NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

PROGRAM OF POSTGRADUATE STUDIES
Data Science and Information Technologies

SPECIALIZATION
Big Data and Artificial Intelligence

MASTER'S THESIS

Towards more robust text-to-SQL translation

Anna V. Mitsopoulou

ATHENS

MAY 2024

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

Επιστήμη Δεδομένων και Τεχνολογίες Πληροφορίας

ΕΙΔΙΚΕΥΣΗ

Μεγάλα Δεδομένα και Τεχνητή Νοημοσύνη

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αξιόπιστη μετάφραση από φυσική γλώσσα σε SQL

Άννα Β. Μητσοπούλου

ΑΘΗΝΑ

ΜΑΙΟΣ 2024

Master's Thesis

Towards more robust text-to-SQL translation

Anna V. Mitsopoulou

S.N.: DS1200014

MAY 2024

Διπλωματική Εργασία

Αξιόπιστη μετάφραση από φυσική γλώσσα σε SQL

Άννα Β. Μητσοπούλου

Α.Μ.: DS1200014

ΕΠΙΒΛΕΠΟΝΤΕΣ:

Γεωργία Κούτρικα, Διευθύντρια Έρευνας, Ερευνητικό Κέντρο Αθηνά

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ:

Γεωργία Κούτρικα, Διευθύντρια Έρευνας, Ερευνητικό Κέντρο Αθηνά

Γιάννης Ιωαννίδης, Καθηγητής, ΕΚΠΑ

Ίων Ανδρουτσόπουλος, Καθηγητής, ΟΠΑ

ΜΑΙΟΣ 2024

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.

SUBJECT AREA: Database Systems

KEYWORDS: Machine Translation, Deep Learning, Semantic Parsing, Databases

ΠΕΡΙΛΗΨΗ

Παρά τους γρήγορους ρυθμούς ανάπτυξης στο ερευνητικό πεδίο της μετάφρασης από φυσική γλώσσα σε SQL, τα Text-to-SQL συστήματα αντιμετωπίζουν κρίσιμες προκλήσεις. Οι συλλογές δεδομένων που χρησιμοποιούνται για την εκπαίδευση και την αξιολόγηση αυτών των συστημάτων είναι ζωτικής σημασίας για τον καθορισμό της απόδοσής τους και της εξέλιξης σε αυτό το πεδίο. Σε αυτή την εργασία, εισάγουμε μία μεθοδολογία για την ανάλυση των text-to-SQL συλλογών δεδομένων και παρουσιάζουμε την ανάλυσή τους, παρέχοντας πολύτιμες πληροφορίες για τις δυνατότητες και τους περιορισμούς των text-to-SQL συστημάτων, αλλά και τον τρόπο με τον οποίο οι συλλογές δεδομένων επηρεάζουν την εκπαίδευση και την αξιολόγηση αυτών των συστημάτων. Διερευνούμε τις υπάρχουσες μεθόδους αξιολόγησης και προτείνουμε μια αναλυτική αξιολόγηση συστήματος βασισμένη στην ανάλυση των σφαλμάτων. Δείχνουμε πώς η ανάλυση των δεδομένων μπορεί να βοηθήσει στην εξήγηση της συμπεριφοράς ενός συστήματος σε διαφορετικά σύνολα δεδομένων. Χρησιμοποιώντας την ανάλυση σφαλμάτων μας, δείχνουμε πώς μπορούμε να εντοπίσουμε τις πηγές σφαλμάτων ενός συστήματος text-to-SQL για ένα συγκεκριμένο σύνολο δεδομένων και να αποκαλύψουμε ευκαιρίες για βελτιώσεις του συστήματος.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Συστήματα Βάσεων Δεδομένων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Μηχανική Μετάφραση, Βαθιά Μάθηση, Σημασιολογική Ανάλυση, Βάσεις Δεδομένων

ACKNOWLEDGEMENTS

I would like to express the deepest appreciation to Dr. Georgia Koutrika, Research Director at Athena Research Center, for the excellent guidance and persistent support. I would also like to thank my family and friends for the love and support they have provided me throughout this work.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

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, with the use of large language models (BERT [11], T5 [52], GPT [51]) and the creation of task-specific datasets (e.g., WikiSQL [80], Spider [74]) used for system training and evaluation, text-to-SQL systems [39, 49, 54, 63] have achieved significant advancements. These recent approaches tackle the text-to-SQL problem as a *language translation problem*, and train a neural network on a large amount of {NL query/SQL} pairs [30].

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. There are several reasons that mainly stem from the fact that neural text-to-SQL systems cannot make good predictions for unseen domains and unseen queries.

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 [74] 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 the points where the system requires improvements. Text-to-SQL benchmarks fall short in highlighting the real capabilities and limitations of text-to-SQL systems.

In this work, we present an extensive list and a structured survey of existing text-to-SQL datasets that helps the reader get a good grasp of the landscape of benchmarks, 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 analyze several datasets, and provide valuable insights into their value, complexity, and possible limitations. This analysis also provides us with insights into the limitations of current text-to-SQL systems, and presents several opportunities for research for the development of more effective benchmarks and systems, alike.

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:

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

2. RELATED WORK

2.1 Code generation with Transformers

In the last few years, large pre-trained language models ([61], [11], [52], [51]) are increasingly used for several downstream natural language tasks. Following the state-of-the-art performance of these models, there have also been an increasing number of attempts to broaden their use in programming language generation and understanding tasks. As in language models, systems' architectures can be separated into encoder-only, decoder-only, or encoder-decoder.

2.1.1 Encoder-only

CodeBERT [12] is a pre-trained model in multiple programming languages. The training objectives used by the authors are the masked language objective and the replaced token detection objective, introduced in [8]. To adapt the second objective in their use case, they introduce a NL and a PL generator that generate plausible alternatives for randomly selected positions. Then, the CodeBERT model is requested to predict whether a word is original or not. GraphCodeBERT [21] is a model that tries to utilize the information from the code structure. The input of the model contains the comments, the code, and the variables sequence from the code's data flow. The authors train the model with the masked language objective (MLM) applied to the source code and comments, the edge prediction objective, in which they mask edges in the data flow and train the model to predict whether the edge exists, and the node alignment objective, which train the model to predict edges between code tokens and nodes.

2.1.2 Decoder-only

In [7], the authors fine-tune GPT [51] in Python code files collected from GitHub, producing Codex, a model that can create Python code from docstrings and vice versa. They experiment with additional supervised fine-tuning in a corpus that contains only functions to adapt the model to the distribution of the requested tasks, resulting in a model with improved performance. In SYNCHROMESH [47] the authors evaluate the use of zero-shot GPT architectures. Authors at [44] have created a family of models, CodeGen, all trained in The Pile dataset [19], while some are further trained in additional corpora (BigQuery, BigPython), with the next-token prediction objective. They additionally explore the fragmentation of problems into simpler ones with the use of multi-turn prompts, each of which will solve a subproblem. In [43] authors make an effort towards a unification across architecture, objectives, sampling procedures, and data distribution. Even though they do not fully achieve their goal, they present a model, CodeGen2, trained in Stack v1.1 [32] with a mix of span corruption and causal language objective.

2.1.3 Encoder-Decoder

PLBART [1] is a model pretrained with 3 denoising objectives, token masking, token deletion, and token infilling [36]. In PYMT5 [9], authors train a model in Github repositories with the span-masking objective. In CodeT5 [67], except for the masking objective, the authors define 2 identifier-aware objectives to capture the code-specific characteristics. In more detail, they introduce identifier tagging, in which the model is requested to predict whether a token is an identifier or not, and masked identifier prediction where they mask all identifiers in the input code and train the model to predict the value of each unique identifier.

2.1.4 Datasets

The majority of the models created in the last few years use primarily mined code from Github to train their models, such as Stack [32] and CodeSearchNet [27]. For the evaluation of the systems, several benchmarks have been used mostly focusing on the performance of the models in the Python programming language(APPS [26], MBPP [2], HumanEval [7]). A more extensive evaluation benchmark is CodeXGlue [41], a collection of datasets for numerous code-related tasks such as code repair, translation, and summarization, that try to simulate the Glue benchmark [62] used for the evaluation of language models.

Although SQL can be considered a programming language, it is not included in the training or evaluation of most of the code-related models. Exceptions of the above, as far as we know, are SYNCHROMESH [47], in which the model is evaluated in the Spider dataset, resulting in much lower performance than current SOTA, Staqc [71], a large dataset containing mined answers of Python and SQL code snippets with their corresponding questions from Stack Overflow, which has not been used in any of the above systems, and Stack [32] dataset, which contains SQL files, definitely UPDATE, CREATE, DELETE and INSERT statements but an unclear number of SQL queries.

2.2 Text to SQL translation

The most successful existing text-to-SQL systems rely on large language models (LLMs) finetuned mainly with two datasets, WikiSQL [80] and Spider [74], and define problem-specific solutions to tackle the main discovered challenges, namely domain generalization, the translation of complex SQL queries, and schema linking.

2.2.1 Pre-training techniques

Introducing pre-training tasks is a popular approach to improving the 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 [58] created a synthetic dataset and trained a model using 4 objectives (denoising, column prediction, column recovery, and SQL generation) resulting in an encoder that can better represent SQL. SeaD [69] introduces 2 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 [73] 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 [10] introduces 3 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 [77] proposes extra pretraining of the decoder to reduce SQL grammar errors.

All the pretraining methods used seem to have a positive impact on the models' performance but there is not a clear go-to solution among them.

2.2.2 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 [22], RATSQL [63], and LGESQL [5] construct a graph with the database schema and natural language question entities that reference schema elements and use a relation-aware self-attention [57] encoder to capture the relations between input segments. In [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 2 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 [72]. In RASAT [50], they construct a graph similar to RATSQL and create relation embeddings that pass to the multi-head relation-aware self-attention. GRAPHIX-T5 [39] modifies the architecture of the T5 model by introducing a relational graph attention network (RGAT [65]) 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.

2.2.3 Tasks decoupling

Schema linking is one of the main challenges in the task of text-to-SQL. Recently, there is a tendency to unburden the main translation model from the schema-linking problem. SLSQL [35] 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 [38] 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 [49] the translation is broken down into 4 simpler tasks, schema linking, SQL classification and decomposition, SQL generation, and self-correction.

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 [20]. 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 [23] and MIMICSQL [66], which have a size similar to Spider's. Regarding the creation method, most of the datasets were created through crowdsourcing/user studies (Yelp [70], IMDb [70], Scholar [28], Geoquery [75], Advising [13], ATIS [25], Fiben [56]). Additionally, some datasets were automatically created using templates. These datasets include Restaurants [48, 60] and MIMICSQL [66], which also included additional filtering of the produced questions by users. Finally, there are datasets created from user logs. These include

Table 1: Text-to-SQL datasets overview. ”-” denotes unavailable information.

Dataset	NLQ-SQL	Databases	Domains	Category
Academic [37]	196	1	Microsoft Academic Search	Single domain
Advising [13]	4570	1	University courses	
Geoquery [75]	880	1	US geographical facts	
IMDb [70]	131	1	Movies	
Yelp [70]	128	1	Business reviewing	
Scholar [28]	816	1	Academic database	
Atis [25]	5418	1	Air travel information system	
Restaurants [48, 60]	250	1	Info about restaurants in N. California	
Fiben [56]	300	1	Financial	
SEDE [23]	12023	1	Stack Exchange Website	
MIMICSQL [66]	10000	1	Electronic medical records	
WikiSQL [80]	80654	24241	Wikipedia domains	Cross domain
Staqc [71]	119519	-	-	
Spider [74]	10181	200	Wikipedia, college courses, SQL tutorial websites	
KaggleDBQA [33]	272	8	Kaggle datasets in multiple domains	
EHRSQL [34]	24000	2	Electronic medical records	
BIRD [40]	12751	95	Professional domains	
Spider-Syn [17]	8034	200	Spider domains	Perturbed
Spider-realistic [10]	508	20	Spider domains	
Spider-DK [18]	535	-	Spider domains	
ADVETA [46]	-	-	Spider, WikiSQL, WDC domains	
DR Spider [6]	15269	-	Spider domains	
MT-TEQL [42]	62430	2273	Spider domains	
Spider-CG [16]	45599	-	Spider domains	Augmented
GRAPPA synthetic data [73]	-	-	Spider domains	
GAP2SQL synthetic data [58]	30000	-	Spider domains	
SHiP [78]	-	-	Wikitables and spider train domains	

Academic [37], which was generated from logs from the Microsoft Academic Search, and SEDE [23], 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 [56] 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 [71] 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 [33] contains a small number of queries upon real databases from Kaggle. BIRD [40] contains questions over large-scale databases aiming to better represent real use-case scenarios. Finally, EHRSQL [34] 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 [33, 40], 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 [17] is created by replacing schema references in the NL questions with their synonyms in the train and development Spider sets. Spider-Realistic [10] is created by removing or paraphrasing explicit mentions of column names from a subset of NL questions in the Spider development set. MT-TEQL [42] is generated by applying transformation rules in the schema or the utterance of the Spider queries. ADVETA (ADVErsarial Table perturbAtion) [46] is built by applying perturbations in the tables of WikiSQL, Spider, and WTQ. Spider-DK [18] 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 [76].

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 [73] 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 [58] 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 [64]. The method was used to create augmented datasets from the Geoquery [75] and Spider databases and pre-train models in the downstream task. In SHiP [78], 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 [16] 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.

Table 2: Dataset analysis axes

SQL queries	Structural variety, Operator variety, Operator usage, Schema usage, Content usage
Databases	Schema complexity, Schema quality, Database size
NL questions	Schema linkage, Lexical complexity, Syntactic complexity, Readability

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 [13,33,34,40,55,74], the percentage of columns mentioned in the natural language questions [10,33,59], the quality of the natural language queries [42], the number of templates existing in the dataset [13,23,28], 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) [23,66]. Furthermore, dataset analyses 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. Table 2 shows the main axes of our methodology.

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 comprises five axes.

Structural variety. To gain a general view of the variety of the SQL queries, we focus on 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. 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.

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. The report could additionally contain the percentages of the numbers of each operator separately, but we believe that it would not add significant value. Nevertheless, as joins play a significant role in the query complexity, the average number of the join operators is also reported.

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 SQL 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 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 [30] 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 neural model.

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

Question	Rarity	Lexical density
What is the area of California?	0	0.33
What is the total number of patients who had coronary atherosclerotic native vessel?	0.5	0.57
How many such stocks are there whose last traded value does not exceed 1?	0.27	0.78

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 [53]; and (b) Lexical density: the ratio of the content words to the total words of an NL question [29]. 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 [14].

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. 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 GitHub repository[1]. The following plots present the evaluation (test) splits of the datasets. For datasets with no splits, we consider the whole dataset as an evaluation set, while in the case of Spider and BIRD, where the test split is not available, we consider the dev set as the evaluation split. We omit the analysis of the training sets, as in all datasets, except Atis, the distributions in the training set are similar to the evaluation one. The omitted plots can be found in our Github repository. Lastly, in the figures and tables, we use the abbreviations: Geo (Geoquery), MIMIC (MIMICSQL), Restos (Restaurants), K-DBQA (KaggleDBQA).

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, MIMIC-SQL, 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.

[1] https://github.com/athenarc/Experimental-Analysis-of-Text-to-SQL-Benchmarks

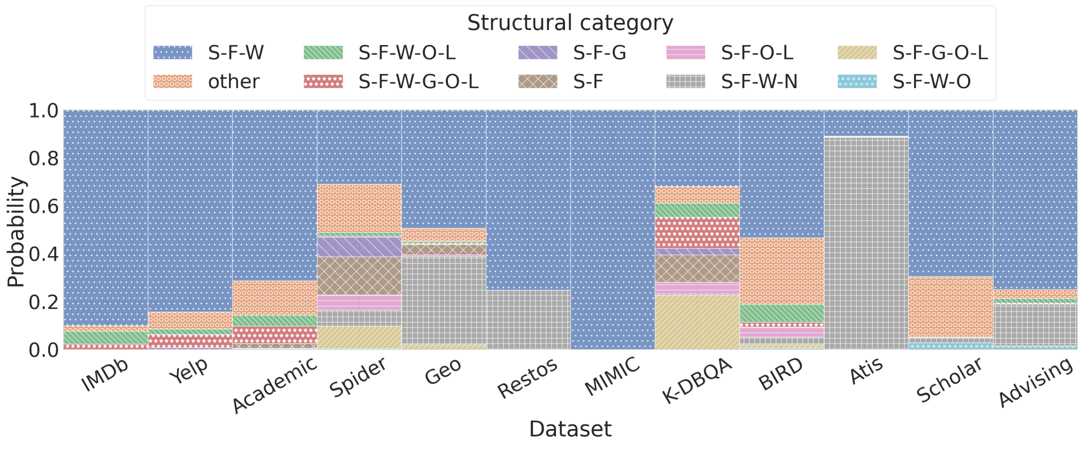


Figure 1: Structural variety in the SQL queries of the datasets. ({S}elect, {F}rom, {W}here, {G}roup by, {O}rder by, {L}imit, {N}esting

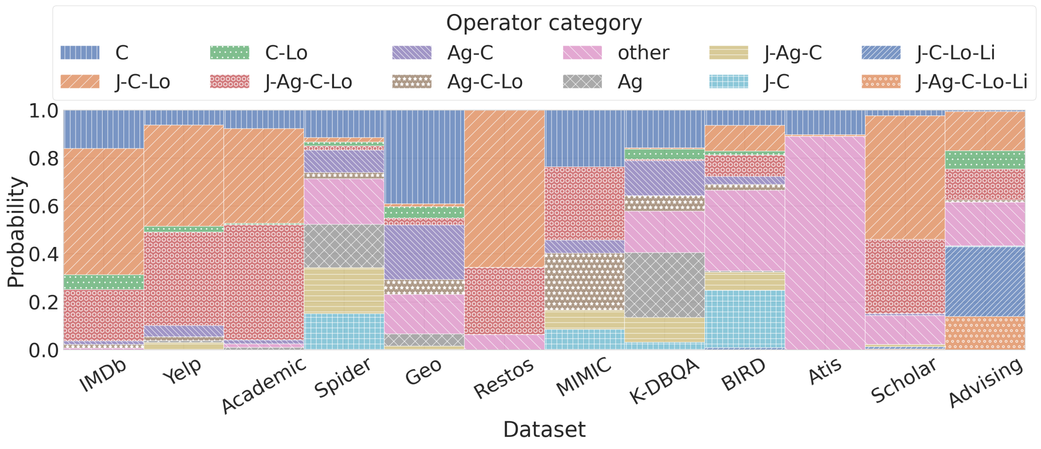
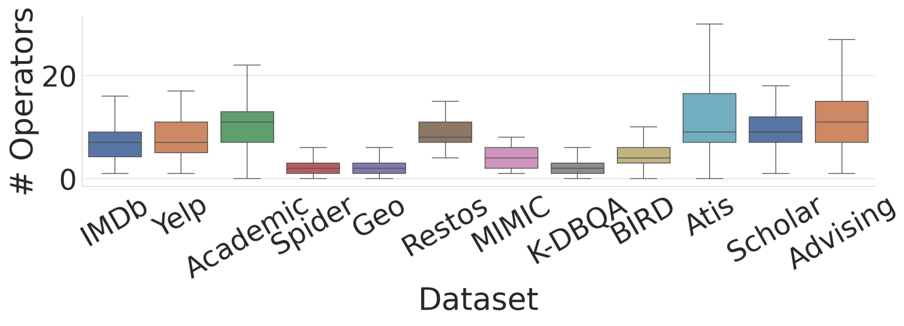


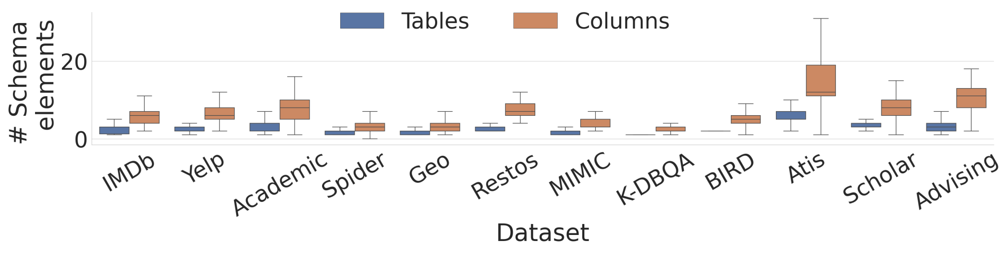
Figure 2: Operator variety in the SQL queries of the datasets. ({J}oin, {Ag}gregate, {C}omparison, {Lo}gical, {Me}mbership, {Li}ke)

5.1.2 Operator variety

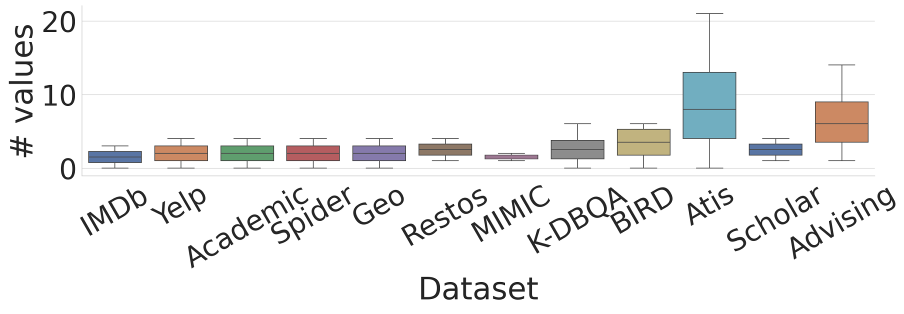
Figure 2 depicts the eleven most common operator type combinations existing in the datasets. Geoquery and Advising are the only single-domain datasets that contain multiple operator categories in a substantial percentage. MIMICSQL offers some operator type variety but it is more imbalanced. The multi-domain datasets have the highest varieties. In the Advising dataset, half of its queries contain `like` operators. *Overall*, the datasets do not cover a broad spectrum of operator combinations, while arithmetic, membership, and like operators appear rather rarily, if not at all.



(a) Operator usage



(b) Schema usage



(c) Content usage

Figure 3: Usage analysis of the SQL queries of the datasets.

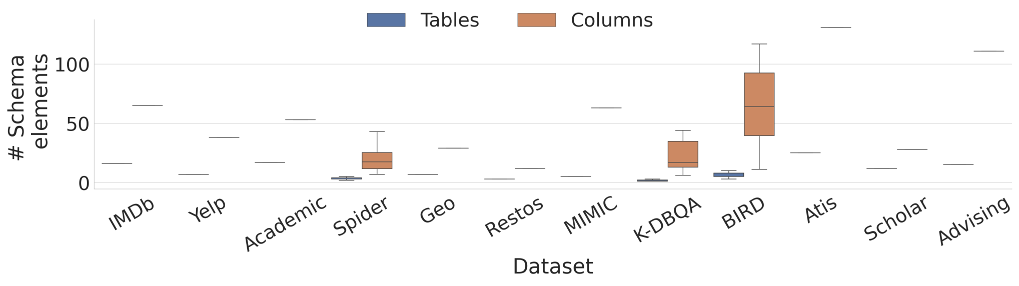


Figure 4: Schema complexity in the databases of the datasets.

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. Atis stands out with the highest operator usage. 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. 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’ only few tables and columns.

5.2 Analysis of the Databases

5.2.1 Schema complexity

Figure 4 shows 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

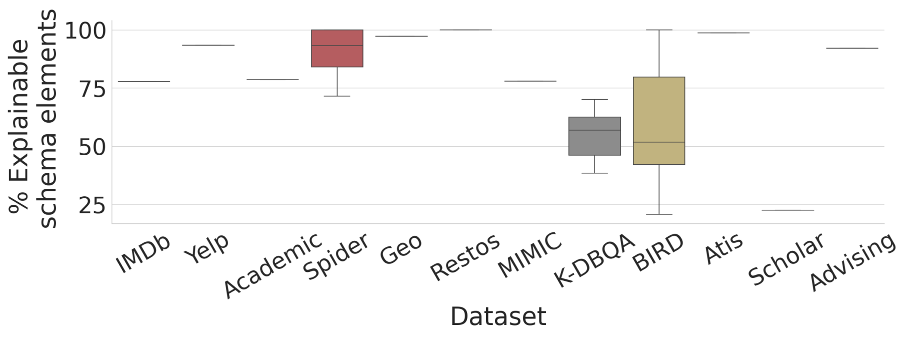


Figure 5: Schema quality in the databases of the datasets.

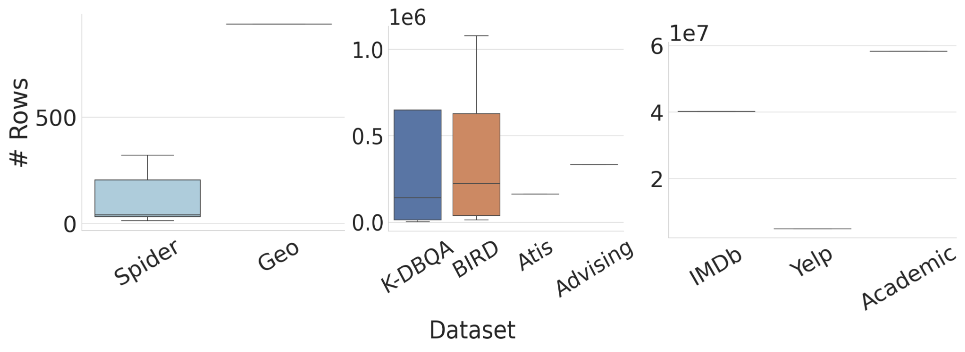


Figure 6: Size of databases in the datasets.

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. Scholar is the only single-domain dataset with the lowest schema quality across all datasets (∼20%) because its column names are often concatenation of multiple words without underscore or camel case (e.g., *citedpaperid*). *Overall*, most datasets use easy 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 the Academic and IMDb followed by Yelp, while 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 [10,35] 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 in terms of syntactic complexity, while BIRD and MIMICSQL have the more complex questions.

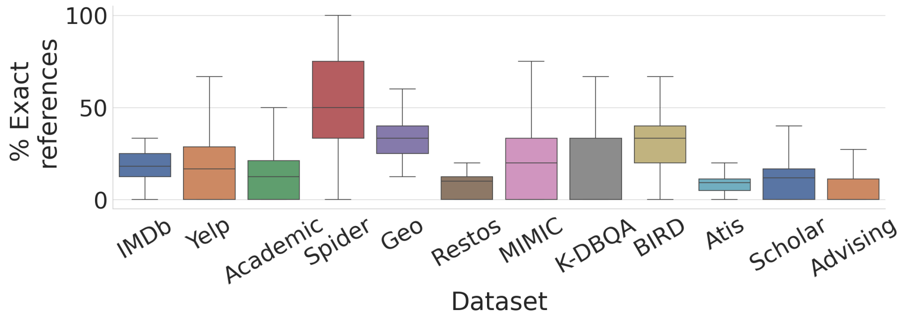


Figure 7: Schema linkage in the NL questions of the datasets.

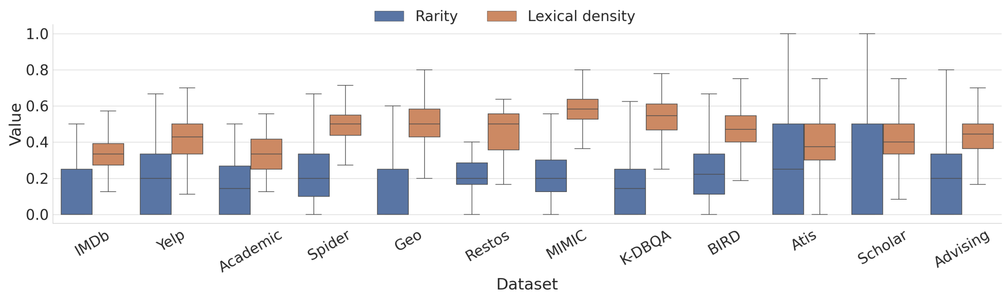


Figure 8: Lexical complexity of the NL questions.

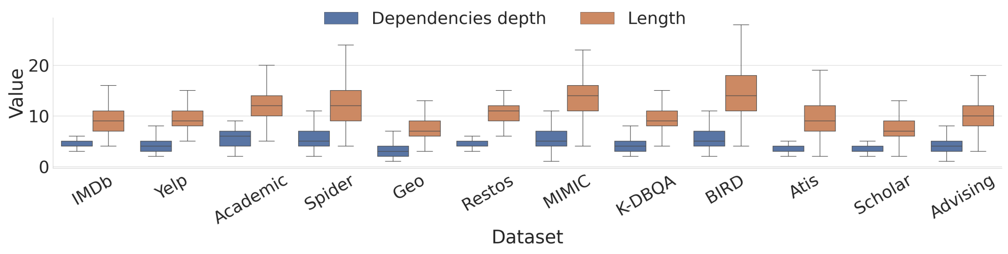


Figure 9: Syntactic complexity of the NL questions.

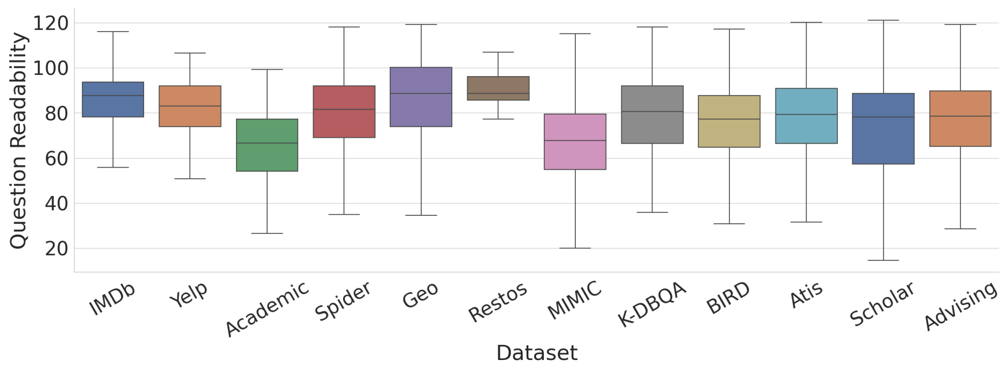


Figure 10: Readability of datasets NL questions.

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.

Overall, NL questions are easily understood by humans.

5.4 Summary

Table 4: Summary of datasets in the analyzed axes. L: Low, M: Medium, H: High.

	SQL Queries					**NLQs**				**Databases**		
Dataset	Structural Variety	Operator Variety	Operator Usage	Schema Usage	Content Usage	Schema linkage	Lexical Compl.	Syntactic Compl.	Read/ty	Schema Compl.	Schema quality	DB Size
Academic	L	L	M	L	M	L	L	H	M	M	M	H
Advising	L	M	M	M	M	L	M	M	M	M	H	M
Geo	H	L	L	L	M	M	L	M	H	M	H	L
IMDb	L	L	M	L	L	L	L	M	H	L	M	H
Yelp	L	L	M	L	L	L	L	M	H	L	H	H
Scholar	M	L	M	L	L	L	M	M	M	M	L	N/A
Atis	H	H	M	M	M	L	L	M	M	M	H	M
Restos	L	L	M	M	M	L	M	M	H	L	H	M
MIMIC	L	L	L	L	L	M	H	H	M	L	M	N/A
Spider	M	L	L	L	L	H	M	H	H	L	M	L
K-DBQA	M	L	L	L	L	L	L	M	M	L	L	M
BIRD	M	L	L	L	L	M	M	H	M	M	M	M

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 create several problems in the models like biased predictions and reduced generalization capabilities [24]. **(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 [27] used for training code-related models contains 2 million training examples, with even larger datasets created after it (Stack [32], The Pile [19]). Lastly, since most of the datasets have been built within the scope of the text-to-SQL task (e.g., through crowdsourcing or researchers' manual work), **(6)** the questions may not represent real use case scenarios, making it difficult to understand the performance that a system would have if we used it, for instance, as an assistant for data analysis.

5.4.2 What are the best datasets for training?

There are two scenarios: **Finetuning**. A critical requirement for a dataset used for training a PLM is to be of substantial size. As a result, most existing datasets cannot be used as a standalone training solution. Additionally, the use of a system in multiple domains requires a multi-domain training dataset. The datasets that meet the size and domain requirements are the most popular ones that are already used for training, namely WikiSQL, Spider and BIRD. We believe that a combination of the existing diverse big datasets (e.g., Spider+BIRD) would be the best strategy for training. On the other hand, the small datasets should be left out of the training set as their value could be higher serving as out-of-distribution evaluation sets.

Prompting. Lately, a popular solution for creating systems for a downstream task is the use of pre-trained large language models with prompting. In this scenario, the dataset requirements are minimized to finding similar examples to the provided one. This means that all datasets can be equally valuable in creating a pool of diverse examples from which the prompt examples will be selected.

5.4.3 What are the best datasets to test the capabilities of a system?

Evaluating on multiple datasets is necessary to measure the coverage of the types of questions a system can support [13]. Therefore, the more datasets used for evaluation the more robust will be the understanding of a system's capabilities. The most valuable datasets in the evaluation process are the most diverse compared to the training datasets, or the ones with unique characteristics. For example, it would be valuable to test the performance of a system trained on Spider, in a dataset like MIMICSQL, which has a database with demanding terminology and it is different from most of the databases existing in Spider. In the same manner, evaluating a system with the Atis dataset, which contains queries with a higher number of filters would be of high value in determining the capabilities of a system.

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 [80] 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 [79], 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 [23], an accuracy metric that considers semantically equivalent queries [31], and a metric called test suite execution accuracy [79] that tests the execution results of the queries over diverse variations of the database contents. Finally, QATCH [45] 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 [15, 55], and the throughput, i.e., the number of queries that can be processed when a maximum number of processes are given [15]. A metric called VES (Valid Efficiency Score) computes the efficiency of the valid predicted SQL queries [40]. A query is valid if its result set aligns with this of the ground-truth SQL query. In this case, efficiency refers to the query's running time.

Although these metrics are valuable components 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 [20,68], 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 [22,31,33,35,42, 63] 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 downfalls, 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

The decision for the set of categories that will achieve the best results, 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 downfalls 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 [13, 23, 28], as 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:

$$avg(Jaccard([select, from, where, nesting\_where\_1,$$
$$nesting\_where\_2], [select, from, where]),$$

$$Jaccard([select, from, where], []),$$
$$Jaccard([select, from], [])) = 0.2$$

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 naïve 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 table names, column names, and values. For models without constant prediction, we do not consider literals and numbers in 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, we could use our metric only in queries that 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 checkpoints of the T5 in the Spider dataset from [54] that made it easy to get the predictions of the models for all datasets. We did not select any LLM, 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

Table 5: Models' evaluation in several text-to-SQL datasets with execution match (EM) and execution accuracy (EX). ({b}ase_lm100k, {l}arge, {l}arge{-lm}100k, {b}ase-lm100k + {P}ICARD (b+P))

T5	Spider		Geo		Atis		Academic		MIMIC		K-DBQA		IMDb		Yelp		Scholar		BIRD		Restos	
	EM	EX	EM	EX	EM	EX	EM	EX	EM	EX	EM	EX	EM	EX	EM	EX	EM	EX	EM	EX	EM	EX
b	59.4	59.3	4.2	16.8	0	0.4	4.6	6.1	2	-	11.3	16.7	11.4	11.4	4.6	7.8	0	-	1.3	4.1	0	7.2
l	67	68.3	10.3	16.4	0	0.4	4.6	4.1	4.3	-	16.7	21	12.9	14.5	4.6	8.5	0	-	1.9	8.2	0	21.6
l-lm	71.1	73	13.8	19.6	0	0.2	3.5	5.1	4.7	-	15.1	21.6	18.3	18.3	5.4	11.7	0	-	2.2	7.6	0	26.4
3B	71.5	72.8	16.8	19	0	1.2	5.1	5.6	8.2	-	18.9	21.6	16	17.5	4.6	11.7	0	-	3.1	9.5	0	4
b+P	66.2	67.4	13.4	32.9	0.9	6.7	5.1	6.1	2.8	-	18.3	24.8	11.4	16	4.6	8.5	0	-	2.7	10	0	0

the PICARD method [54]. 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 [54]. 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 existing 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 reformatting the query. These include:

1. 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.
2. <> operator. We replaced the <> operator with the !=, which was parsable.
3. 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.
4. Backquotes. We replaced backquotes in literals with quotes.
5. `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

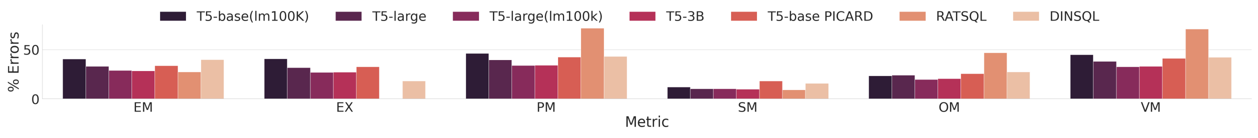


Figure 11: Errors in Spider development set with Exact match (EM), Execution accuracy (EX), Structural match (SM), Operator match (OM) and Variables Match (VM).

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 [10, 13, 17, 18, 45, 46], 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 SFW 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

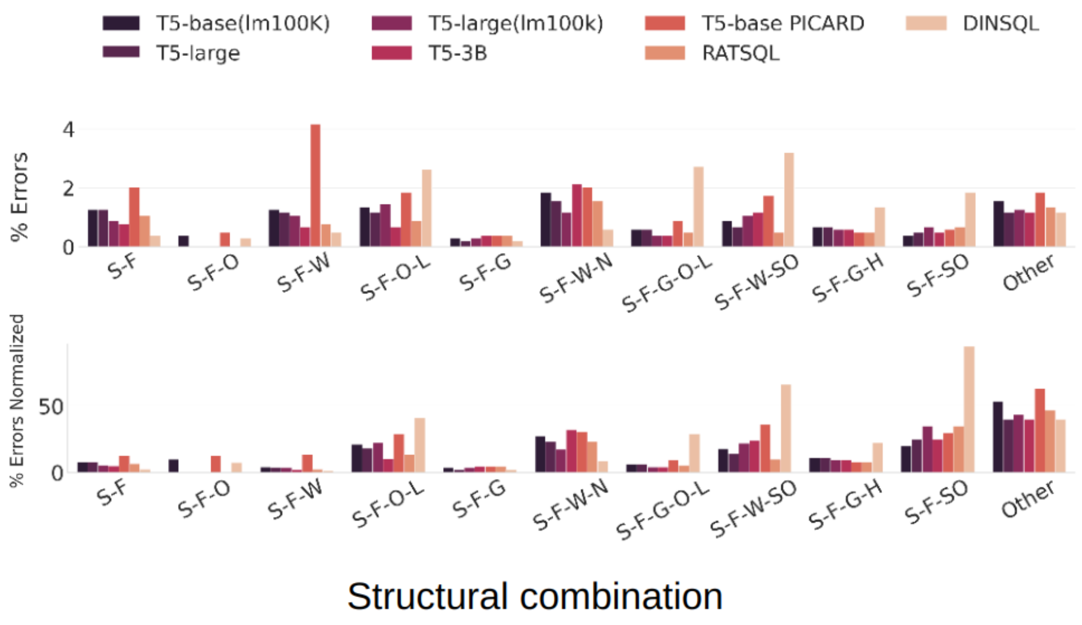


Figure 12: Structural match errors in structural categories.

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.

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

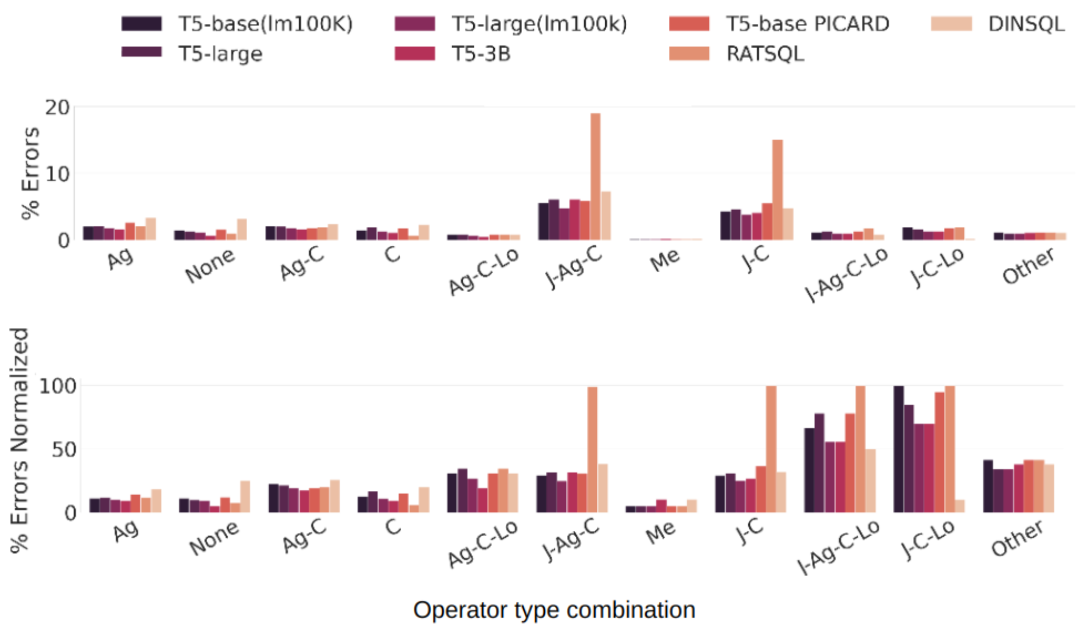


Figure 13: Operator match errors in operator type combinations categories.

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 [63], which uses a task-specific encoder and decoder, and DIN-SQL [49], which is based on GPT-4. For the RATSQL model, we reproduced the results of the RAT-SQL+BERT following the instructions in their repository, while for the DINSQL, 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 DINSQL models remains the same but slightly elevated, as opposed to RATSQL, in which we can see a large increase in errors.

Table 6: Examples of predictions

Dataset	NL Question	SQL query	Prediction
Atis	flights from washington to seattle	`SELECT DISTINCT flight.flight_id FROM flight WHERE (flight.from_airport IN (SELECT airport_service.airport_code FROM airport_service WHERE airport_service.city_code IN (SELECT city.city_code FROM city WHERE city.city_name = 'WASHINGTON')) AND flight.to_airport IN (SELECT airport_service.airport_code FROM airport_service WHERE airport_service.city_code IN (SELECT city.city_code FROM city WHERE city.city_name = 'SEATTLE')))`	`select flight.flight_id from flight join aircraft on flight.aircraft_code = aircraft.aircraft_code where aircraft.aircraft_name = 'washington' and aircraft.aircraft_name ='seattle'`
	i want to go from boston to washington on a saturday	`SELECT DISTINCT flight.flight_id FROM flight WHERE (flight.from_airport IN (SELECT airport_service.airport_code FROM airport_service WHERE airport_service.city_code IN (SELECT city.city_code FROM city WHERE city.city_name = 'BOSTON')) AND (flight.to_airport IN (SELECT airport_service.airport_code FROM airport_service WHERE airport_service.city_code IN (SELECT city.city_code FROM city WHERE city.city_name = 'WASHINGTON')) AND flight.flight_days IN (SELECT days.days_code FROM days WHERE days.day_name = 'SATURDAY')))`	`select airport.city from airport join airport_service on airport.airport_code = 'Boston' and airport. city_code = 'Denver'`
Scholar	who publishes in NIPS ?	`SELECT DISTINCT writes.authorId FROM paper, venue, writes WHERE paper.venueId = venue.venueId AND venue.venueName = 'NIPS' AND paper.paperId = writes.paperId"`	`select t1.authorname from author as t1 join writes as t2 on t1.authorid = t2.authorid join journal as t3 on t2.journalid = t3.journalid where t3.journalname = "NIPS"`
	Fetch me the most cited publications for Artificial Intelligence	`SELECT DISTINCT cite.citedPaperId, count(cite.citingPaperId) FROM cite, paperKeyphrase, keyphrase WHERE cite.citedPaperId = paperKeyphrase.paperId AND paperKeyphrase.keyphraseId = keyphrase.keyphraseId AND keyphrase.keyphraseName = 'Artificial Intelligence' GROUP BY cite.citedPaperId ORDER BY count(cite.citingPaperId) DESC`	`select t1.title, t2.numcitedby from paper as t1 join cite as t2 on t1.paperid = t2.citingpaperid where t1.title = "Artificial Intelligence" group by t2.citedpaperid order by count(*) desc limit 1`
Geo	what is the area of california	SELECT STATEalias0.AREA FROM STATE AS STATEalias0 WHERE STATEalias0.STATE_NAME = ”california”	select area from state where state_name = ”california”
MIMICSQL	find the number of patients diagnosed with mitral valve disorders.	`SELECT COUNT ( DISTINCT DEMOGRAPHIC.SUBJECT_ID ) FROM DEMOGRAPHIC INNER JOIN DIAGNOSES on DEMOGRAPHIC.HADM_ID = DIAGNOSES.HADM_ID WHERE DIAGNOSES.LONG_TITLE = Mitral valve disorders"`	`select count(*) from demographic as t1 join diagnoses as t2 on t1.hadm_id = t2.hadm_id join primary_disease as t3 on t2.primary_disease = t3.primary_disease group by t3.primary_disease having count(*) >= 1`
	what is the total number of patiemts who had crnry athrscl natve vssl?	`SELECT COUNT ( DISTINCT DEMOGRAPHIC.SUBJECT_ID ) FROM DEMOGRAPHIC INNER JOIN DIAGNOSES on DEMOGRAPHIC.HADM_ID = DIAGNOSES.HADM_ID WHERE DIAGNOSES.SHORT_TITLE = "Crnry athrscl natve vssl"`	`select count(*) from crnry athrscl natve vssl`

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 the 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 RATSQL 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 DINSQL, 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

Recognizing the vital role of datasets used for the training and evaluation of text-to-SQL systems play, 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 our error metrics, to detect equivalences and to report false negatives and positives cases. 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.

ABBREVIATIONS - ACRONYMS

NL	Natural Language
T5	Text-to-Text Transfer Transformer
PL	Programming Language
MLM	Masked Language Modeling
SOTA	State Of The Art
LLM	Large Language Model
SADGA	Structure-Aware Dual Aggregation Network
RGAT	Relational Graph Attention Network
ADVETA	Adversarial Table Perturbation
SCFG	Synchronous Context Free Grammar
Geo	Geoquery
MIMIC	MIMICSQL
Restos	Restaurants
K-DBQA	Kaggle DataBases Question Answering
e.g.	from the Latin phrase exempli gratia, meaning “for example.”
i.e.	from the Latin phrase ”id est”, meaning “that is”
etc	et cetera
PLM	Pretrained Language Model
VES	Valid Efficiency Score

GPT	Generative Pretrained Transformer
JSON	JavaScript Object Notation
BERT	Bidirectional Encoder Representations from Transformers
MBPP	Mostly Basic Python Porgramming
GAP	Generation Augmented Pretraining
SeaD	Schema-aware Denoising
GP	Grammar Pretraining
SLSQL	Schema-Linking SQL

REFERENCES

[1] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. Unified pre-training for program understanding and generation. *arXiv preprint arXiv:2103.06333*, 2021.

[2] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[3] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. Tabel: Entity linking in web tables. In *International Semantic Web Conference*, pages 425–441. Springer, 2015.

[4] Ruichu Cai, Jinjie Yuan, Boyan Xu, and Zhifeng Hao. Sadga: Structure-aware dual graph aggregation network for text-to-sql. *Advances in Neural Information Processing Systems*, 34:7664–7676, 2021.

[5] Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. Lgesql: line graph enhanced text-to-sql model with mixed local and non-local relations. *arXiv preprint arXiv:2106.01093*, 2021.

[6] Shuaichen Chang, Jun Wang, Mingwen Dong, Lin Pan, Henghui Zhu, Alexander Hanbo Li, Wuwei Lan, Sheng Zhang, Jiarong Jiang, Joseph Lilien, et al. Dr. spider: A diagnostic evaluation benchmark towards text-to-sql robustness. *arXiv preprint arXiv:2301.08881*, 2023.

[7] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[8] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020.

[9] Colin B Clement, Dawn Drain, Jonathan Timcheck, Alexey Svyatkovskiy, and Neel Sundaresan. Pymt5: multi-mode translation of natural language and python code with transformers. *arXiv preprint arXiv:2010.03150*, 2020.

[10] Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. Structure-grounded pretraining for text-to-sql. *arXiv preprint arXiv:2010.12773*, 2020.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[12] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*, 2020.

[13] Catherine Finegan-Dollak, Jonathan K Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. Improving text-to-sql evaluation methodology. *arXiv preprint arXiv:1806.09029*, 2018.

[14] Rudolf Franz Flesch and Alan J Gould. The art of readable writing. *(No Title)*, 1949.

[15] Han Fu, Chang Liu, Bin Wu, Feifei Li, Jian Tan, and Jianling Sun. Catsql: Towards real world natural language to sql applications. *Proceedings of the VLDB Endowment*, 16(6):1534–1547, 2023.

[16] Yujian Gan, Xinyun Chen, Qiuping Huang, and Matthew Purver. Measuring and improving compositional generalization in text-to-sql via component alignment. *arXiv preprint arXiv:2205.02054*, 2022.

[17] Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R Woodward, Jinxia Xie, and Pengsheng Huang. Towards robustness of text-to-sql models against synonym substitution. *arXiv preprint arXiv:2106.01065*, 2021.

[18] Yujian Gan, Xinyun Chen, and Matthew Purver. Exploring underexplored limitations of cross-domain text-to-sql generalization. *arXiv preprint arXiv:2109.05157*, 2021.

[19] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

[20] Orest Gkini, Theofilos Belmpas, Georgia Koutrika, and Yannis Ioannidis. An in-depth benchmarking of text-to-sql systems. In *Proceedings of the 2021 International Conference on Management of Data*, pages 632–644, 2021.

[21] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, et al. Graphcodebert: Pre-training code representations with data flow. *arXiv preprint arXiv:2009.08366*, 2020.

[22] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. Towards complex text-to-sql in cross-domain database with intermediate representation. *arXiv preprint arXiv:1905.08205*, 2019.

[23] Moshe Hazoom, Vibhor Malik, and Ben Bogin. Text-to-sql in the wild: a naturally-occurring dataset based on stack exchange data. *arXiv preprint arXiv:2106.05006*, 2021.

[24] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284, 2009.

[25] Charles T Hemphill, John J Godfrey, and George R Doddington. 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*, 1990.

[26] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*, 2021.

[27] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*, 2019.

[28] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. Learning a neural semantic parser from user feedback. *arXiv preprint arXiv:1704.08760*, 2017.

[29] Victoria Johansson. Lexical diversity and lexical density in speech and writing: A developmental perspective. *Working papers/Lund University, Department of Linguistics and Phonetics*, 53:61–79, 2008.

[30] George Katsogiannis-Meimarakis and Georgia Koutrika. A survey on deep learning approaches for text-to-sql. *The VLDB Journal*, 2023.

[31] Hyeonji Kim, Byeong-Hoon So, Wook-Shin Han, and Hongrae Lee. Natural language to sql: where are we today? *Proceedings of the VLDB Endowment*, 13(10):1737–1750, 2020.

[32] Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, et al. The stack: 3 tb of permissively licensed source code. *arXiv preprint arXiv:2211.15533*, 2022.

[33] Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. Kaggledbqa: Realistic evaluation of text-to-sql parsers. *arXiv preprint arXiv:2106.11455*, 2021.

[34] Gyubok Lee, Hyeonji Hwang, Seongsu Bae, Yeonsu Kwon, Woncheol Shin, Seongjun Yang, Minjoon Seo, Jong-Yeup Kim, and Edward Choi. Ehrsql: A practical text-to-sql benchmark for electronic health records. *Advances in Neural Information Processing Systems*, 35:15589–15601, 2022.

[35] Wenqiang Lei, Weixin Wang, Zhixin Ma, Tian Gan, Wei Lu, Min-Yen Kan, and Tat-Seng Chua. Re-examining the role of schema linking in text-to-sql. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6943–6954, 2020.

[36] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[37] Fei Li and Hosagrahar V Jagadish. Nalir: an interactive natural language interface for querying relational databases. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 709–712, 2014.

[38] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence (AAAI)*, 2023.

[39] Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. *arXiv preprint arXiv:2301.07507*, 2023.

[40] Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiaxi Yang, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, et al. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *arXiv preprint arXiv:2305.03111*, 2023.

[41] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. Codexglue: A machine learning benchmark dataset for code understanding and generation. *arXiv preprint arXiv:2102.04664*, 2021.

[42] Pingchuan Ma and Shuai Wang. Mt-teql: evaluating and augmenting neural nlidb on real-world linguistic and schema variations. *Proceedings of the VLDB Endowment*, 15(3):569–582, 2021.

[43] Erik Nijkamp, Hiroaki Hayashi, Caiming Xiong, Silvio Savarese, and Yingbo Zhou. Codegen2: Lessons for training llms on programming and natural languages. *arXiv preprint arXiv:2305.02309*, 2023.

[44] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*, 2022.

[45] Simone Papicchio, Paolo Papotti, and Luca Cagliero. Qatch: Benchmarking sql-centric tasks with table representation learning models on your data. *Advances in Neural Information Processing Systems*, 36, 2024.

[46] Xinyu Pi, Bing Wang, Yan Gao, Jiaqi Guo, Zhoujun Li, and Jian-Guang Lou. Towards robustness of text-to-sql models against natural and realistic adversarial table perturbation. *arXiv preprint arXiv:2212.09994*, 2022.

[47] Gabriel Poesia, Oleksandr Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. Synchromesh: Reliable code generation from pre-trained language models. *arXiv preprint arXiv:2201.11227*, 2022.

[48] Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 149–157, 2003.

[49] Mohammadreza Pourreza and Davood Rafiei. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *arXiv preprint arXiv:2304.11015*, 2023.

[50] Jiexing Qi, Jingyao Tang, Ziwei He, Xiangpeng Wan, Chenghu Zhou, Xinbing Wang, Quanshi Zhang, and Zhouhan Lin. Rasat: Integrating relational structures into pretrained seq2seq model for text-to-sql. *arXiv preprint arXiv:2205.06983*, 2022.

[51] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[52] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67, 2020.

[53] John AS Read. *Assessing vocabulary*. Cambridge university press, 2000.

[54] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. *arXiv preprint arXiv:2109.05093*, 2021.

[55] Jaydeep Sen, Chuan Lei, Abdul Quamar, Fatma Özcan, Vasilis Efthymiou, Ayushi Dalmia, Greg Stager, Ashish Mittal, Diptikalyan Saha, and Karthik Sankaranarayanan. Athena++ natural language querying for complex nested sql queries. *Proceedings of the VLDB Endowment*, 13(12):2747–2759, 2020.

[56] Jaydeep Sen, Fatma Ozcan, Abdul Quamar, Greg Stager, Ashish Mittal, Manasa Jammi, Chuan Lei, Diptikalyan Saha, and Karthik Sankaranarayanan. Natural language querying of complex business intelligence queries. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1997–2000, 2019.

[57] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155*, 2018.

[58] Peng Shi, Patrick Ng, Zhiguo Wang, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Cicero Nogueira dos Santos, and Bing Xiang. Learning contextual representations for semantic parsing with generation-augmented pre-training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 13806–13814, 2021.

[59] Alane Laughlin Suhr, Kenton Lee, Ming-Wei Chang, and Pete Shaw. Exploring unexplored generalization challenges for cross-database semantic parsing. 2020.

[60] Lappoon R Tang and Raymond Mooney. Automated construction of database interfaces: Intergrating statistical and relational learning for semantic parsing. In *2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 133–141, 2000.

[61] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[62] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[63] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *arXiv preprint arXiv:1911.04942*, 2019.

[64] Bailin Wang, Wenpeng Yin, Xi Victoria Lin, and Caiming Xiong. Learning to synthesize data for semantic parsing. *arXiv preprint arXiv:2104.05827*, 2021.

[65] Kai Wang, Weizhou Shen, Yunyi Yang, Xiaojun Quan, and Rui Wang. Relational graph attention network for aspect-based sentiment analysis. *arXiv preprint arXiv:2004.12362*, 2020.

[66] Ping Wang, Tian Shi, and Chandan K Reddy. Text-to-sql generation for question answering on electronic medical records. In *Proceedings of The Web Conference 2020*, pages 350–361, 2020.

[67] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859*, 2021.

[68] Nathaniel Weir, Prasetya Utama, Alex Galakatos, Andrew Crotty, Amir Ilkhechi, Shekar Ramaswamy, Rohin Bhushan, Nadja Geisler, Benjamin Hättasch, Steffen Eger, et al. Dbpal: A fully pluggable nl2sql training pipeline. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 2347–2361, 2020.

[69] Kuan Xuan, Yongbo Wang, Yongliang Wang, Zujie Wen, and Yang Dong. Sead: End-to-end text-to-sql generation with schema-aware denoising. *arXiv preprint arXiv:2105.07911*, 2021.

[70] Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. Sqlizer: query synthesis from natural language. *Proceedings of the ACM on Programming Languages*, 1(OOPSLA):1–26, 2017.

[71] Ziyu Yao, Daniel S Weld, Wei-Peng Chen, and Huan Sun. Staqc: A systematically mined question-code dataset from stack overflow. In *Proceedings of the 2018 World Wide Web Conference*, pages 1693–1703, 2018.

[72] Pengcheng Yin and Graham Neubig. A syntactic neural model for general-purpose code generation. *arXiv preprint arXiv:1704.01696*, 2017.

[73] Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, Richard Socher, and Caiming Xiong. Grappa: Grammar-augmented pre-training for table semantic parsing. *arXiv preprint arXiv:2009.13845*, 2020.

[74] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 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.

[75] John M Zelle and Raymond J Mooney. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, pages 1050–1055, 1996.

[76] Chen Zhao, Yu Su, Adam Pauls, and Emmanouil Antonios Platanios. Bridging the generalization gap in text-to-sql parsing with schema expansion. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5568–5578, 2022.

[77] Liang Zhao, Hexin Cao, and Yunsong Zhao. Gp: Context-free grammar pre-training for text-to-sql parsers. *arXiv preprint arXiv:2101.09901*, 2021.

[78] Yiyun Zhao, Jiarong Jiang, Yiqun Hu, Wuwei Lan, Henry Zhu, Anuj Chauhan, Alexander Li, Lin Pan, Jun Wang, Chung-Wei Hang, et al. Importance of synthesizing high-quality data for text-to-sql parsing. *arXiv preprint arXiv:2212.08785*, 2022.

[79] Ruiqi Zhong, Tao Yu, and Dan Klein. Semantic evaluation for text-to-sql with distilled test suites. *arXiv preprint arXiv:2010.02840*, 2020.

[80] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.