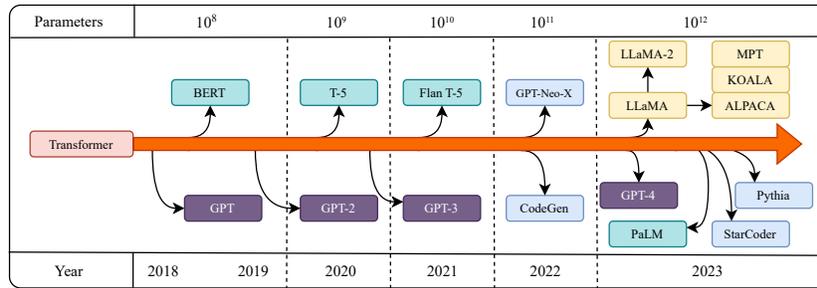


Figure 1: Development of Large Language Models from Year 2018-2023

various LLMs. The entire pipeline is shown in Figure 2. For several of our experiments, we rely on the SPIDER dataset [33], which provides databases, questions in natural language, and the corresponding "ground truth" SQL queries. For a given question, the goal of the Text2SQL model is to generate a SQL query that generates the same result as the SQL query from the Ground Truth. To assess the quality of the generated SQL queries, several metrics are computed (detailed in Section 5). Some of these metrics are computed directly from the code (i.e., we compare the generated SQL queries against the ground truth queries), and others are computed based on the result of the execution of the queries. The pipeline also includes some key steps inside the Text2SQL model:

- **Preprocessing:** The incoming natural language query is tokenized and transformed into numerical representations as embedding vectors.
- **Encoding:** The tokenized and numerical representation of the natural language query is fed into the trained Encoder of the Seq-to-Seq model.
- **Context Vector:** The Encoder processes the input query and generates a fixed-size context vector that captures the semantic information of the query.
- **Decoding:** The context vector is passed to the trained Decoder of the Seq-to-Seq model. The Decoder generates the SQL query token by token based on the information encoded in the context vector.
- **SQL Query:** As the Decoder generates SQL tokens, they are combined to form the final SQL query.

The Text2SQL module, using a Seq-to-Seq model, directly learns the mapping between natural language queries and SQL queries. Through a process of encoding the natural language input and decoding the corresponding SQL output, it enables seamless communication between users and the bank’s database, allowing users to interact using natural language queries via the chatbot interface.

3.2 Challenges

The implementation of a Text2SQL module for a banking chatbot comes with several challenges. In this subsection, we discuss the main challenges and potential approaches to address them:

Model Selection: With the emergence of new Language Model architectures, such as LLMs mentioned in Section 2, it becomes challenging to determine which model would yield the best performance for the Text2SQL task. Selecting the most appropriate model requires careful consideration of factors like model size, training data, computational resources, and performance on the specific

task. Conducting comparative experiments and model evaluations will be essential to identify the optimal model for our use case.

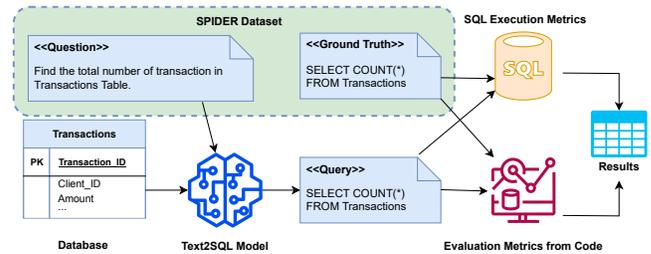


Figure 2: Pipeline of Text to SQL Evaluation Practice

Optimizing Generated Questions: The chatbot system’s generated questions may not always be in the most effective format for the Text2SQL module. Optimizing the generated questions to align better with the model’s input expectations can improve overall performance. Techniques such as data augmentation, paraphrasing, or using intermediate templates may be explored to enhance the quality of user queries and ensure better Text2SQL conversion.

Cost-Performance Trade-off: Most of the companies and research groups have limited computing and financial resources. Thus, achieving a high performance-to-cost ratio is crucial. LLMs and extensive training datasets can be computationally expensive and resource-intensive. Exploring smaller, more efficient models, and leveraging transfer learning techniques could be potential solutions to balance performance and resource constraints. In addition to cost considerations, solutions that consume fewer resources also offer environmental advantages.

Evaluation Metrics and Database Access: Evaluating the performance of the Text2SQL module becomes challenging when direct access to the real database is limited or restricted, like when we working on a bank database. It is essential to define suitable evaluation metrics that can measure the module’s effectiveness without the need for executing queries on the actual database. Metrics such as logical form matching and semantic correctness can provide insights into the module’s performance even when direct execution is not feasible.

Limited Banking Domain Dataset: Availability of a sufficient dataset specifically tailored to the banking domain is crucial for training a reliable Text2SQL model. Collecting domain-specific data can be time-consuming and may require domain experts. Leveraging transfer learning with pre-trained models on larger datasets

and incorporating domain-specific fine-tuning can help mitigate the data scarcity issue.

To address these challenges effectively, we propose a combination of strategies, including thorough experimentation with different models, techniques for question optimization, efficient resource utilization, well-defined evaluation metrics, and creative data augmentation methods. Additionally, collaborating with domain experts and exploring partnerships with other organizations in the banking domain could help access more relevant data and resources to improve the Text2SQL module’s performance for the bank’s chatbot.

4 EMPIRICAL SET UP

In this section, we propose the research questions and our practice set up to resolve them.

4.1 Research Questions

Our evaluation tackles the following 5 research questions (RQs):

RQ1. Which LLM is the most accurate for the Text2SQL task?

RQ2. What is the effect of the question rephrasing, prompt optimization, and post-processing steps on the performance of LLMs?

RQ3. Which LLM is the most efficient regarding execution time and computational resources?

RQ4. When the database is not reachable, can we propose metrics to evaluate the quality of the generated SQL queries that are as accurate as the Execution Match metric?

RQ5. Text2SQL tasks in financial scenarios (especially banks) are lacking in existing datasets. How should we solve it?

4.2 Model Selection

Our selection of LLMs is basically based on the selection of metrics in three directions, including the overall ranking list of LLMs, the ranking of Spider datasets, and LLMs with superior performance on Stack datasets.

As the undisputed performance ceiling, we choose ChatGPT based on GPT-4 as one of our research targets, and an interesting topic is how big the gap between other models and ChatGPT and GPT-4 [2] or GPT-3.5-Turbo.

HuggingFace Ranking After LLMs gained enough attention, the HuggingFace platform launched a LeaderBoard¹, where we observed that some models are considered ‘breakthroughs’ in this ranking, while others are optimized and fine-tuned versions of these models [4]. In order to avoid interference, we chose the original models (standalone releases) as we can find it. We also selected the most current models according to Figure 1 of the LLM development history, including LLaMA [29], GPT-NeoX, and so on. Based on the background above, we have chosen six LLMs to test against OpenAI models: MPT [27], ALPACA [26], KOALA [5], OpenAssistant-Pythia [12] [1], LLaMA-2 [30], and ORCA [19].

Spider Dataset Leaderboard Our test SPIDER dataset has a frequently updated benchmark performance ranking². We selected the two models PICARD [25] and RESDSQL [15] as a baseline of non-LLM models.

¹https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard

²<https://yale-lily.github.io/spider>

The Stack Database The Stack dataset is one of the richest code-based corpora containing programming languages available. Therefore, we selected several LLMs as candidates based on their performance on the Stack dataset [11]. We used several typical models as the Supervised group (models that have been pre-trained on programming languages), taking into account the existing research like CodeGen2 [20], StarCoder [17], and nsql [14].

4.3 Data Selection

We run our experiments on two Datasets.

❶ **The SPIDER Dataset:** We selected the “dev” evaluation subset from the SPIDER dataset for our evaluation. This subset contains 20 databases, with questions in natural language together with the corresponding ground truth SQL queries. The 20 databases cover four domains: Business (4), Entertainment (7), Education (4), and Travel and Leisure (5). Ground-truth queries in this subset are executable, as confirmed in related work. The databases are evenly distributed, with nearly half (9) having four tables, 4 containing three tables or fewer, 4 featuring five tables, and the remaining databases consisting of five or more tables. This diverse dataset allows for a thorough evaluation of Text2SQL models across different domains and database complexities.

Questions in natural language and Difficulty Level: The Text2SQL task manifests varying degrees of complexity depending on the difficulty of the questions in natural language. The questions are spread over four distinct difficulty levels:

Level 1: This simplest level involves uncomplicated queries where users retrieve straightforward information from a single table.

Level 2: Moving up the complexity ladder, Level 2 introduces basic statistical operations such as SUM, AVG, and COUNT. This requires the translation of these arithmetic operations into corresponding SQL expressions.

Level 3: Progressing to Level 3, the complexity deepens with the inclusion of GROUP BY clauses. This necessitates an understanding of aggregate functions applied to grouped data.

Level 4: The highest echelon of complexity is reached at Level 4, where JOIN operations are introduced. This challenges the model to grasp relationships between multiple tables and generate intricate SQL statements that seamlessly connect data from different sources.

After classification, we have 267 level 1 questions, 239 level 2 questions, 120 level 3 questions, 408 level 4 questions in SPIDER dev-set.

❷ **A new small size Financial Dataset:** We designed a specific dataset for the Text2SQL task on the financial scene, which is a bank transactions database with 30 questions. We aim to challenge all SOTA models and find the gap between in-practice tests and research evaluation. The database structure is depicted in Figure 3. The database schema encapsulates the relationships within a financial domain, embodying the interplay of clients, beneficiaries, and transactions. Organized into three distinct tables—‘Source’, ‘Beneficiary’, and ‘Transactions’—the schema orchestrates the nuanced connections between primary keys, foreign keys, and regular columns. This schema not only delineates the core attributes associated with each entity but also employs color-coded annotations to differentiate primary keys, foreign keys, and standard columns.

Questions in natural language and Difficulty Level: We asked for expert help and manually rephrased all questions to follow ethics standards and avoid business conflicts. We created 30 questions with various difficulty levels: 8 Level-1 questions, 6 Level-2 questions, 5 Level-3 questions, and 7 Level-4 questions.

We release this new financial dataset to the public at the following address: <https://github.com/Etamin/FinChallenge>

| Source | Beneficiary | Transactions |
|-------------|----------------|------------------|
| Client_ID | Beneficiary_ID | Transaction_ID |
| Type | Bank_Branch_ID | Time |
| Branch_ID | Country_Code | Client_ID |
| Contract_ID | Country_Name | Beneficiary_ID |
| BIC_Code | BIC_Code | Currency |
| IBAN | IBAN | Amount |
| | | Transaction_Type |

Figure 3: Financial Dataset for Text-to-SQL (green cells indicate foreign keys)

4.4 Rephrased Dataset

During our preliminary evaluation encompassing all SPIDER Dev datasets, it became evident that 9.4% of questions posed significant challenges for all models we tested, resulting in incorrect responses across diverse categories. These questions, which appear to be inadequately formed or ambiguous, can be classified as 'bad expressions'. To address this issue, two potential avenues emerge: manual rectification or leveraging automated models to curate improved question formulations; Given the tedious task of rectifying a substantial number of questions, we opted to explore the efficacy of the automated approach. For all questions of the SPIDER dataset, we choose 5 rephrased questions via ChatGPT, and if any one of them can get a better execution match and semantic similarity, we will choose this "correct" question to replace the original one.

4.5 Experiment Settings

4.5.1 Environment. We have limited computing resources, Only one NVIDIA® DGX-1(4*V100(32GB)) can be used for inference. It includes a Intel® Xeon® E5-2698 v4 @ 2.20GHz CPU and 4 NVIDIA Tesla V100 GPU with 32Gib Memory.

This environment limits us to using smaller models such as the LLaMA model with 13 billion parameters for the testing. However, being restricted to the same number of parameters allows us to compare all of these models horizontally. And use int-8 inference mode if possible, which reduces GPU memory occupation to 1/4.

4.5.2 Prompt for LLMs. In this study, we address the prompt influence challenge by designing an instructive prompt format tailored to the Text2SQL task. Our proposed format entails a structured arrangement that begins with a task description, followed by pertinent database information, and concluding with the natural language question. This format aims to guide the LLMs explicitly, providing a clear outline of the required information and context. By segmenting the prompt into distinct sections, we intend to enhance the model's understanding of the task and streamline its generation of accurate SQL queries.

In a notable case study showcasing the significant influence of prompts on Text2SQL model performance, we examined the original ALPACA model. We introduced two distinct prompt types for

evaluation: Type I in Figure 4, aligned with the input format design of the T5 SOTA model, and Type II in Figure 5, structured according to the CodeX-Davinci input format. Strikingly, this change in input format led to a remarkable performance boost, with execution match rates soaring from 11.2% to 20.8%. This underscores the pivotal role that prompt design plays in shaping model outcomes. However, it is important to note that not all models respond equally to such prompts. Models like OpenAssistant, for instance, encounter difficulties in comprehending Type II prompts, resulting in a meager 5% execution match rate—a clear illustration of the nuances and challenges posed by varying prompt structures in the Text2SQL task.

```
Given the database structure as "| table_1: column_1, column_2, ... | ...", given the following database:
concert_singer | stadium : stadium_id, location, name, capacity, highest, lowest, average | singer : singer_id, name, country, song_name, song_release_year, age, is_male | concert : concert_id, concert_name, theme, stadium_id, year | singer_in_concert : concert_id, singer_id|
Give me only the SQLite query of the question(only raw text of the code):
```

Figure 4: Type I

```
###
SQLite SQL tables, with their properties:
#
# stadium(Stadium_ID, Location, Name, Capacity, ...)
# singer(Singer_ID, Name, Country, Song_Name, ...)
# concert(concert_ID, concert_Name, Theme, ...)
# singer in concert(concert_ID, Singer_ID)
#
### Show name, country, age for all singers ordered by age from the oldest to the youngest.
Answer:
SELECT
```

Figure 5: Type II

4.5.3 Post-Processor. Our post-processing procedure is a multi-step approach aimed at enhancing the reliability and security of the output generated by LLMs during Text2SQL tasks. In the first part, we implement a robust strategy by running the LLM five times to mitigate potential instability in its output, ensuring a more consistent and dependable result. In the second part, we meticulously filter out any extraneous elements that are not integral to the SQL query, streamlining the output for improved clarity and accuracy. Finally, in the third part, we rigorously examine the output to eliminate any potentially harmful or risky code, such as "DROP TABLE" commands, safeguarding the integrity and safety of the generated SQL queries. This comprehensive post-processing workflow not only enhances the quality of the LLM-generated SQL queries but also bolsters their security, making them more suitable for practical applications in database management and query processing.

5 EVALUATION METRICS

Let us consider the SQL query generated by the LLM under test and the ground-truth query from the dataset: we set the prediction query as token sequence Y' , and the ground-truth query as token sequence

Y. The model is $f(Q) = Y'$, in which Q represents the question written in natural language. Based on Qin's survey paper [24] on Text2SQL, we selected the evaluation metrics below.

- **Exact Match**

Being the simplest metric in our study, Exact Match(EM) is set to 1 if $Y = Y'$, otherwise it is set to 0.

- **Executable**

Assuming the SQL execution progress on input Y' is $P(Y') = Z$, and when $Z \neq NULL || Error$, the executable rate is set to 1, otherwise 0.

- **Execution Match**

It is the most common metric when we talk about Text2SQL tasks. We set $P()$ is the execution procedure. When $P(Y) = P(Y')$ execution match is set to 1, otherwise 0.

- **BLEU Score**

The BLEU (Bilingual Evaluation Understudy) score we mentioned in the Related Work section, is a widely used metric for evaluating the quality of machine-generated translations in natural language processing and machine translation tasks. We normalize it to 0 to 1, 1 is best, 0 is worst.

The financial industry operates within a stringent regulatory framework, given the potentially devastating socio-economic repercussions that may arise from mishandling data or making risky decisions. For example, beyond financial regulations, the General Data Protection Regulation (GDPR) has a significant impact on financial institutions operating within the European Union (EU) or processing the personal data of EU residents. Therefore, SQL queries generated automatically cannot be executed on the databases without proper GDPR checks. This means that performance of Text2SQL methods should be evaluated based on static (no-execution) metrics. Such metrics will further better assess the performance of Text2SQL in a non-binary manner: some models may generate incomplete or non-executable SQL queries that are still valuable as they are very close to the ground truth, hence to what the operator wished to have. Measuring the similarity of queries, in a reliable way, appears as a promising alternative for assessment. In this work, we propose the following two metrics:

- **SQAM (SQL Query Analysis Metric)**

This is the first SQL evaluation metric we propose, it evaluates input queries by breaking them down into their core components using regular expressions, including SELECT, FROM, WHERE, GROUP BY, HAVING, and ORDER BY clauses. It then further dissects these components into subcomponents, such as select columns and table names. The resulting accuracy score ranging from 0 to 1 is based on the subcomponents match rate.

- **TSED (Tree Similarity of Editing Distance)**

This is the second SQL evaluation metric we propose, using abstract syntax tree (AST) as features, calculating the editing difference between 2 ASTs from the prediction query and the ground-truth query. Then we calculate the TSED based on the editing distance D and the node numbers N of the larger AST from predict or ground truth. $TSED = D/N$. Similarly to SQAM, this metric also ranges from 0 to 1, 1 being the best and 0 the worst.

In the following subsections, we will introduce them in detail.

5.1 SQAM

The SQL Query Analysis Metric (SQAM)³ evaluation method operates through a systematic process of dissecting input queries into their fundamental components, such as the SELECT, FROM, WHERE, GROUP BY, HAVING, and ORDER BY clauses, employing regular expressions. These primary segments are further deconstructed into their subcomponents, including *select columns*, *table names*, and *where conditions*, through the application of additional regular expressions.

Subsequently, the SQAM evaluation metric computes an accuracy score by juxtaposing the subcomponents present in the input query with those found in the ground truth query. Notably, this comparison considers the hierarchical importance of each subcomponent within the query structure. For instance, components like the SELECT clause carry greater weight in the evaluation compared to others like the ORDER BY clause. The accuracy score, a crucial output of this process, is quantified as the percentage of matched subcomponents, thoughtfully weighted by their respective significance in shaping the overall query. This nuanced approach to evaluating query similarity through SQAM underscores its precision in assessing the alignment of essential query components, thereby providing a robust measure of query correspondence.

5.2 Tree Similarity of Editing Distance (TSED)

We propose a novel metric that we name Tree Similarity of Editing Distance (TSED)⁴ to assess the similarity between two SQL queries. This innovative approach, illustrated in Figure 6, encompasses three distinct stages to provide a comprehensive evaluation. The initial stage involves the transformation of the two SQL queries into Abstract Syntax Trees (ASTs), facilitating a structured comparison. Following this, we calculate the editing distance between these ASTs, capturing the extent of their dissimilarity. The final step involves normalizing this distance measure to a scale between 0 and 1, allowing for a straightforward and interpretable assessment of query similarity.

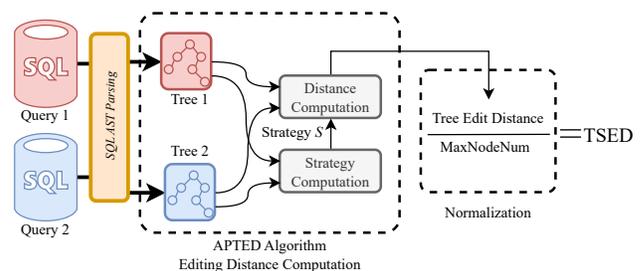


Figure 6: Pipeline of TSED Evaluation Metric

SQL Parsing: A Mathematical Perspective SQL (Structured Query Language), as a domain-specific language tailored for relational database management, inherently possesses a rich syntactic structure. The critical step of converting raw SQL text into its associated Abstract Syntax Tree (AST) is often referred to as SQL parsing. This conversion can be aptly modelled as a function, denoted as

³<https://github.com/ezzini/SQAM>

⁴<https://github.com/Etamin/TSED>

P . Given the space of all SQL queries, \mathcal{Q} , and the corresponding space of ASTs, \mathcal{A} , the function P establishes a mapping between these two spaces. For any particular query q that belongs to \mathcal{Q} , its corresponding AST is:

$$A(q) = P(q) \quad \text{where} \quad A(q) \in \mathcal{A} \text{ and } q \in \mathcal{Q} \quad (1)$$

Given the vast spectrum of SQL syntax, enriched with its numerous variations and intricacies, the process of parsing is undeniably intricate. A parser is tasked with navigating and interpreting diverse clauses, conditions, and embedded ambiguities within the SQL text.

Modern Solutions: One of the contemporary tools addressing this challenge is the `@florajs/sql-parser`⁵. Beyond traditional parsing, this tool embeds a nuanced transformation function. If we consider \mathcal{S} as the space encompassing all SQL strings, this transformation function creates a bridge from any SQL string s in \mathcal{S} to its corresponding AST, a , in \mathcal{A} :

$$a = T(s) \quad \text{where} \quad s \in \mathcal{S} \text{ and } a \in \mathcal{A} \quad (2)$$

We describe its cardinality, $|\mathcal{A}|$, using an indicator function δ to highlight the uniqueness of each AST:

$$|\mathcal{A}| = \sum_{i=1}^n \delta(A_i) \quad \text{where} \quad \delta(A_i) = \begin{cases} 1, & \text{if } A_i \text{ is unique within } \mathcal{A} \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

Hierarchical Transformation from JSON to AST. Upon translating an SQL query into its AST, which is then expressed as a JSON structure, we proceed to represent this AST as a tree. The tree offers a clear hierarchical perspective, encapsulating the relational and logical intricacies of SQL queries.

We introduce a bijective function $C : J \rightarrow T$, which is responsible for mapping every JSON object J into a tree representation, T , where T is represented as: $T = \{N, E\}$, where $N = \{n_1, n_2, \dots, n_k\}$ and $E = \{e_{12}, e_{13}, \dots, e_{(k-1)k}\}$.

Given this, the relationship between nodes n_i and n_j through the edge e_{ij} can be described by a matrix representation M :

$$M(n_i, n_j) = \begin{cases} 1, & \text{if } e_{ij} \in E \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

The tree's depth, $depth(T)$, corresponds to the maximum nested level, d , in J . This depth is intrinsically linked to the longest path P in T :

$$d(J) = \max_{\forall \text{ paths } P} \text{length}(P) \quad (5)$$

Our synthesis approach employs a recursive methodology. As we iterate over each key-value pair in J , nodes are birthed in N . When encountering a nested JSON object or array, the function delves recursively to ensure that the hierarchies in J and T are synchronized. Now, let $B(n)$ represent the set of children nodes for node n . The tree's branching factor, b , can be formulated as: $b = \max_{n \in N} |B(n)|$.

Tree Distance Computation. One of the fundamental challenges in comparing trees, especially those derived from SQL queries, is

quantifying their similarity or difference. Here, the concept of tree edit distance comes into play.

The tree edit distance between two trees T_1 and T_2 measures the minimum number of basic edit operations (insertion, deletion, and renaming of nodes) required to transform T_1 into T_2 . Mathematically, the tree edit distance function Δ can be defined as:

$$\Delta(T_1, T_2) = \min_{ops} \sum_{i=1}^n w(ops_i) \quad (6)$$

where ops is a sequence of n edit operations that transform T_1 to T_2 , and $w(ops_i)$ is the cost associated with the i^{th} operation. The goal is to find the sequence of operations that minimizes the total cost.

However, computing the tree editing distance is computationally intensive. The problem is NP-hard, and naive algorithms have a time complexity that is doubly exponential in the size of the trees. Let $|T_1|$ and $|T_2|$ denote the sizes of trees T_1 and T_2 respectively. The computational complexity C can be represented as:

$$C(|T_1|, |T_2|) = O\left(2^{2^{|T_1|+|T_2|}}\right) \quad (7)$$

Given the need for efficient computation, we employ the 'APTED' algorithm [23] [22], a state-of-the-art tool designed for tree edit distance computation. The library optimizes the process using advanced algorithms and data structures, significantly reducing computational time. In our project, the 'APTED' function compute the distance, the return δ will be an integer: $\delta = APTED(T_1, T_2)$, where δ represents the tree edit distance between T_1 and T_2 .

In essence, tree edit distance provides a robust metric to quantify the similarity between SQL query structures, and the 'apted' library facilitates its efficient computation, enabling a deeper understanding and comparison of different SQL queries.

Normalization.

Beyond merely calculating the raw editing distance, we take into account the complexity of the queries by normalizing the distance using the maximum number of tree nodes:

$$TSED = \delta / \text{MaxNodes}(T_1, T_2) \quad (8)$$

This normalization allows us to obtain a more comprehensive and meaningful final result, as it accounts for variations in query complexity and structure. By computing the distance-to-maximum-nodes ratio, we arrive at a final evaluation metric that offers a nuanced perspective on the accuracy and efficiency of our Text2SQL conversion techniques. This approach ensures that our assessments are not only sensitive to the accuracy of the conversions but also consider the relative complexity of queries, enabling a more balanced and informative measure of performance.

6 EXPERIMENT RESULTS

6.1 Benchmarks

Execution Match Score on the SPIDER dataset

In our experimentation, we organized the models into three distinct groups as illustrated in Table 1: general purpose LLMs, Code-Specific LLMs, and Sequence-to-Sequence models. Table 1 further presents the Execution Match score on the SPIDER dataset for each studied LLM and for each of the four difficulty levels. Note that for all LLMs, we run our experiments with both Type I and Type II prompts (cf. 4.5.2), and we always select best performance.

⁵<https://github.com/florajs/sql-parser>

| Model Type | Model Name | Parameter Size | Level 1 | Level 2 | Level 3 | Level 4 | All |
|-------------------|-----------------------|----------------|---------|---------|---------|---------|--------------|
| General LLM | ChatGPT-3.5-turbo | 175B | 0.760 | 0.799 | 0.408 | 0.493 | 0.623 |
| | DIN-SQL+GPT-4 | 1.76T | 0.861 | 0.866 | 0.700 | 0.654 | 0.762 |
| | CodeX-Davinci-3 | 175B | 0.730 | 0.799 | 0.392 | 0.382 | 0.570 |
| | MPT-7B-instruct | 7B | 0.262 | 0.381 | 0.117 | 0.091 | 0.205 |
| | ALPACA | 7B | 0.311 | 0.460 | 0.192 | 0.083 | 0.242 |
| | KOALA | 7B | 0.195 | 0.218 | 0.017 | 0.071 | 0.131 |
| | OpenAssistant-pythia | 12B | 0.202 | 0.322 | 0.025 | 0.069 | 0.157 |
| | ORCA-mini | 7B | 0.243 | 0.280 | 0.101 | 0.076 | 0.169 |
| Code Specific LLM | LLaMA-2 | 7B | 0.225 | 0.393 | 0.101 | 0.081 | 0.192 |
| | CodeGen2 | 7B | 0.375 | 0.498 | 0.167 | 0.066 | 0.257 |
| | StarCoder | 15.5B | 0.584 | 0.628 | 0.275 | 0.208 | 0.410 |
| | Vicuna | 7B | 0.060 | 0.134 | 0.008 | 0.042 | 0.064 |
| Seq-to-Seq Model | nsql | 6B | 0.772 | 0.732 | 0.608 | 0.277 | 0.548 |
| | T5(tscholak/cxmefzzi) | 3B | 0.828 | 0.782 | 0.650 | 0.434 | 0.641 |
| | PICARD+T5 | 3B | 0.790 | 0.799 | 0.558 | 0.502 | 0.652 |
| | RESDSL | 3B | 0.872 | 0.857 | 0.666 | 0.696 | 0.775 |

Table 1: Benchmark Results of Execution Match of all Models we tested on the "dev" SPIDER dataset

The overall winner is the GPT-4 + DIN approach which emerged as the most effective choice across all General LLMs. Furthermore, when focusing on models with fewer than 7 billion parameters, ALPACA stood out as the top-performing option following prompt optimization.

Within the Code-Specific LLMs group, nsql-6b took the lead as the superior model, with the entire subgroup showcasing significantly improved performance compared to other LLM models of a similar parameter size. This finding underscores the value of specialized models in handling domain-specific tasks effectively.

Meanwhile, the Seq-to-Seq models demonstrated a consistently high level of performance, closely resembling the capabilities of models like RESDSL. However, we anticipate that these Seq-to-Seq models may face formidable competition in the near future.

RQ1: Which LLM is the most accurate for the Text2SQL task?

Answer: The super giant GPT-4 model is the best-performing LLM. However, the results suggest that specific training datasets and "focused" LLMs can also perform well. For example, nsql-6b performs better than other models with 7 billion parameters. This means that general LLM is not a silver bullet. We also notice that the Level-4 questions are always hard for every model, and Level-1 and Level-2 questions usually lead to similar performances for these models.

Time and Memory performances of Three typical LLMs

To investigate the time and memory performance of LLMs, we selected three typical LLMs that we can download and run on our local machine. We selected T5 (with 3B parameters) and two versions of LLaMA (7B and 70B parameters).

As indicated in Table 2 above, it is evident that the 70-billion-parameter model consumes a substantial 70 gigabytes of GPU memory, rendering it impractical for use on our modest DGX workstation with limited memory capacity. Furthermore, the inference time for this model exceeds a staggering 100 seconds per instance. These findings underscore the critical importance of evaluating the performance and resource requirements of LLMs, as they demand considerable computational resources that may not be feasible or efficient for certain hardware setups and use cases.

| Model | Time | GPU Memory |
|------------------|-------|------------|
| T5-3B(fp32) | ~1.6s | ~12GiB |
| LLaMA-7B(int-8) | ~5.5s | ~7GiB |
| LLaMA-70B(int-8) | ~100s | ~70GiB |

Table 2: Inference Time & Memory for Different Parameters

RQ3: Which LLM is the most efficient regarding execution time and computational resources?

Answer: We highly recommend the adoption of the 7-billion-parameter model for small-sized businesses or research groups. When considering the performance of nsql-6b, with dedicated attention to training data and prompt optimization, the 6-billion-parameter model can surpass the capabilities of certain 13-billion-parameter or larger models. This presents a cost-effective and efficient alternative for organizations seeking high-performing language models without the extensive computational overhead associated with larger counterparts.

6.2 Rephrase SPIDER

This experiment aims to assess whether rephrasing a question helps generate better SQL queries. As explained in Section 4.4, thanks to ChatGPT, we first generate 5 variants of the initial questions. One critical challenge to conduct this experiment is that a typical LLM does not generate always the same answer, even if the question is always the same. Consequently, if a generated SQL query is better with a rephrased question, we do not know whether the query is better because of the rephrasing or because of the random nature of the LLM. To overcome this issue, we perform this experiment with the only model that stays "constant", i.e., which always generates the same answer each time we ask the same question. This model is PICARD. According to the result presented in Table 3, the dataset with rephrased questions yields better results than the original SPIDER dataset. This result suggests that a research direction on question optimization is worthy of consideration. Oftentimes, models just respond "badly" to "bad" problems.

| Batch | Exec Match | BLEU | SQAM | TSED |
|----------|--------------|--------------|--------------|--------------|
| Original | 0.651 | 0.249 | 0.847 | 0.617 |
| Rephrase | 0.797 | 0.271 | 0.891 | 0.725 |

Table 3: Rephrase Results on PICARD

RQ2: What is the effect of the question rephrasing, prompt optimization, and post-processing steps on the performance of the LLM?

Answer: Results from Sections 4.5 and this SubSection suggest that optimizing questions and prompts is very effective and can significantly improve performance in certain scenarios. Performance improvements of 5% and 14% were achieved on ALPACA and PICARD respectively.

6.3 TSED & SQAM

In this experiment, we study the effectiveness of the two metrics we proposed, TSED and SQAM, in measuring the quality of the generated SQL queries. We consider a metric as effective if the metric is "close" to the Execution Match metric, which is the best metric (among the ones we consider in this study) to measure semantic similarity between a generated SQL query and the ground truth SQL query. As a reminder, as explained in Section 5, the execution match metric cannot always be used under the constraints of our industrial partner. This is why we proposed TSED and SQAM, which can be computed without executing the generated queries.

TSED and SQAM scores (as well as Bleu Score, Exact Match, and Executable) are presented in Table 4. Impressively, both TSED and SQAM metrics exhibited very high Pearson product-moment correlation coefficients (PCCs) with execution match, averaging around an exceptional 0.95 as shown in Table 5. Notably, TSED demonstrated a slight edge over SQAM in terms of PCC. However, what sets TSED apart is its remarkable performance in the SSD (Sum of Squares of Deviations) metric, where lower values are indicative of superior accuracy. As depicted in Figure 7, both TSED and SQAM exhibited trends closely aligned with execution match, with TSED displaying an even closer correlation. Notably, we observed that the BLEU score for the higher performance part proved to be unreliable in this context.

In a noteworthy special case, we observe that TSED (Tree Similarity for Editing Distance) proves to be more effective than BLEU when evaluating the similarity of SQL queries. Let us consider two SQL queries: Query 1:

```
SELECT stadium.name, count()
FROM concert
JOIN stadium
ON concert.Stadium_ID = stadium.Stadium_ID
GROUP BY concert.stadium_id;
```

and Query 2:

```
SELECT T2.name, count()
FROM concert AS T1
JOIN stadium AS T2
ON T1.stadium_id = T2.stadium_id
GROUP BY T1.stadium_id;
```

Remarkably, both queries achieve a perfect execution match with a score of 1.0, indicating that they produce identical results when executed. However, when assessed using the TSED metric,

| Model Name | BLEU | SQAM | TSED | Eable | EM |
|----------------|--------------|--------------|--------------|--------------|--------------|
| ChatGPT-3.5 | 0.347 | 0.728 | 0.510 | 0.885 | 0.219 |
| DIN-SQL+GPT4 | 0.508 | 0.821 | 0.651 | 0.988 | 0.319 |
| Davinci | 0.348 | 0.564 | 0.575 | 0.844 | 0.235 |
| MPT-7B | 0.193 | 0.557 | 0.343 | 0.465 | 0.020 |
| ALPACA-7B | 0.205 | 0.572 | 0.384 | 0.552 | 0.126 |
| KOALA-7B | 0.109 | 0.444 | 0.240 | 0.305 | 0.022 |
| OpenAssistant | 0.160 | 0.491 | 0.260 | 0.348 | 0.050 |
| ORCA | 0.163 | 0.457 | 0.258 | 0.399 | 0.047 |
| LLaMA-2 | 0.193 | 0.561 | 0.298 | 0.417 | 0.074 |
| CodeGen2-7B | 0.264 | 0.615 | 0.367 | 0.583 | 0.124 |
| Starcode-15.5B | 0.287 | 0.649 | 0.448 | 0.769 | 0.160 |
| Vicuna-7B | 0.122 | 0.428 | 0.063 | 0.102 | 0.026 |
| nsql-6B | 0.443 | 0.814 | 0.602 | 0.741 | 0.308 |
| T5-3B | 0.243 | 0.843 | 0.671 | 0.863 | 0.409 |
| PICARD+T5/3B | 0.249 | 0.847 | 0.617 | 0.904 | 0.371 |
| RESDSL | 0.210 | 0.784 | 0.621 | 0.998 | 0.037 |

Table 4: Evaluation Metrics: Bleu Score, SQAM, TSED, Executable (Eable), and Exact Match (EM), on the SPIDER dataset. Scores are computed as an average of the scores for the 4 question difficulty levels.

| | BLEU | SQAM | TSED |
|-------|-------|-------|--------------|
| PCC ↑ | 0.681 | 0.948 | 0.951 |
| SSD ↓ | 0.804 | 1.022 | 0.129 |

Table 5: Correlation and statistical difference between 3 evaluation metrics and Execution Match

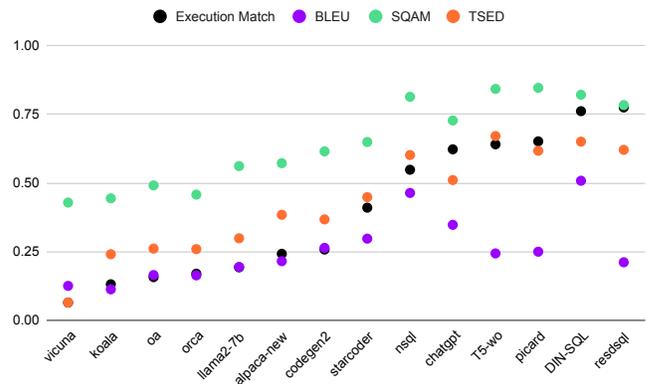


Figure 7: Distribution of Evaluation Metrics

these queries exhibit a similarity score of 0.864, signifying a high degree of structural similarity in their abstract syntax trees (ASTs). In contrast, the BLEU score, which ranges from 0 to 1, reaches only a modest 0.257. This intriguing case underscores the nuanced nature of query evaluation metrics and highlights how TSED excels at capturing structural similarities that BLEU may overlook.

RQ4: When the database is not reachable, can we propose metrics to evaluate the quality of the generated SQL queries that are as accurate as the Execution Match metric?

Answer: TSED is a good metric that can effectively evaluate SQL queries without access to the database.

6.4 Experiments on the New Financial Dataset

For this experiment, we selected ChatGPT(GPT-4), ALPACA, and nsq1-6B, which performed the best according to Table 1 for their respective Model Type group. Regarding Seq-to-Seq models, we selected PICARD+T5 instead of RESDSQL because RESDSQL performed poorly on the new financial dataset.

We run the 4 LLMs on our financial dataset for the Text2SQL task, and Table 6 presents the results. Despite some models being specially designed for SQL generation in natural language processing, ChatGPT continues to stand as the state-of-the-art (SOTA) model for this task. However, when confronted with the challenges posed by this distinct dataset, which does not conform to the structured format of the SPIDER dataset, our model comparison revealed some intriguing results. Specifically, the PICARD model, which has shown exceptional promise in previous evaluations, failed to reach the same level of performance as ChatGPT. Even more notably, our minor dataset proved to be a formidable challenge for both language models, including non-Language Models (non-LLMs) and Language Models (LLMs). None of the models achieved an execution match surpassing the 70% threshold, underscoring the complexity of our bank transaction sub-dataset and the need for further research and innovation in Text2SQL tasks under non-standardized domains.

| Model | BLEU | SQAM | TSED | EM |
|-------------|--------------|--------------|--------------|--------------|
| ChatGPT-3.5 | 0.582 | 0.696 | 0.599 | 0.633 |
| ALPACA-7B | 0.417 | 0.596 | 0.425 | 0.300 |
| NSQL-6B | 0.444 | 0.678 | 0.486 | 0.433 |
| PICARD-3B | 0.057 | 0.705 | 0.484 | 0.566 |

Table 6: Results of Banking sub-data-set for Text2SQL, EM is Execution Match

RQ5: Text2SQL tasks in financial scenarios (especially banks) are lacking in existing datasets. How should we solve it?

Answer: We introduce an efficient new financial dataset and assess it using several state-of-the-art (SOTA) models. This newly crafted dataset poses a substantial challenge, featuring a notably higher incidence of level 4 complexity problems compared to the widely recognized SPIDER dataset. This observation underscores the significance of encouraging companies to engage in the creation and evaluation of such sub-datasets.

7 THREATS TO VALIDITY

The validity concerns most pertinent to our evaluation are internal and external validity.

Internal Validity. In this research, internal validity would encompass ensuring that the differences in performance observed among the selected LLMs on the provided datasets can be confidently attributed to the variations in model capabilities and not influenced by

confounding factors or unintended biases. In the realm of internal validity, the research strategically harnesses multi-time output averaging to mitigate the inherent variability in LLM outputs proposed by Tian et al [28]. Recognizing the potential for erratic responses, this approach tempers the impact of occasional instability, thereby fostering more reliable and consistent performance measurements. Furthermore, the study introduces a meticulously designed post-processor tailored to the idiosyncrasies of LLM output.

External Validity. External validity would involve evaluating whether the outcomes observed among the chosen LLMs on the given datasets can be extended to a broader range of scenarios, datasets, or real-world applications. On the front of external validity, the research extends its purview to encompass banking scenarios, thereby transcending the initial domain boundaries. By juxtaposing the performance of the selected LLMs against the original SPIDER dataset, the study ventures beyond its initial context and broadens the applicability of its findings.

Construct Validity. Construct validity would involve ensuring that the chosen LLMs, evaluation metrics, and datasets accurately capture the dimensions of interest related to Text2SQL performance. In terms of construct validity, the research takes a comprehensive approach by evaluating the performance of LLMs across five distinct evaluation metrics. By encompassing multiple dimensions of assessment, the study robustly captures the nuances of LLM-generated SQL queries.

8 CONCLUSION

In this paper, we investigated five research questions focusing on the practice of the text2SQL models in financial scenarios. We evaluate a number of LLMs on the SPIDER dataset for the text2SQL task, study challenges from financial practice, evaluate metrics that do not need to run the SQL queries, and optimize some of the questions and prompts. We contribute a benchmark that can be used for model selection, 2 open-source evaluation metrics called SQAM and TSED, and present a small text2SQL challenge of the bank database. We find three main research findings: ❶ We found that in the case of general-purpose 7B LLMs perform poorly on Text2SQL tasks and require more SQL-related training data. The nsq1-6B LLM appears to be a good alternative to giant general-purpose LLMs. ❷ The dataset needs to be optimized, especially the quality of questions, and find effective prompts. ❸ The TSED metric is effective in measuring the quality of generated SQL queries.

In our ongoing project, we have exciting plans for future work. We intend to extend the application of our innovative evaluation metrics beyond the banking domain and apply them to major code generation tasks, thereby broadening the scope of their utility. Additionally, we are in the process of developing an automatic toolkit tailored for Text-to-Code evaluation. This toolkit will not only streamline the evaluation process but also contribute to the standardization of evaluation procedures in the field.

Acknowledgement. This work was funded by Luxembourg’s National Research Fund (FNR) under the grant BRIDGES. We are grateful to the DataLab team at BGL BNP Paribas for their valuable insights and assistance.

REFERENCES

- [1] Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*. PMLR, 2397–2430.
- [2] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712* (2023).
- [3] Deborah A Dahl, Madeleine Bates, Michael K Brown, William M Fisher, Kate Hunicke-Smith, David S Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. 1994. Expanding the scope of the ATIS task: The ATIS-3 corpus. In *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*.
- [4] Beeching Edward, Fourrier Clémentine, Habib Nathan, Han Sheon, Lambert Nathan, Rajani Nazneen, Sanseviero Omar, Tunstall Lewis, and Wolf Thomas. 2023. Open LLM Leaderboard. https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard.
- [5] Xinyang Geng, Arnav Gudibande, Hao Liu, Eric Wallace, Pieter Abbeel, Sergey Levine, and Dawn Song. 2023. Koala: A Dialogue Model for Academic Research. Blog post. <https://bair.berkeley.edu/blog/2023/04/03/koala/>
- [6] Charles T Hemphill, John J Godfrey, and George R Doddington. 1990. The ATIS spoken language systems pilot corpus. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*.
- [7] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15565* (2022).
- [8] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685* (2021).
- [9] Huggingface. [n. d.]. Huggingface Leaderboard. https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard last accessed: May 2023.
- [10] George Katsogiannis-Meimarakis and Georgia Koutrika. 2023. A survey on deep learning approaches for text-to-SQL. *The VLDB Journal* (2023), 1–32.
- [11] Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro von Werra, and Harm de Vries. 2022. The Stack: 3 TB of permissively licensed source code. *Preprint* (2022).
- [12] Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi-Rui Tam, Keith Stevens, Abdullah Barhoum, Nguyen Minh Duc, Oliver Stanley, Richard Nagyfi, et al. 2023. OpenAssistant Conversations—Democratizing Large Language Model Alignment. *arXiv preprint arXiv:2304.07327* (2023).
- [13] Rainer Koschke, Raimar Falke, and Pierre Frenzel. 2006. Clone detection using abstract syntax suffix trees. In *2006 13th Working Conference on Reverse Engineering*. IEEE, 253–262.
- [14] Numbers Station Labs. 2023. *NSText2SQL: An Open Source Text-to-SQL Dataset for Foundation Model Training*. <https://github.com/NumbersStationAI/NSQL>
- [15] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 13067–13075.
- [16] Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. 2023. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. *arXiv preprint arXiv:2301.07507* (2023).
- [17] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. 2023. StarCoder: may the source be with you! *arXiv preprint arXiv:2305.06161* (2023).
- [18] Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022. P-Tuning v2: Prompt Tuning Can Be Comparable to Fine-tuning Universally Across Scales and Tasks. *arXiv:2110.07602* [cs.CL]
- [19] Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. 2023. Orca: Progressive Learning from Complex Explanation Traces of GPT-4. *arXiv:2306.02707* [cs.CL]
- [20] Erik Nijkamp, Hiroaki Hayashi, Caiming Xiong, Silvio Savarese, and Yingbo Zhou. 2023. Codegen2: Lessons for training llms on programming and natural languages. *arXiv preprint arXiv:2305.02309* (2023).
- [21] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Philadelphia, Pennsylvania, USA, 311–318. <https://doi.org/10.3115/1073083.1073135>
- [22] Mateusz Pawlik and Nikolaus Augsten. 2015. Efficient computation of the tree edit distance. *ACM Transactions on Database Systems (TODS)* 40, 1 (2015), 1–40.
- [23] Mateusz Pawlik and Nikolaus Augsten. 2016. Tree edit distance: Robust and memory-efficient. *Information Systems* 56 (2016), 157–173.
- [24] Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, et al. 2022. A survey on text-to-sql parsing: Concepts, methods, and future directions. *arXiv preprint arXiv:2208.13629* (2022).
- [25] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. *arXiv preprint arXiv:2109.05093* (2021).
- [26] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford Alpaca: An Instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca.
- [27] MosaicML NLP Team. 2023. *Introducing MPT-7B: A New Standard for Open-Source, Commercially Usable LLMs*. www.mosaicml.com/blog/mpt-7b Accessed: 2023-05-05.
- [28] Haoye Tian, Weiqi Lu, Tsz On Li, Xunzhu Tang, Shing-Chi Cheung, Jacques Klein, and Tegawendé F Bissyandé. 2023. Is ChatGPT the Ultimate Programming Assistant—How far is it? *arXiv preprint arXiv:2304.11938* (2023).
- [29] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).
- [30] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shrusti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).
- [31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).
- [32] Tao Yu, Rui Zhang, He Yang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, et al. 2019. Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases. *arXiv preprint arXiv:1909.05378* (2019).
- [33] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887* (2018).
- [34] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103* (2017).