Analysis of Text-to-SQL Benchmarks: Limitations, Challenges and Opportunities

Anna Mitsopoulou
Athena Research Center, Greece
anna.mitsopoulou@athenarc.gr

ABSTRACT
Despite being a fast-paced research field, text-to-SQL systems face critical challenges. The datasets used for the training and evaluation of these systems play a vital role in determining their performance as well as the progress in the field. In this work, we introduce a methodology for text-to-SQL dataset analysis, and we perform an in-depth analysis of several text-to-SQL datasets, providing valuable insights into their capabilities and limitations and how they affect training and evaluation of text-to-SQL systems. We investigate existing evaluation methods, and propose an informative system evaluation based on error analysis. We show how our dataset analysis can help explain the behavior of a system on different datasets. Using our error analysis, we further show how we can pinpoint the sources of errors of a text-to-SQL system for a particular dataset and reveal opportunities for system improvements.

1 INTRODUCTION
Text-to-SQL systems translate natural language (NL) questions to SQL relieving users from the use of SQL for accessing data in relational databases. In recent years, text-to-SQL systems [37, 44, 51, 58] have achieved significant advancements due to the use of large language models (BERT [9], T5 [49], GPT [48]) and the creation of task-specific datasets (e.g., WikiSQL [75], Spider [68]) used for system training and evaluation. These approaches tackle the text-to-SQL problem as a language translation problem, and they train a neural network on a large amount of {NL question/SQ query} pairs [27].

Unfortunately, unlike systems that translate from one natural language to another, or from natural language to code (e.g., Python), text-to-SQL systems face challenges and do not enjoy as broad adoption, despite the high competition that exists among them. The datasets used for the training and evaluation of text-to-SQL systems play a vital role in the performance of these systems as well as in determining the progress in the field.

While a system trained on a benchmark like Spider [68] may exhibit good performance on this benchmark, when it is used on a different benchmark or used in a real application/domain, it does not fare as well. Several factors, such as the type of SQL queries, their distribution, the domains, and even the size of the data, matter when training a system. A system cannot perform well for unseen (or even not “seen enough”) queries or data. On the other hand, when evaluating a text-to-SQL system, a text-to-SQL benchmark may create false expectations on the query translation capabilities of the system. For example, a system achieving 80% accuracy on a dataset with simple queries could be worse than one achieving 60% accuracy on a dataset with more complex queries. An absolute accuracy number does not provide much insight unless we consider the characteristics of the evaluation dataset, such as the types and distributions of NL and SQL queries. It is also important to be able to pinpoint the sources of the errors a text-to-SQL system makes, and hence reveal opportunities for system improvements.

While WikiSQL [75] and Spider [68] are the first large-scale, multi-domain benchmarks for training and evaluating neural text-to-SQL systems, several datasets have preceded and followed them (e.g., [6, 15, 32, 38, 70]) that serve different purposes and focus on different aspects of the text-to-SQL problem. In contrast to the effort to understand text-to-SQL systems through studies and surveys [1, 2, 7, 18, 24, 28, 39, 47], extensive studies and evaluations of text-to-SQL datasets are missing. However, as we explained above, understanding the capabilities and limitations of text-to-SQL datasets is vital for making progress in the field.

In this work, we present a structured survey of text-to-SQL datasets, their design objectives as well as their shortcomings. Moreover, we present a text-to-SQL dataset analysis methodology that provides a set of dimensions and measures to analyze and characterize the richness and distributions of the SQL queries, the natural language questions and the databases covered by a dataset. Using our methodology, we evaluate several datasets, and provide valuable insights into their value, complexity, and limitations. This analysis also provides insights into the limitations of current text-to-SQL systems, and reveals several opportunities for research for the development of more effective benchmarks.

Furthermore, we investigate the methods and metrics for evaluating text-to-SQL systems and we point out their shortcomings. We propose an alternative in the direction of a more informative evaluation that combines a new metric and error analysis based on an automatically generated SQL query categorization that can provide insights about the system capabilities and pain points. We show the potential of our approach for providing a more fine-grained system evaluation. In particular, we experimentally show how our dataset analysis can help explain the behavior of a system on different datasets. Using our proposed error analysis, we further show how we can pinpoint the sources of errors of a text-to-SQL system for a particular dataset.

In a nutshell, our contributions are summarized as follows:

• We present a structured study of text-to-SQL datasets.
• We present a methodology for evaluating text-to-SQL datasets.
• We present an in-depth analysis of several text-to-SQL datasets.
• We provide an error analysis method combining a new metric and an automatically generated SQL query categorization.
• We show the potential of our dataset analysis methodology and error analysis for more fine-grained and insightful system evaluations.

2 TEXT-TO-SQL SYSTEMS
Research on text-to-SQL systems dates back to seventies, however, recently, the use of deep learning techniques has given a great boost in the development of such systems [31]. The most
successful text-to-SQL systems \cite{36, 37, 44, 51} rely on pretrained language models (e.g., T5 or GPT-based architectures) and use techniques, like pretraining, structure utilization or tasks decoupling to improve the performance in the task.

**Pre-training.** Introducing pre-training tasks is a popular approach to improve a model’s performance. In text-to-SQL systems they have been commonly used to tackle task-specific problems, like schema linking. In more detail, GAP2SQL \cite{55} created a synthetic dataset and trained a model using four objectives (denoising, column prediction, column recovery, and SQL generation) resulting in an encoder that can better represent SQL. Seal \cite{63} introduces two schema-aware denoising objectives aiming to minimize the schema-linking problem. In the first objective, erosion, the model takes as input the natural language question and the schema with permuted, removed or added columns, and the requested output is the SQL query. The second objective re-permutes the mentioned entities in the SQL or NL question and trains the model to reconstruct their order. GRAPPA \cite{67} proposes a grammar-augmented pre-training framework for table semantic parsing, using the MLM objective in the natural language and table headers input. Spider-Realistic \cite{8} introduces three pretraining objectives (column grounding, value grounding, and column-value mapping) to better capture the alignments between the natural language and the tabular data of a database schema. GP \cite{72} proposes extra pretraining of the decoder to reduce SQL grammar errors.

**Structure utilization.** The use of LLMs introduces restrictions regarding the formulation of the text-to-SQL task. This is not a problem when we have to represent text, such as a natural language question, or a SQL query, but it restricts the potential information gathered from a database schema, which is better represented as a graph structure. Several systems introduce techniques that can incorporate the additional information provided by the database schema structure. IRNet \cite{19}, RATSQL \cite{58}, and LGEOSQL \cite{5} construct a graph with the database schema and natural language question entities that reference schema elements and use a relation-aware self-attention \cite{54} encoder to capture the relations between input segments. In \cite{4}, the authors use graph encodings for both the natural language question and the database schema and they introduce a Structure-Aware Dual Aggregation Network (SADGA) to learn the alignment between the two graphs. The rest of their architecture consists of an encoder with a relational-aware self-attention to further unify the SADGA representations and the commonly used decoder of \cite{66}. In RASAT \cite{46}, they construct a graph similar to RATSQL and create relation embeddings that pass to the multi-head relation-aware self-attention. GRAPHIX-T5 \cite{37} modifies the architecture of the T5 model by introducing a relational graph attention network (RGAT \cite{60}) and jointly passing to the decoder the output of the RGAT and the T5’s encoder block, to incorporate both the semantics and structure of the schema.

**Tasks decoupling.** Schema linking is one of the main challenges in the task of text-to-SQL. Recently, there has been a tendency to unb耳den the main translation model from the schema-linking problem. SLSQL \cite{34} proposes a schema-linking extension to the base model that can learn the relations between the natural language question and the schema elements and pass to the decoder a schema-aware representation. RESDSQL \cite{36} introduces a ranking-enhanced encoder that can rank the schema elements by relevance to the natural language question and provide the encoder only with the most similar to the natural language question. In DIN-SQL \cite{44} the translation is broken down into four simpler tasks, schema linking, SQL classification and decomposition, SQL generation, and self-correction.

Furthermore, many works focus on the robustness of text-to-SQL systems against small dataset perturbations \cite{6, 8, 13–15}, in an effort toward systems that can handle more diverse translation scenarios. Finally, there are works \cite{45, 56} that try to utilize existing benchmarks and incorporate them in the evaluation process.

## 3 Text-to-SQL Datasets

A text-to-SQL dataset is a set of NL/SQL query pairs defined over one or more databases. Text-to-SQL datasets play an integral role in the development and benchmarking of text-to-SQL systems. Notably, early non-neural systems did not rely on common benchmarks \cite{17}. WikiSQL and Spider are the first large-scale, multi-domain benchmarks that made training and evaluating text-to-SQL systems possible. Both have become very popular with Spider being the most used one.

We divide text-to-SQL datasets into 4 categories: (a) single-domain datasets contain queries defined over one database; (b) cross-domain datasets are defined upon a collection of domains and databases; (c) perturbed datasets are based on existing ones with introduced variations; and (d) augmented datasets are generated by automatic methods that create a large set of NL/SQL pairs. Table 1 groups text-to-SQL datasets and provides information about their size and domains. Note that a dataset may fall in more than one category (e.g., a perturbed dataset is also cross-domain). For easiness, we have grouped them based on their most prominent category.

### 3.1 Single-domain Datasets

Most of the text-to-SQL datasets that contain queries in a single domain were created before Spider, and were used for a particular system. The majority of them were published before 2017 but most of the SOTA text-to-SQL systems do not use them. This is mainly due to their small size, which limits their use for training neural models. Nevertheless, their size is not a problem in the evaluation process, where they could provide insights in a system’s performance in different use-case scenarios.

These datasets exhibit diversity in terms of (a) size, (b) creation methods, and (c) databases. Regarding their size, most of the datasets are small, with the exceptions of SEDE \cite{20} and MIMICSQL \cite{61}, which have a size similar to Spider’s. Regarding the creation method, most of the datasets were created through crowdsourcing/user studies (Yelp \cite{64}, IMDb \cite{64}, Scholar \cite{25}, Geoquery \cite{69}, Advising \cite{16}, ATIS \cite{22}, Fiben \cite{53}). Additionally, some datasets were automatically created using templates. These datasets include Restaurants \cite{43, 57} and MIMICSQL \cite{61}, which also included additional filtering of the produced questions by users. Finally, there are datasets created from user logs. These include Academic \cite{35}, which was generated from logs from the Microsoft Academic Search, and SEDE \cite{20}, which was created from logs from the Stack Exchange Data Explorer. Regarding the databases, the majority of the datasets are defined upon existing databases, in some cases with alterations or simplifications. For example, the FIBEN \cite{53} database is created by mapping two existing financial ontologies into one.

### 3.2 Cross-domain Datasets

WikiSQL contains simple SQL queries over Wikipedia tables from multiple domains. Spider consists of 10,181 questions and 5,693
unique complex SQL queries on 200 databases with multiple tables, covering 138 different domains. The Spider SQL queries are divided into 4 levels: easy, medium, hard, and extra hard. The difficulty is defined based on the number of SQL components, projections, and conditions so that more complex queries are considered harder.

Staqc [65] was created by mining SQL-related questions and their answers from Stack Overflow. To the best of our knowledge, its use as a training or evaluation dataset is very limited, probably because of the lack of a database schema. KaggleDBQA [32] contains a small number of queries upon real databases from Kaggle. BIRD [38] contains questions over large-scale databases aiming to better represent real use-case scenarios. Finally, EHRSQL [33] contains questions over two databases related to health records.

WikiSQL, Spider, and BIRD stand out in this category as they cover a broad spectrum of domains. Even so, there are databases and domains, such as scientific and business ones, that are more challenging than the ones in these datasets: they may have a very complex database schema, use special terminology, contain cryptic table and column names, and so forth. Inevitably, the general-purpose datasets, such as the ones above, cannot cover these particularities, and more work is required to make a text-to-SQL system work on a new domain. Additionally, despite current efforts [32, 38], it is unclear if the existing benchmarks cover queries of different difficulty levels that capture the challenges present in real-world use cases.

### 3.3 Perturbed Datasets

As we will see in our analysis, Spider has several drawbacks, and serving as the primary evaluation dataset conceals various shortcomings of systems. Creating a large text-to-SQL dataset from scratch demands extensive manual effort.

As an alternative, numerous initiatives focus on creating variations of existing datasets. These variations emphasize on specific challenges, such as schema linking, which is the correlation of the natural language question with the database elements (tables, columns, values). Their goal is to provide a more accurate assessment of the system capabilities and/or boost these capabilities by enriching the training dataset.

Spider-Syn [14] is created by replacing schema references in the NL questions with their synonyms in the train and development Spider sets. Spider-Realistic [8] is created by removing or paraphrasing explicit mentions of column names from a subset of NL questions in the Spider development set. MT-TEQL [40] is generated by applying transformation rules in the schema or the utterance of the Spider queries. ADVETA (ADVErsarial Table perturbAtion) [42] is built by applying perturbations in the tables of WikiSQL, Spider, and WTQ. Spider-DK [15] is derived by selecting a sample of the Spider development set and creating paraphrases of the natural language questions to incorporate domain knowledge. DR-Spider [6], created by applying perturbations in natural language questions, queries, and database schemas, has been proposed to simulate diverse task-specific robustness challenges. Two synthetic datasets are proposed to quantify domain generalization [71].

Existing systems show a substantial performance drop across all perturbed datasets. This makes the value of such datasets in the evaluation process clear and highlights the limited capabilities of current text-to-SQL models in handling new datasets and specific challenges, even with small differences from the originals.
3.4 Augmented Datasets

The use of deep neural networks in the text-to-SQL task requires a large amount of data in the training process. The absence of a huge dataset for the text-to-SQL task and the low adaptability of existing models in databases without domain-specific training have led to several efforts to create augmented datasets. These datasets are used in the pretraining process either with the text-to-SQL task or with pretraining tasks defined in each work. GRAPPA [67] contains augmented NLQ-SQL pairs over WikiTables [3], and it was created by using a Synchronous Context-free grammar (SCFG) containing rules for the SQL queries and their corresponding questions induced from Spider examples. GAP2SQL [55] has been created by crawling SQL queries from Github and using a SQL-to-Text model to create the corresponding natural language questions. A method for creating an augmented dataset using a database-specific probabilistic context-free grammar (PCFG) and a SQL-to-Text system is described in [59]. The method was used to create augmented datasets from the Geoquery [69] and Spider databases and pre-train models in the downstream task. In SHiP [73], given a database schema, SQL queries are generated based on templates and a schema-weighted column sampling, and the corresponding natural language questions are built with a SQL-to-Text parser. Spider-CG [13] has been created by generating multiple variations of the natural language questions and the SQL queries from Spider. This was achieved by adding new clauses or conditions, or substituting existing ones in the SQL queries, along with the corresponding changes in the natural language questions. While previous efforts used the augmented dataset in the pre-training process, the authors of SpiderCG finetune their model only with their augmented dataset and do not use the training set of Spider.

The augmented datasets, with the exception of Gap2sql, do not significantly enhance the diversity of SQL queries, because their production rules rely on templates that exist in current datasets. The variation from existing SQL queries is based on the different utilization of the variables (tables, columns, values). On the other hand, the natural language questions in most of the systems are new, as they are produced from a deep neural network. In general, the augmented datasets introduce new examples to some degree, can boost underrepresented categories, and pose new challenges.

The main problem that these datasets face is quality. They cannot guarantee syntactic and semantic correctness of the SQL and natural language questions due to the use of deep neural networks and the random selection of variables to fill in the SQL templates. Consequently, despite the observed boost in performance that existing systems have shown with the use of an augmented training dataset, we should keep in mind that the low quality of these datasets can lead to systems susceptible to semantic errors.

4 DATASET ANALYSIS METHODOLOGY

An analysis of the characteristics of the datasets used for training and evaluating text-to-SQL systems can help explain the performance of a system. For instance, poor performance in nested queries could be due to their absence in the training dataset, while excellent performance in another dataset could be due to the dataset’s poor ability to provide enough variant examples, which results in hiding system vulnerabilities.

Apart from the size of the dataset that is typically given, several works provide additional statistics for text-to-SQL datasets. These include the frequency of specific clauses [10, 32, 33, 38, 52, 68], the percentage of columns mentioned in the natural language questions [8, 32, 56], the quality of the natural language queries [40], the number of templates existing in the dataset [10, 20, 25], and summary metrics related to the SQL queries or the NL questions (e.g., the average conditions in the queries, the average length of the questions, etc) [20, 61]. Furthermore, dataset analysis typically incorporate statistics about the databases, such as the number of columns, tables or rows, or the size of the databases in a dataset.

Existing approaches focus on different characteristics of text-to-SQL datasets failing to provide a uniform, multi-aspect and fine-grained analysis and comparison of such datasets. To address this gap, we propose a methodology for characterizing text-to-SQL datasets that provides a set of facets that capture the diversity and distribution of the SQL queries, the natural language questions, and the databases in a dataset. Our methodology incorporates the statistics used in previous works along with several new additions.

4.1 SQL Queries

The analysis of the SQL queries in a dataset provides an overview of the type of queries that a system is capable of predicting and helps explain system shortcomings due to unbalanced query distributions in the training data. Our SQL query analysis methodology comprises five axes, summarized in Table 2.

**Structural variety.** To gain a general understanding of the variety of the SQL queries, we examine their structure. The structural components consist of select, from, where, group by, having, order by, limit, set operators, and nesting. Each SQL query is categorized based on the combination of its structural components. Then, the structural variety of the SQL queries in a dataset is shown by reporting the percentage of each structural combination in the dataset.

**Operator variety.** Operations (e.g. logical, mathematical) expressed in the natural language question must be translated into SQL. To explore the SQL complexity from this angle, we examine the operators of the SQL queries. We have considered operator types instead of operators to reduce the number of possible combinations. Specifically, each query is characterized based on the combination of the following operator types:

- Aggregates: count, max, min, sum, avg
- Comparison operators: =, >, >=, <, <=, !=, between, not between
- Logical Operators: and, or
- Arithmetic Operators: +, -, /, * %
- Membership Operators: in, not in
- Join Operators: join, outer join, left join, cross join, etc
- Like Operators: like, not like
- Null Operator: is null, is not null

The operator variety is shown by the percentage of each operator combination in the dataset.

<table>
<thead>
<tr>
<th>SQL queries</th>
<th>Structural variety, Operator variety, Operator usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Databases</td>
<td>Schema complexity, Schema quality, Database size</td>
</tr>
<tr>
<td>NL questions</td>
<td>Schema linkage, Lexical complexity, Syntactic complexity, Readability</td>
</tr>
</tbody>
</table>
**Operator usage.** The complexity introduced by the operators in a SQL query is not caused only by their variety, but additionally by their quantity. For that reason, we explore their usage in a dataset by reporting, for each number of operators, the percentage of SQL queries that contain so many operators.

While the first three axes (structural variety, operator variety and usage) focus on the query complexity, the next two focus on the interaction of the query with the database elements.

**Schema usage.** The usage of schema elements in the SQL queries is shown by reporting the percentages of queries for each number of columns and tables used in the queries of a dataset.

**Content usage.** To understand the usage of the database values in the SQL queries, for each number of values mentioned in SQL queries of a dataset, the report includes the percentage of queries that contain this number of values.

### 4.2 Databases

The databases upon which the queries are formulated have a significant impact on the difficulty of the queries. Our database analysis in a dataset focuses on three axes.

**Schema complexity.** We measure the complexity of the database schema by calculating the number of tables and columns in each database of the dataset. The larger and more convoluted a database schema, the more complex SQL queries may become, and the more difficult it is to map NL questions to this schema. Schema serialization as the dominant approach to encoding the database schema in the input of text-to-SQL systems [27] may also be a problem.

**Schema quality.** To understand the schema quality of a database, we measure the percentage of schema elements that are valid English words. In this way, we can have an intuition on how easy a database schema is to be understood by a text-to-SQL system. For example, the attribute hadm in the MIMICSQL database, which refers to hospital admission, is not an English word, and it cannot be easily understood by a language model, on which all current approaches are based.

**Database size.** The database size can have an impact on a text-to-SQL system. Some systems implement schema linking methods that require a database search, whose overhead is affected by the database size. Furthermore, text-to-SQL systems typically focus on how to translate a NL question to an equivalent but not necessarily efficient SQL query. This oversight becomes critical as the performance of SQL queries deteriorates with the database size. For the database size, we report the total number of rows across tables in every database in a dataset.

#### Table 3: Example values of lexical complexity metrics

<table>
<thead>
<tr>
<th>Question</th>
<th>Rarity</th>
<th>Lexical density</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is the area of California?</td>
<td>0.33</td>
<td></td>
</tr>
<tr>
<td>What is the total number of patients who had coronary atherosclerotic native vessel?</td>
<td>0.5</td>
<td>0.57</td>
</tr>
<tr>
<td>How many such stocks are there whose last traded value does not exceed 1?</td>
<td>0.27</td>
<td>0.78</td>
</tr>
</tbody>
</table>

### 4.3 Natural Language Questions

The analysis of the natural language (NL) questions in a dataset helps understand the type of questions that a system is capable of successfully translating to equivalent SQL queries. Our analysis of the NL questions has four axes.

**Schema linkage.** One aspect that can determine the difficulty of the natural language questions in the task of text-to-SQL is how well they align with the underlying schema. Therefore, we report the percentages of the schema elements required in the corresponding SQL that are referenced by their exact name in the NL question.

**Lexical complexity.** A NL question may be expressed in simple words or use more rare words making it possibly harder for a text-to-SQL system to find an equivalent SQL query. To understand the complexity of the vocabulary used in NL questions, we adopt well-known metrics: (a) rarity: the ratio of the rare words to the content words of an NL question [50]; and (b) lexical density: the ratio of the content words to the total words of an NL question [26]. Content words are the important words (e.g., not the articles) of a text based on their part-of-speech tag. Table 3 presents example questions with their corresponding rarity and lexical density values.

**Syntactic complexity.** To measure syntactic complexity of a NL question, we report the: (a) Dependency depth: the depth of the NL question dependency tree; and (b) Length: the number of words in the NL question. As an alternative to the length, the number of dependencies in the NL question dependency tree could be reported.

**Readability.** We also report the readability of a NL question. For this purpose, we adopt one of the most popular formulas, the Flesch reading ease [11].

### 5 DATASET ANALYSIS RESULTS

We present the results of our analysis of text-to-SQL datasets using our methodology. We included publicly available datasets that are not derived from Spider, as the latter have small differences from the original dataset. Most of the datasets were obtained thanks to work in [10], in which they thoroughly collected multiple text-to-SQL datasets and made them easily accessible. We report the summary statistics across all axes of our methodology, enabling us to thoroughly compare the datasets. Due to space constraints, the full analysis of each dataset is included in our
5.1 Analysis of the SQL Queries

5.1.1 Structural variety. Figure 1 shows the percentages of the nine most common structural categories across datasets. As we can see the vast majority of the queries in the datasets are of type SFW. Most single-domain datasets exhibit small structural variety. Among them, Atis, MIMICSQL, and Restaurants, have exclusively SFW queries with or without nesting. On the other hand, Spider and KaggleDBQA have the highest structural varieties. This may be partially attributed to the existence of multiple databases in each dataset that lend themselves to expressing richer types of questions. Lastly, BIRD and Scholar are the more diverse ones containing the highest percentages of queries that do not belong to the most common categories. Overall, we observe a high imbalance in the structural categories of the queries in the datasets and a focus on a few, simple, categories.

5.1.2 Operator variety. Figure 2 depicts the eleven most common operator type combinations existing in the datasets. Geoquery and Advising are the only single-domain datasets that contain multiple operator categories in a substantial percentage. MIMICSQL offers some operator type variety but it is more imbalanced. The multi-domain datasets have the highest varieties. In the Advising dataset, half of its queries contain like operators. Lastly, Atis is the dataset with the most diverse operator type combinations, containing almost exclusively combinations other than the most common across datasets. Overall, the datasets do not cover a broad spectrum of operator combinations, while arithmetic, membership, and like operators appear rarely, if not at all.

5.1.3 Operator usage. Figure 3a depicts the number of operators used in queries in the considered datasets. In most datasets, queries use 0-10 operators. Interestingly, the most popular datasets (Spider, BIRD) are among the ones with the lowest use of operators.

5.1.4 Schema usage. Figure 3b shows the number of tables and columns used by the SQL queries in each dataset. The schema usage (and in particular the column usage) is proportional to the operator usage in the datasets. The vast majority of the queries in all datasets, except Atis, mention at most 10 columns. Atis is the dataset with the highest column usage, containing approximately 3 times more columns than most of the datasets. Regarding the number of tables used, the differences across datasets are small, and most of the times, queries contain fewer than five tables.

5.1.5 Content usage. Figure 3c provides insights into the number of values used in the queries in the datasets. Most queries in almost all datasets involve an average of less than four values. A notable exception are Atis queries, which contain 2-3 times more values than the queries in the rest of the datasets. In other words, Atis queries involve several conditions on values.

Overall, regarding schema and content usage, Atis queries make better use of the schema and content of the Atis database, while Spider and BIRD queries ‘touch’ few tables and columns.

5.2 Analysis of the Databases

5.2.1 Schema complexity. Figure 4 depicts the number of columns and tables for the databases in each dataset. The majority of the datasets contain a single database, resulting in one line in the plot. All databases have a small number of schema elements, with fewer than 25 tables and 125 columns. Focusing on the multi-domain datasets (Spider, BIRD, KaggleDBQA), we observe small variations in the schema complexity of their database collection. The small size of the existing schema allows schema serialization in the input of the systems, which is the most popular input method, however, is not necessarily representative of real databases with much larger schemas. Overall, all datasets use toy databases.

5.2.2 Schema quality. Figure 5 shows the percentages of database schema elements that are valid English words. The single-domain datasets and Spider contain a high percentage (> 75%) of explainable schema elements. The other cross-domain datasets (BIRD, KaggleDBQA) have less self-explainable database schemas.
5.2.3 Database size. Figure 6 shows the total number of rows in each database. As we can see, the biggest databases are Academic and IMDb followed by Yelp. The rest of the datasets have much smaller databases. Overall, these databases do not present significant efficiency challenges for the predicted SQL queries.

5.3 Analysis of the NL Questions

5.3.1 Schema linkage. Figure 7 depicts the exact schema reference percentages in the NL questions of all datasets. In most datasets, on average, only 10-35% of the NL questions contain exact schema references. The datasets with the lowest use of exact references are Restaurants, Atis and Advising. Spider is by far the dataset with the highest percentages, with a 50% average. The high number of exact references in Spider has been mentioned [8, 34] as a downside that makes the schema linking task easier than in real use cases.

5.3.2 Lexical complexity. Figure 8 shows the values of rarity and lexical density across all datasets. IMDb, Geoquery, and Academic questions have the lowest average rarity values. In other datasets, we can not detect noticeable differences. Regarding the lexical density, the average in most datasets varies from 0.3 to 0.6. MIMICSQL has the highest lexical density, while Academic is the one with the lowest. Overall, the datasets contain rather simple NL questions as they contain many pronouns and auxiliaries rather than nouns and lexical verbs (based on lexical density) and do not contain rare words (based on rarity).

5.3.3 Syntactic complexity. Figure 9 shows the values of the metrics regarding the syntactic complexity of the NL questions. Geoquery has the simplest questions, while BIRD and MIMICSQL have the more complex questions.

5.3.4 Readability. Figure 10 presents the readability scores of the questions existing in the datasets. The majority of the questions across datasets have a high readability score. MIMICSQL and Academic are the datasets with the lowest scores.

Scholar is the only single-domain dataset with the lowest schema quality across all datasets (~20%) because its column names are often concatenations of multiple words without underscore or camel case (e.g., citedpaperid). Overall, most datasets use easy database schemas.

5.4 Summary

Table 4 provides a summary of our findings across the considered aspects. For every aspect of our analysis, we group the results and create 3 different levels, Low, Medium, and High, characterizing the dataset regarding this aspect. The criteria defining each level are described in the long report in our GitHub repository.

5.4.1 Where do existing datasets fall short? The variety of SQL queries, NL questions, and databases in the existing datasets do not cover a broad spectrum of all the possible cases, raising several concerns regarding the robustness of the training and evaluation process. (1) The imbalanced training datasets can create several problems in the models like biased predictions and reduced generalization capabilities [21]. (2) Real applications typically involve more complex queries with several tables, conditions, nesting, formulas, etc. Thus, a system trained on an existing benchmark will probably not cope with these queries. (3) The distribution of queries in the evaluation datasets is also unbalanced giving more focus on simpler queries. As a result, a system’s accuracy will not represent accuracy balanced across different SQL types. (4) Real databases contain hundreds of tables and columns, which means that systems trained and evaluated on the existing benchmarks have not tested their translation capabilities over more realistic databases nor their capabilities of generating efficient queries. Additionally, taking into account the broad usage of fine-tuning techniques in the task of text-to-SQL, (5) existing datasets are fairly small compared to the datasets used for training neural models, such as models for code understanding and generation. For example, CodeSearchNet [23] used for training code-related models contains 2 million training examples, with even larger datasets created after it (Stack [30], The Pile [16]). Lastly, since most of the datasets have been built within
Table 4: Summary of datasets in the analyzed axes. L: Low, M: Medium, H: High.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Structural Variety</th>
<th>Operator Usage</th>
<th>Schema Usage</th>
<th>Content Usage</th>
<th>SQL Queries</th>
<th>NLQs</th>
<th>Databases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Academic</td>
<td>L</td>
<td>M</td>
<td>L</td>
<td>M</td>
<td>L</td>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>Advising</td>
<td>L</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>L</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>Geo</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>M</td>
<td>L</td>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>IMDb</td>
<td>L</td>
<td>L</td>
<td>M</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>Yelp</td>
<td>L</td>
<td>L</td>
<td>M</td>
<td>M</td>
<td>L</td>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>Scholar</td>
<td>M</td>
<td>L</td>
<td>M</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>Atis</td>
<td>H</td>
<td>H</td>
<td>M</td>
<td>M</td>
<td>L</td>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>Restos</td>
<td>L</td>
<td>L</td>
<td>M</td>
<td>M</td>
<td>L</td>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>MIMIC</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>Spider</td>
<td>M</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>K-DBQA</td>
<td>M</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>BIRD</td>
<td>M</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>M</td>
</tr>
</tbody>
</table>

Efforts to enhance the robustness of accuracy metrics and mitigate false results include Partial Component Matching F1, which is similar to Spider’s component matching but uses a parser that can process a larger set of SQL queries [20], an accuracy metric that considers semantically equivalent queries [29], and a metric called test suite execution accuracy [74] that tests the execution results of the queries over diverse variations of the database contents. Finally, QATCH [41] proposes a set of new metrics that can more accurately depict the capabilities of a system.

Efficiency metrics. Translating a NL question to SQL occurs with a non-negligible overhead. Furthermore, the predicted SQL query may not be the most efficient one, an issue that becomes more critical for databases with a large number of tables, columns and rows. Efficiency metrics include the latency of processing an entire query [12, 52], and the throughput, i.e., the number of queries that can be processed when a maximum number of processes are given [12]. A metric called VES (Valid Efficiency Score) computes the efficiency of the valid predicted SQL queries.

6 TEXT-TO-SQL EVALUATION METHODS

6.1 Existing Evaluation Approaches

Accuracy metrics. The primary method for evaluating the performance of a text-to-SQL system is by computing its accuracy, i.e., the percentage of the SQL queries that are translated correctly. This is accomplished by either comparing the predicted and ground truth SQL queries or by comparing their execution results. These correspond to Spider’s exact match and execution match and WikiSQL’s [75] logical form accuracy and execution accuracy metrics.

While these metrics are widely utilized, they are not completely accurate by design. The exact match can result in false negatives due to equivalent queries, while execution accuracy can result in false positives when distinct queries coincidentally produce the same execution result. In addition to that, Spider’s implementation of exact match produces erroneous results in several other cases. The most important one, as was also mentioned in [74], is the fact that the exact match does not consider the join’s “on” condition in the comparison. For example, the exact match score of the queries “select * from author join actor on author.name = actor.name” and “select * from author join actor on author.id = actor.id” will be 1, i.e., the queries will be erroneously considered the same.

Efforts to enhance the robustness of accuracy metrics and mitigate false results include Partial Component Matching F1, which is similar to Spider’s component matching but uses a parser that can process a larger set of SQL queries [20], an accuracy metric that considers semantically equivalent queries [29], and a metric called test suite execution accuracy [74] that tests the execution results of the queries over diverse variations of the database contents. Finally, QATCH [41] proposes a set of new metrics that can more accurately depict the capabilities of a system.

Efficiency metrics. Translating a NL question to SQL occurs with a non-negligible overhead. Furthermore, the predicted SQL query may not be the most efficient one, an issue that becomes more critical for databases with a large number of tables, columns and rows. Efficiency metrics include the latency of processing an entire query [12, 52], and the throughput, i.e., the number of queries that can be processed when a maximum number of processes are given [12]. A metric called VES (Valid Efficiency Score) computes the efficiency of the valid predicted SQL queries.
A query is valid if its result set aligns with that of the ground-truth SQL query. In this case, efficiency refers to the query’s running time.

Although these metrics are valuable parts of a comprehensive evaluation, they do not help understand translation errors.

**Query categorization.** Towards a more insightful evaluation, many systems categorize the SQL queries or the natural language questions existing in the dataset and report the accuracy results in every category. With this approach, they gain a deeper understanding of the system capabilities across different categories. The most popular categorization is the SQL query categorization introduced in Spider, which divides the queries into four groups based on hardness criteria. However, this categorization is too generic, and it fails to effectively highlight system challenges. While more extensive categorizations have been proposed [17, 62], they have not gained wide adoption. These categorizations, while beneficial, face challenges such as a non-automatic process for query classification, hindering their application to new datasets. Moreover, a common limitation lies in the lack of clear justification for the criteria underlying selected categories, limiting their broad applicability.

**Error analysis.** Another direction towards a more comprehensive evaluation is error analysis. In an effort to provide insights into their system's errors, many works [19, 29, 32, 34, 40, 58] manually select a subset of the wrong predictions made by their model, and group the error causes. This categorization provides useful information about the system’s downsides, but it requires extensive and repetitive manual work.

### 6.2 Automated Error Analysis

Given the above analysis of evaluation metrics, we introduce an automatic categorization for both queries (Section 6.2.1) and errors (Section 6.2.2) as the foundational step toward an evaluation framework that can be easily adapted across diverse contexts.

#### 6.2.1 SQL Categorization

Determining the best set of categories, in terms of error explainability, is not trivial. The first challenge is the definition of the features that constitute a category or can be combined to create a more general one. The second challenge is the optimal selection from these categories, which will result in a reasonable number of categories, capable of depicting the downsides of a model. We begin our study by defining two general sets of categories to examine if any valuable information can be gained and decide if a more thorough effort for a different categorization would be useful.

The first set of categories aims at structural categorization and contains all structural combinations of a query. The second one categorizes queries based on the operator types combination they contain. We selected these categorizations, as from the analysis of the datasets we observed that they can sufficiently depict the structural and operator variety of the SQL queries. Hence, they provide a more fine-grained analysis compared to template analysis [10, 20, 25], since templates combine structural and operator categories.

#### 6.2.2 Our Partial Match.

To create the error categories, we built on the components match defined in Spider, reformatting the components and defining three categories of matches: (a) structural match, (b) operator match, and (c) variable match. With this categorization, we try to identify problems that arise due to the difficulty of a model understanding the requested structure, the confusion in recognizing the requested relations, or the model’s inability to select the correct database components and extract the values from the natural language query, respectively.

**Structural Match.** To calculate errors in the structure, we check whether the predicted query’s structural components are equivalent to the ones of the gold query. In more detail, we create a set with the names of existing structural components in every subquery. In the case of nesting or of a set operator, we additionally store information for the position in which they exist (e.g., in the WHERE clause). The score of the structural match is produced by the average of the Jaccard similarity on the two sets for every compared subquery.

For example, for the gold query 'select name from students where age < (select avg(age) from students where age>17) and grade in (select grade from best_grades)' and the predicted query 'select name from students where grade>10 and age>17', the structural match will be:

\[
\text{avg}(\text{Jaccard}(\text{select, from, where, nesting\_where\_1}, \\
\text{select, from, where}\_2), \text{Jaccard}(\text{select, from, where}\_3)) = 0.2
\]

The main problem that arises with this approach is the selection of the subqueries to be compared. Due to the difficulty of finding an optimal solution, we choose a naive approach by comparing the subqueries with the order they exist in the query and we leave the exploration for a better solution as future work.

**Operator Match.** A similar approach is followed for the calculation of the operator match. In this case, we create the set, for each subquery, containing a unique entry for every operator, alongside its positional information. The operator match value for a pair of queries is the average of the Jaccard similarities of all subqueries.

**Variable Match.** To calculate the variable match of two queries, for each subquery, we create a set containing entries for the variable names and types, where the variable types consist of column names, variable names, and values. For models without constant prediction, we omit literals and numbers from the comparison. Finally, we define the average of the above 3 matches as the partial match score of two queries. With these metrics, we do not aim at predicting with precision the accuracy of a model. Instead, we focus on error explainability to validate our intuition, that a more detailed analysis of the errors could make the process of evolving text-to-SQL models easier and more robust.

#### 6.2.3 Discussion

The advantage of our method is that it automatically creates an error analysis that could assist the creator of a system to understand its pain points and produce more robust models. Hence, our method can be used as an additional tool for system evaluation. Nevertheless, it is not perfect as there are cases in which it falls short.

The proposed metrics for error analysis could result in false negatives in the case of equivalent queries. For example, the queries 'select name, age from singer order by age limit 3' and 'select name, age from singer where singer.id in (select singer.id from singer order by age limit 3)' will result in errors in all matches, even though the two queries are the same. To mitigate the impact of the equivalent queries in the depicted errors, our metric could be used only in queries where we know are wrong (e.g., in queries with 0 execution accuracy). Additionally, our proposed method will not point out the cause of errors related to the natural language questions. For example, the elevated errors in a structural category could be caused by ambiguities in the natural language question, but our metric will only show that the model struggles...
in this category. The exploration of the natural language effect in the models’ errors is an important aspect of the error analysis and we leave it for future work.

7 EXPERIMENTS FOR SYSTEM EVALUATION

In this section, we describe experiments with existing text-to-SQL models over the analyzed datasets. Our purpose is to show how the dataset analysis using our methodology of Section 4 can shed more light into the performance of a text-to-SQL model. Then, as a second step, we focus on Spider, and perform an error analysis as described in Section 6.2 that demonstrates how our approach can pinpoint the sources of errors and provide additional insights into the performance of a text-to-SQL system in a dataset.

7.1 Using Dataset Analysis in System Evaluation

We provide the results of text-to-SQL models in all the analyzed datasets focusing on insights that stem from the extra information provided by the analysis of the evaluated datasets. Hence, our focus is on showing the value of dataset analysis for system evaluation and not the value of any particular text-to-SQL model.

7.1.1 Models. We have selected several variations of the T5 model, which is used as the base component in several systems at the top of Spider’s leaderboard. The reason for this selection is primarily the available checkpoins of the T5 in the Spider dataset from [51] that made it easy to get the predictions of the models for all datasets. We did not select any LLCM, e.g., all the GPT-4 based architectures existing on top of the Spider and BIRD leaderboard, as their cost for getting the predictions in all datasets was prohibitive. More specifically, our models consist of T5-base_lm100k, T5-large, T5-large_lm100k, T5-3B and T5 with the PICARD method [51]. For the datasets that had the database available and in a .sqlite format (Geoquery, Atis, KaggleDBQA, BIRD, Restaurants, Advising) in addition to PICARD, we enabled the option of using the DB content provided by the authors [51]. The lm_100k suffix suggests that the model was trained for 100k additional steps with the language modeling objective and PICARD is a constrained decoding algorithm.

7.1.2 Dataset preprocessing. We removed unnecessary blank spaces from the literals (e.g., "VLDB" instead of "VLDB") from the gold queries of IMDb, Yelp, and Academic, as we saw that their execution resulted in empty sets and we assumed that they were wrong.

7.1.3 Metrics. To measure the performance of the models we used the execution accuracy and the implementation of the Spider’s exact match. Through experiments, we figured out that the exact match could not parse a large portion of the queries in several datasets. For this reason, we preprocessed the queries before passing them to the metric to correct some of the error cases that were fixable by reformattting the query. These include:

- Implicit joins. Queries that had tables in the from clause separated by a comma were not parsable. We replaced the implicit joins with the ‘join’ keyword.
- ≈ operator. We replaced the <> operator with the ‘!=’, which was parsable.
- Inner join. The exact match could not parse queries that specified the type of join (outer join, left outer join, inner join etc.). We could not reformat these queries without altering their logic, but we replaced the appearances of ‘inner join’ with ‘join’, as it is the default join method.
- Backquotes. We replaced backquotes in literals with quotes.
- where clause content in parentheses. We removed redundant parentheses in the where clause (e.g., ‘select * from singer where (name='A' and age>18)’).

These changes significantly increase the number of parsable queries, though there are still many that remain unparsable. The number of parsing errors in each dataset along with their causes are in our GitHub repository.

7.1.4 Results. Table 5 presents the performance of the T5 models in the analyzed datasets. As already mentioned, all models are trained on Spider. Hence, the table shows their performance in several datasets. It is important to mention that even though the value of evaluating a model over multiple datasets has been repeatedly underscored in the literature [8, 10, 15, 41, 42], most of the current systems are evaluated in only one dataset. The table’s results once again demonstrate the need for this broader evaluation.

Focusing on the results, we observe that the Atis, Scholar, and Restaurants datasets have the worst performances. If we recall our analysis, Atis is a dataset with many more conditions than other datasets and all three datasets are among the ones with the lowest percentage of exact schema references. Additionally, Scholar had the lowest percentage of explainable schema items. Through manual inspection of the predictions, we can observe that the above characteristics seem to be the main source of errors. As we can observe in the example predictions of the Atis and Scholar datasets in Table 6, the models struggle to connect with the schema, they often hallucinate schema elements and in the case of Atis, they produce much shorter queries than the ground truth ones.

Geoquery, KaggleDBQA and IMDb are the datasets with the best performances. We believe that the fact that they all contain a significant percentage of easy queries, i.e., queries with only one type of operator and SFV queries - combined with their low operator, schema, and content usage contributes to the correct prediction of a considerable portion of their corpus. Table 6 presents a correct prediction in the Geoquery dataset, demonstrating the simplicity of the NL question and the corresponding SQL query.

Similarly, we can detect hints of the primary challenges encountered by the models for most of the datasets. For instance, given the MIMICSQL dataset, we can observe that it has only simple queries with low schema usage. Combined with one of the lowest question readability, the high lexical and syntactic complexity in the natural language questions, and one of the highest lower bounds in the content usage we could infer that possibly the model struggles with understanding the DB content in the questions. For example, as we can see in Table 6, it omits the ‘mitral valve disorders’ value in the first example and it incorrectly translates the value ‘crnry athrscl natve vssl’ in the second example, as a schema element.

Regarding the capabilities of different models, we see that they tend to improve with changes in model size, extra pretraining, or the use of PICARD for datasets that are similar to the Spider (mainly regarding the SQL axes). This indicates that the techniques for improving the model affect datasets close to the one used for training, but seem to have limited gains in datasets with different characteristics.
Table 5: Models’ evaluation in several text-to-SQL datasets with execution match (EM) and execution accuracy (EX). ([base lm100k, l]arge, [l]arge{-lm}100k, [b]ase-lm100k + [P]ICARD (b+P))

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T5</td>
<td>59.4</td>
<td>59.3</td>
<td>4.2</td>
<td>16.8</td>
<td>0</td>
<td>0.4</td>
<td>4.6</td>
<td>6.1</td>
<td>2</td>
<td>-</td>
<td>11.3</td>
<td>16.7</td>
<td>11.4</td>
<td>14.4</td>
<td>4.6</td>
<td>7.8</td>
<td>0</td>
<td>1.3</td>
<td>4.1</td>
<td>0</td>
<td>7.2</td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>67</td>
<td>68.3</td>
<td>10.3</td>
<td>16.4</td>
<td>0</td>
<td>0.4</td>
<td>4.6</td>
<td>4.1</td>
<td>4.3</td>
<td>-</td>
<td>16.7</td>
<td>21</td>
<td>12.9</td>
<td>14.5</td>
<td>4.6</td>
<td>8.5</td>
<td>0</td>
<td>1.9</td>
<td>8.2</td>
<td>0</td>
<td>21.6</td>
<td></td>
</tr>
<tr>
<td>l-lm</td>
<td>71.1</td>
<td>73</td>
<td>13.8</td>
<td>19.6</td>
<td>0</td>
<td>0.2</td>
<td>3.5</td>
<td>5.1</td>
<td>4.7</td>
<td>-</td>
<td>15.1</td>
<td>21.6</td>
<td>18.3</td>
<td>18.3</td>
<td>5.4</td>
<td>11.7</td>
<td>0</td>
<td>2.2</td>
<td>7.6</td>
<td>0</td>
<td>26.4</td>
<td></td>
</tr>
<tr>
<td>3B</td>
<td>71.5</td>
<td>72.8</td>
<td>16.8</td>
<td>19</td>
<td>0</td>
<td>1.2</td>
<td>5.1</td>
<td>5.6</td>
<td>8.2</td>
<td>-</td>
<td>18.9</td>
<td>21.6</td>
<td>16</td>
<td>17.5</td>
<td>4.6</td>
<td>11.7</td>
<td>0</td>
<td>3.1</td>
<td>9.5</td>
<td>0</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>b+P</td>
<td>66.2</td>
<td>67.4</td>
<td>13.4</td>
<td>32.9</td>
<td>0.9</td>
<td>6.7</td>
<td>5.1</td>
<td>6.1</td>
<td>2.8</td>
<td>-</td>
<td>18.3</td>
<td>24.8</td>
<td>11.4</td>
<td>16</td>
<td>4.6</td>
<td>8.5</td>
<td>0</td>
<td>2.7</td>
<td>10</td>
<td>0</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Figure 11: Errors in Spider development set with Exact match (EM), Execution accuracy (EX), Structural match (SM), Operator match (OM) and Variables Match (VM).

Figure 12: Structural match errors in structural categories.

Figure 13: Operator match errors in operator type combinations categories.

The above serves as a demonstration of the valuable insights our dataset analysis can provide, aiding in both a better understanding of a model’s capabilities and the identification of its limitations.

7.2 Using Error Analysis in System Evaluation

To demonstrate the effectiveness of our error analysis in revealing additional information regarding the sources of errors in a model, we present the results of our method in analyzing the performance of text-to-SQL systems in the Spider dataset. We selected Spider because it is the most popular and we were able to collect results from multiple systems. More specifically, except from the above-used models, we report the results in RATSQL [58], which uses a task-specific encoder and decoder, and DINSQIL [44], which is based on GPT-4. For the RATSQL model, we reproduced the results of the RATSQL-BERT following the instructions in their repository, while for the DINSQIL, we have parsed the predictions from the given file in their repository.

Figure 11 shows the percentages of errors with the exact match, the execution accuracy metric, and our error metrics, namely partial, structural, operator, and variable match. The execution match is not reported for the RATSQL, since it does not predict the values in the produced SQL queries. For example, in the predicted SQL query “select distinct singer.country from singer where singer.age > ‘terminal’”, instead of a value in the age condition the model predicts just the ‘terminal’ symbol, that denotes a value existence.

We observe that the performance of the models decreases if we evaluate them, in the Spider development set, with our partial match, instead of Spider’s exact match. This happens due to the errors in the exact match implementation, as previously mentioned. In more detail, the distribution of the partial match errors for the T5 and DINSQIL models remains the same but slightly elevated, as opposed to RATSQL, in which we can see a large increase in errors.

Focusing on our metrics for error explainability, namely the structural, operator, and variable match, it is clear that the primary source of mistakes, in all models, is the wrong prediction of the variables. The second most important source of errors lies in the prediction of operators, while structural mismatches remain comparatively low and steady for all models. Focusing on the T5 model variations, increase in size, as well as extra pretraining, provide significant improvements, mainly by reducing variable errors. Regarding the RATSQL, we can see that there is a huge difference in the operator and variable match errors compared to the rest of the models. Finally, we can observe that the PICARD method with the use of column values in the input reduces mainly the errors in the variables, while surprisingly generating more errors in the structural match.

By combining the SQL categorization with the error categorization, we create Figures 12 and 13, which depict the structural
and operator match errors for the structural and operator categorization, respectively. In this way, we can observe more clearly the errors in each category.

Figure 12 shows that in the queries with nesting or set operators, the T5-3B model has the highest error percentage among the T5 models, possibly indicating that the training data for these more complex categories are not enough to successfully train a model this big. Additionally, DINSQL’s errors significantly increase in structural combinations with limit, nesting, or set operators. This behavior combined with this high difference between the exact and execution match could indicate that DINSQL produces equivalent queries with different structures, more often than the rest of the models. We also observe that different models seem to have achieved complementary understandings of the SQL structure. We should mention though that due to the low differences (less than 0.5%) those are not safe conclusions.

Figure 13 shows the huge deficiency of RATSQIL when there is a join operator, leading to the point that queries containing joins rarely produce a correct answer. Additionally, from the normalized errors, we can observe that all models, except DINSQIL, struggle the most in queries with logical operators and in the ‘other’ category, which contains the most rare operator combinations in the Spider dataset. This can be attributed to the use of Spider as a training dataset in the models, which creates biases regarding the predicted queries, and it highlights the importance of a more diverse dataset during the training process. Moreover, the small percentage of errors of these operator combinations, due to their low usage in the evaluation dataset, makes clear the importance of the distribution in the evaluation dataset in pinpointing model vulnerabilities.

**Overall**, error categorization can provide useful insights into the sources of errors and the differences between models. Given that our implementation is open source and the only requirement for the error analysis is a JSON file, with the predictions of a model over a dataset, we believe that it provides an easy way to start the analysis of any model, without extra overhead and enable an in-depth comparison of several state-of-the-art systems.

### 8 CONCLUSIONS

In this work, we introduced a methodology for text-to-SQL dataset analysis, and we performed an in-depth analysis of several text-to-SQL datasets. We examined existing evaluation methods, and proposed an automated error analysis method. We showed how our dataset analysis can help explain the behavior of a system better than the systems’ original evaluations. Using our error analysis, we further showed how we can pinpoint the sources of errors of a text-to-SQL system for a particular dataset. Our work provides several insights into the limitations of current text-to-SQL systems and datasets, and opens up opportunities for the development of more effective benchmarks, evaluation methodologies and systems. Future work could include the upgrade of the error metrics, to detect equivalences and to report false negatives and positives for each prediction, that could uncover biases of the models regarding operators or structural components. Additionally, we could explore extra axes in the datasets analysis, for instance, for the detection of ambiguities in the NL questions. Designing novel benchmarks using our dataset methodology is another important direction. Finally, the evaluation of SOTA text-to-SQL systems in the analyzed datasets with our error analysis would be a valuable next step to understanding the current capabilities in this task.

### REFERENCES


