Text-to-SQL in the Wild: A Naturally-Occurring Dataset Based on Stack Exchange Data

Moshe Hazoom[1] **Vibhor Malik**[2] **Ben Bogin**[1,3]
[1]Rupert [2]Columbia University [3]Tel Aviv University
{ben,moshe}@hirupert.com, malik.vibhor@columbia.edu

Abstract

Most available semantic parsing datasets, comprising of pairs of natural utterances and logical forms, were collected solely for the purpose of training and evaluation of natural language understanding systems. As a result, they do not contain any of the richness and variety of natural-occurring utterances, where humans ask about data they need or are curious about. In this work, we release SEDE, a dataset with 12,023 pairs of utterances and SQL queries collected from real usage on the Stack Exchange website. We show that these pairs contain a variety of real-world challenges which were rarely reflected so far in any other semantic parsing dataset, propose an evaluation metric based on comparison of partial query clauses that is more suitable for real-world queries, and conduct experiments with strong baselines, showing a large gap between the performance on SEDE compared to other common datasets.

1 Introduction

Semantic parsing, the task of mapping natural language into logical forms that can be executed on a database or knowledge graph, has been studied mostly on *academic datasets*, where both the utterances and the queries were written as part of a dataset collection process (Hemphill et al., 1990; Zelle and Mooney, 1996; Yu et al., 2018), and not in a natural process where users ask questions about data they need or are curious about. As a result, these datasets generally do not contain any of the richness and diversity of natural-occurring utterances, even if the data on which the questions are asked about is collected from a real-world source.

Recent methods (Wang et al., 2020a; Herzig et al., 2020; Yu et al., 2021) have significantly improved results on such academic datasets: state-of-the-art models have yield impressive results of over 70%, for example, on Spider (Yu et al., 2018) in a challenging cross-domain setup, where models are trained and tested on different domains, and up to 80%-90% (Nguyen et al., 2021; Zhao and Huang, 2014) on single-domain datasets such as ATIS (Hemphill et al., 1990) and GeoQuery (Zelle and Mooney, 1996). While the cross-domain, zero-shot setup introduces many generalization challenges such as non-explicit mentioning of column names and domain-specific phrases (Suhr et al., 2020; Deng et al., 2020), we argue that even in the easier single-domain setup, it is still unclear how well state-of-the-art models generalize to the challenges that arise from real-world utterances and queries.

In this work, we take a significant step towards evaluation of Text-to-SQL models in a real-world setting, by releasing SEDE: a dataset comprised of 12,023 complex and diverse SQL queries and their natural language titles and descriptions, written by

Title: *Questions which attract bad answers*
Description: *posts which have attracted significantly more controversial or bad answers than good ones*

```
SELECT p.Id as [Post Link], p.Score from (
 SELECT p.ParentId, count(*) as ContACnt from (
  SELECT PostId,
   up = sum(case when VoteTypeId = 2 then 1
     else 0 end),
   down = sum(case when VoteTypeId = 3 then 1
     else 0 end)
   FROM Votes v join Posts p on p.Id = v.PostId
   WHERE VoteTypeId in (2,3) and PostTypeId = 2
   group by PostId
 ) as ContA
 JOIN posts p on ContA.PostId = p.Id
 WHERE down > (up / ##UVDVRatio:int##) and
 (down + up) > ##MinVotes:int##
 GROUP BY p.ParentId
) as ContQ
JOIN posts p on ContQ.ParentId = p.Id
WHERE ContQ.ContACnt > (p.AnswerCount / 2) and
p.AnswerCount > 1
ORDER BY Score desc
```

Table 1: Example from SEDE for a title and description given by the user, together with the SQL query that the user has written.

Proceedings of the 1st Workshop on Natural Language Processing for Programming (NLP4Prog 2021), pages 77–87

real users of the Stack Exchange Data Explorer out of a natural interaction.

In Table 1 we show an example for a SQL query from SEDE, with its title and description. It introduces several challenges that have not been commonly addressed in currently available datasets: comparison between different subsets, complex usage of 2 nested sub-queries and an under-specified question, which doesn't state what *"significantly more"* means (solved in this case with an input parameter, `##UVDVRation##`).

Compared to other Text-to-SQL datasets, we show that SEDE contains at least 10 times more SQL queries templates (queries after canonization and anonymization of values) than other datasets, and has the most diverse set of utterances and SQL queries (in terms of 3-grams) out of all single-domain datasets. We manually analyze a sample of examples from the dataset and list the introduced challenges, such as under-specification, usage of parameters in queries, dates manipulation and more.

We also address the challenging problem of evaluating naturally-occurring Text-to-SQL datasets. In academic datasets, standard evaluation metrics such as denotation accuracy and exact comparison of SQL components can often be used with relative success, but we found this to be a greater challenge in SEDE. Denotation accuracy is inaccurate for under-specified utterances, where any single clause not mentioned in the question could entirely change execution results, while exact match comparison of SQL components (e.g. comparing all `SELECT`, `WHERE`, `GROUP BY` and `ORDER BY` clauses) are often too strict when queries are highly complex. While solving these issues still remains an open problem, to at least partially address them we propose to measure a softer version of the exact match metric, PCM-F1, based on partially extracted queries components, and show that this metric gives a better indication of models' performance than common metrics, which yield a score that is close to 0.

Finally, we test strong baselines on our dataset, and show that even models that get strong results on Spider's development set (63.2% Exact-Match, 86.3% PCM-F1), perform poorly on our dataset, with a PCM-F1 value of 50.6%. We hope that the unique and challenging properties exhibited in SEDE[1] will pave a path for future work on generalization of Text-to-SQL models in real world setups.

2 Background

In the past decades, a broad selection of datasets have been used as benchmarks for semantic parsing: ATIS (Hemphill et al., 1990), GeoQuery (Zelle and Mooney, 1996), Restaurants (Tang and Mooney, 2000), Scholar (Iyer et al., 2017), Academic (Li and Jagadish, 2014), Yelp and IMDB (Yaghmazadeh et al., 2017), Advising (Finegan-Dollak et al., 2018), WikiSQL (Zhong et al., 2017), Spider (Yu et al., 2018), WikiTableQuestions (Pasupat and Liang, 2015), Overnight (Wang et al., 2015) and more. However, the utterances and queries in all of these academic datasets, to the best of our knowledge, were collected explicitly for the purpose of evaluating semantic parsing models, usually with the help of crowd-sourcing (even though in most cases questions are asked about real data). As such, these academic datasets were generated in an artificial process, which often introduces various simplifications and artifacts which are not seen in real-life.

Utterance-Query alignment One arising issue with this artificial process is that utterances are often aligned to their SQL queries counterparts, such that the columns and the required computations are explicitly mentioned (Suhr et al., 2020; Deng et al., 2020). In contrast, natural utterances often do not explicitly mention these, since the schema of the database is not necessarily known to the asking user (for example, the question from Spider "titles of films that include 'Deleted Scenes' in their special feature section" might have been more naturally phrased as "films with deleted scenes" in a real-world setting).

Well-specified utterances Furthermore, the utterances in academic datasets are mostly well-specified, whereas in contrast, natural utterances are often under-specified or ambiguous; they could be interpreted in different ways and in turn be mapped to different SQL queries. Consider the example in Table 1: the definition of *"bad answers"* is not well-defined, and in fact could be subjective. Since under-specified utterances, by definition, can not always be answered correctly, any human or machine attempting to answer such a question would have to either make an assumption

[1] Our dataset and code to run all experiments and metrics is available at https://github.com/hirupert/sede.

on the requirement (usually based on previously seen examples) or ask follow-up questions in an interactive setting (Yao et al., 2019; Elgohary et al., 2020, 2021).

Scope Last, in academic datasets the utterances are usually written by crowd-sourced workers, asked to provide utterances on various data domains which they do not necessarily need or are interested with. As a result, the utterances and queries are often not very diverse or realistic, are inherently limited in scope, and might not reflect real-world utterances.

3 Stack Exchange Data Explorer

To introduce a realistic Text-to-SQL benchmark, we gather SQL queries together with their titles and descriptions from a naturally occurring dataset: the Stack Exchange Data Explorer. Stack Exchange is an online question & answers community, with over 3 million questions asked. The Data Explorer[2] allows any user to query the database of Stack Exchange with T-SQL (a SQL variant) to answer any question they are curious about. The database schema[3] is spread across 29 tables and 211 columns. Common utterance topics are published posts, comments, votes, tags, awards, etc.

Any query that users run in the data explorer is logged, and users are able to save the queries with a title and description for future use by the public. All of these logs are available online, and Stack Exchange have agreed to release these queries, together with their title, description and other metadata. We publish our clean version of this log, which contains 12,023 samples, of which a subset of 1,714 examples is verified by humans to be correct and is used for validation and test. In this section, we explain the cleaning process, analyze the characteristics of the dataset and compare it to other semantic parsing datasets.

3.1 Data cleaning

The raw aggregated log contains over 1.6 million queries, however in its raw form many of the rows are duplicated or contain unusable queries or titles. The reason for this large difference between the original data size and the cleaned version is that any time that the author of the query executes it, an entry is saved to the log. This introduces two issues: First, many of the queries are not complete, since they were executed before writing the entire query (these incomplete queries are usually valid and executable, but are missing some expressions with respect to the given title and description). Second, after completing the writing of a correct query, users often keep changing and executing the query, but they do not update the title and description accordingly.

To alleviate these issues, we write rule-based filters that remove bad queries/descriptions pairs with high precision. For example, we filter out examples with numbers in the description, if these numbers do not appear in the query (refer to the preprocessing script in the repository for the complete list of filters and the number of examples each of them filter). Whenever a query has multiple versions due to multiple executions, we take the last executed query which passed all filters. After this filtering step, we are left with 12,309 examples.

Using these filters cleans most of the noise, but not all of it. To complete the cleaning process, we manually go over the examples in the validation and test sets, and either filter-out wrong examples or perform minimal changes to either the utterances or the queries (for example, fix a wrong textual value) to ensure that models are evaluated with correct data. Out of the 2,000 examples that we have evaluated, we have kept 1,024 and fixed 690[4], leading to a total of 1,714 validated examples which we use for validation and test. While we do not perform verification on the training set, the verification procedure on the validation set allows us to estimate that most of the queries (85.7%) are either entirely accurate or need just a minimal change to be entirely accurate. For example, when the utterance is "users in Brazil" while the matching query contains the expression: `WHERE users.location like %russia%` we either change the utterance to "users in russia" or change the expression to `WHERE users.location like %Brazil%`. The final number of all training, validation and test examples is 12,023.

3.2 Dataset Characteristics

In this sub-section, we quantify and analyze the introduced challenges in SEDE, compared to other commonly used semantic parsing datasets.

First, we manually analyze a sample of 100 ex-

[2]Publicly available at https://data.stackexchange.com/

[3]https://tinyurl.com/sedeschema

[4]We publish both the original and the fixed examples

Category	Dataset			Example test cases	
	SEDE	Spider	ATIS	Title	SQL query
Under specification and Hidden assumptions	87	14	15	*User List: Highest downvotes per day ratio with minimum downvotes*	`WHERE id <> -1`
Parameters	40	0	0	*Rollbacks by a certain user*	`WHERE UserId = @UserId`
Window functions	8	0	0	*List of users in the Philippines.*	`DENSE_RANK() OVER (ORDER BY Reputation DESC)`
Dates manipulation	15	0	0	*Quickest new contributor answers to new contributor questions*	`DATEDIFF(s, Q.CreationDate, A.CreationDate)`
Numerical computations and text manipulation	35	0	0	*Average Number of Views per Tag*	`sum(p.ViewCount)/count(*)`
DECLARE/WITH	11	0	0	*Rollbacks by a certain user*	`DECLARE @UserId AS int = ##UserId:int##`
CASE	10	0	0	*Questions and answers per year*	`CASE WHEN Score < 0 THEN 1 ELSE 0 END`

Table 2: Dataset characteristics comparison of randomly selected 100 samples among SEDE and other popular Text-to-SQL datasets.

amples from SEDE and define 7 categories of introduced challenges. To quantify how often each of these concepts appear in SEDE in comparison to other datasets (SPIDER and ATIS), we sample a subset of equal size from each of the other datasets and count the appearances of these concepts. The analysis is shown in Table 2. Next, we describe each of these concepts.

Under specification and Hidden assumptions Utterances in SEDE are often under-specified, that is, they could be interpreted in different ways. For example, when users write *"top users"*, they might refer to users with the most reputation, but also to users that have written the most answers. Likewise, when users write *"last 500 posts"* they might expect to get just the title field of the posts, but possibly also IDs and dates. Similarly, query authors often add various *assumptions* to the queries which are not mentioned in the questions, because they require some knowledge of the available data. For example, they might filter out a special "Community" user in StackExchange, which should not be accounted for in computation of votes. We consider an utterance/query pair to be under-specified or contain an hidden assumption whenever the query contains an expression in any of the SQL clauses (`SELECT`, `WHERE`, etc.) which is *not* specified in the utterance, or where it is specified in an ambiguous way.

Parameters In some cases, query authors can address under-specified utterances by letting the user fill in the under-specified parameters, which are marked in SEDE with either two hashtags (#) on each side of the parameter name, optionally including the required value type (int, string, etc.) and a default value (e.g. `##UserId:int##`), or using a declared variable using SQL syntax (e.g. `@UserId`). For example, in Table 1, the parameter `##UVDVRatio:int##` is used to indicate that the user should fill in an integer to specify the ratio that *"significantly more"* refers to. More broadly, parameters are also helpful for re-usability, allowing users unfamiliar with a query to effortlessly change some values in it.

Window functions Window functions operate on a set of rows and return a single value for each row from the underlying query, thus allowing to perform various aggregation operators without the need for a separate aggregation query. Window functions are often used in SEDE to report percentiles of a specific value in a row, by using operators such as `ROW_NUMBER() OVER`, `NTILE`, `TOP(X) PERCENT`, etc.

Dates manipulation Queries in SEDE sometimes contain dates arithmetic expressions. See the example category query in Table 2: this expression calculates the difference in seconds from the time the question was created to the time the answer was created.

Numerical computations and text manipulation Queries can perform any arbitrary numerical computation and text manipulation. The computations in SEDE often include multiple nested operators including rounding and conversions to float, for example: `ROUND(CAST(Main.Total AS FLOAT) / Meta.Total, 2) AS 'Ratio'`. Queries can also contain text manipulation such as concatenation, for example: `'stackoverflow.com/tags/' + t.tagName + '/info' as [Link]` which builds a URL from a tag name.

Dataset	Unique Utterances	Unique Queries	Average unique Tables / uttr	Utterance 3-gram	SQL 3-gram	Avg. nesting Level	Unique Templates	Average unique Queries / template
Spider	8,034	4491	1.71	41.7K	25.2K	1.15	1,059	7.6
WikiSQL	**80,654**†	**77,840**†	1†	**375K**	**209K**	1†	488†	165.3†
Academic	196†	185†	3.0†	<1K	<1K	1.04†	92†	2.1†
Advising	4,570†	211†	3.0†	20K	11.2K	1.18†	174†	20.3†
ATIS	5,280†	947†	**3.8**†	13.2K	5.8K	1.39†	751†	7.0†
GeoQuery	877†	246†	1.1†	1.5K	1.4K	**2.03**†	98†	8.9†
IMDB	131†	89†	1.9†	<1K	<1K	1.01†	52†	2.5†
Restaurants	378†	23†	2.3†	<1K	<1K	1.17†	17†	22.2†
Scholar	817†	193†	3.2†	2.6K	2.2K	1.02†	146†	5.6†
Yelp	128†	110†	1.0†	<1K	<1K	1.0†	89†	1.4†
SEDE	12,023	11,767	2.14	42.6K	173K	1.28	**10,664**	**1.1**

Table 3: Comparison of different semantic parsing datasets (for Spider, analysis is performed on training and validation sets only). † denotes that numbers are reported from Finegan-Dollak et al. (2018). Average Unique Queries / template denotes the number of different SQL queries per template, thus lower means more diversity in the dataset. Datasets above dashed line are cross-domain, and below it are single-domain.

DECLARE/WITH SQL queries can be written as a procedural process, where multiple commands are executed sequentially. Query authors can store values in simple variables with `DECLARE`, but more importantly, they can store complete "views" of tables with the `WITH` command. While these commands do not add any expressivity (that is, any query can be written without these commands), they allow writing more clear and concise queries with less nested expressions.

CASE The `CASE` clause is similar to an *if-then-else* statement of any programming language, and is often used to either make the query more readable (e.g. by returning names of values instead of integers) or to perform conditional logic. For example, the clause in Table 2 (last row) counts negative scores using `CASE` function.

Comparison In Table 2 we see that a vast majority of SEDE is not well-specified, which implies that in order for Text-to-SQL models to work robustly in a real-world setting, it should identify cases of ambiguity and possibly proceed with follow-up questions. We see that the rest of the concepts appear in 10% to 40% of SEDE examples, whereas these concepts are not exhibited in any other analyzed dataset.

Next, we show a comparison of quantifiable metrics of popular Text-to-SQL datasets compared to SEDE in Table 3. We see that SEDE is the largest dataset in terms of unique utterances and queries out of all single-domain datasets. To compare diversity and scope, we also measure the number of unique 3-grams for both the utterances and the queries, and see that SEDE has a very diverse set of SQL 3-grams, with almost 6 times the number of the next follower, Spider, and only 17% less than WikiSQL, which is 6.6 bigger in terms of queries. The number of utterance 3-gram is the second largest, after WikiSQL. Last, we count the number of unique *SQL templates*, as defined in Finegan-Dollak et al. (2018): we anonymize the values and group all canonized queries. We see that SEDE has more than 10 times templates than the follower Spider, and that the average number of queries per template is the lowest. We also see that SEDE is third in terms of average nesting level, after ATIS and GeoQuery.

3.3 Limitations

We note that in order to simulate the most realistic setting, an ideal Text-to-SQL dataset would include questions asked by users which are completely unaware of the schema, which are not SQL-savy, and that the person asking the question would be different than the person answering it. While this is not the case in SEDE, we believe its setting is still significantly more realistic that other datasets.

4 Evaluation

Semantic parsing models are usually evaluated in two different forms: execution accuracy and logical forms accuracy. In this section, we show why using any of these metrics is difficult with complex queries such as those in SEDE, and propose a more loose metric for evaluation of models.

Execution accuracy This metric is measured by executing both the predicted and gold query against a dataset, and considers the query to be correct if the two output results are the same (or similar enough). While this metric appears to be exactly

what we want to optimize (yielding a query the outputs a correct output), it does not necessarily cope well with two challenges: spurious queries and under-specified questions. Spurious queries are incorrect queries (with respect to the given question) that happen to result in a correct answer, thus leading to a false-positive count. The problem of spuriousness can be addressed by executing the predicted query on modified versions of the dataset, as proposed in Zhong et al. (2020). The second challenge, evaluating under-specification, is arguably harder to address, as mentioned in Subsection 3.2. For example, consider a question that asks for *"the top 1% active users"*. This question does not specify which columns should be returned, how the rows should be ordered, and how does one measure "being active". As such, a query could be correct with respect to some interpretation, yet its execution result might be different than the execution result of the given gold query.

Logical form accuracy Instead of comparing execution results, another frequent approach is to simply perform a textual comparison between the predicted and gold queries. When comparing SQL queries, it is common to perform a more loose comparison that does not consider the order of appearances of different clauses (e.g. it shouldn't matter which `WHERE` expression is written first), as performed in Spider (Yu et al., 2018). However, as discussed in Zhong et al. (2020), even this looser metric leads to false-negative measures, since multiple queries can all be correct with respect to an utterance, but written in various different manners. Due to the richness of SQL queries in SEDE, its extended scope and the fact that queries are written by many different authors, in our case this problem deteriorates: queries can be written in a substantial number of ways. For example, a query that contains a `WITH` statement could yield exactly the same result without it, by including a nested `FROM` clause instead.

4.1 Sub-tree elements matching

In this work, in order to alleviate the aforementioned issues with exact-match logical form evaluation, we loosen it so that models can get partial scores if at least some part of their predicted expressions are found in the gold query. We do this by parsing both the predicted query and the gold query, comparing different parts of the two parsed trees and aggregating the scores into a single metric, as defined next. We term this metric **P**artial **C**omponent **M**atch **F1** (PCM-F1).

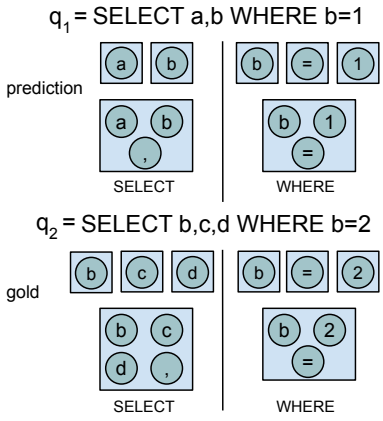

Figure 1: An example for sub-tree matching.

Our proposed metric is based on the "Component Matching" metric which is used in Spider's evaluation (Yu et al., 2019), except that we use a parser that supports a large variety of queries (Spider's parser only supports specific types of queries), define how to compute the metric in a general way (not specific to any SQL-specific clause) and aggregate (average) the F1 scores into a single value, as defined next.

We first use an open-source SQL parser, JSqlParser,[5] to parse a given SQL query q into a tree, and extract a set of elements for each of its sub-trees, considering a sub-tree only if all of its leaves are terminal values in the query (similar to extracting constituents from a parse tree). For example, as can be seen in Figure 1, the predicted query q_1 has 7 relevant sub-trees (marked in rectangles). The sub-tree which represents the expression `b=1` contains four elements: `b`, `=`, `1` and `b=1`. We then split these sets into different categories, based on the SQL query part that the root of the original sub-tree belonged to, for each of the following categories: $C = \{$`SELECT`, `TOP`, `FROM`, `WHERE`, `GROUPBY`, `HAVING`, `ORDERBY`$\}$. We denote all sets of elements for a query q in a category $c \in C$ as $s_c(q)$. For example, as can be seen in Figure 1, the clause $s_{\texttt{SELECT}}(q_1)$ yields 3 sub-trees. Given a predicted query q_p and a gold query q_g, we compute the average F1 metric of all aligned pairs of sets $s_c(q_p)$ and $s_c(q_g)$:

$$\text{PCM-F1}(q_p, q_g) = \frac{1}{\mid C \mid} \sum_{c \in C} F1\left(s_c(q_p), s_c(q_g)\right)$$

[5] https://github.com/JSQLParser/JSqlParser

Model	Spider-Dev				
	PCM-F1	PCM-EM	PCM-F1-NoValues	PCM-EM-NoValues	EM
RAT (Wang et al., 2020b)	88.1	37.3	91.3	69.0	69.7†
RAT+GAP (Shi et al., 2020)	89.3	39.0	92.6	71.8	71.8†
T5-Base *with schema*	85.7	56.7	85.9	57.2	57.6
T5-Large *with schema*	**86.3**	**61.2**	**86.6**	**62.6**	**63.2**

Table 4: Results on Spider with various metrics. While we do not focus on Spider, we show our results for comparison of the model and evaluation metric with a known benchmark. † denotes reported numbers from Spider's official leaderboard. PCM-F1-`NoValues` and PCM-EM-`NoValues` are modified versions of PCM-F1 and PCM-EM, respectively, such that all values in the SQL are anonymized and the `ON` clause is ignored, in order to compare with Spider's official Exact-Match (EM) metric.

where F1 score is the harmonic mean of the precision and recall of the predicted sub-trees $s_c(q_p)$ with respect to the gold sub-trees $s_c(q_g)$. If for some category c, we get that $s_c(q_p)$ is an empty set but $s_c(q_g)$ is not, or vice-versa, we set $F1 = 0.0$ for that category.

Consider Figure 1 for an example. $s_{\texttt{SELECT}}(q_1)$ has 3 sub-trees while the gold category $s_{\texttt{SELECT}}(q_2)$ has 4 sub-trees. The predicted `SELECT` clause has 2 wrong sub-trees (`a` and `a,b`) leading to a precision $p = \frac{1}{3}$, and 2 missing elements leading to a recall $r = \frac{1}{4}$. Similarly, the `WHERE` clause gets a precision of $p = \frac{1}{2}$ and a recall of $r = \frac{1}{2}$. Thus, we get $F1 = 0.285$ for `SELECT` and $F1 = 0.5$ for `WHERE`, leading to a final score PCM-F1 $= 0.392$.

4.2 Limitations

Parsing Queries JSqlParser could only parse 93.2% of the validation SQL queries in SEDE, and 92.5% of the test queries. For that reason, for evaluation we only use the subset of queries which we can parse and evaluate [6]. During evaluation, if the predicted query was not parsed, it receives a score of 0. Note that this does not affect training.

False negatives We note that our metric does not address at all the issue of false negatives - in fact, since it's a looser metric than the Exact Match metric, it is actually more prone to produce false negative outcomes. For SEDE, this issue could be mitigated by improving the similarity function that compares two queries, or by adapting the execution accuracy method in a way that will be less sensitive to instances of under-specification. We leave this challenge for future work.

[6] While we did not use the rest of the validation queries, we have released them in the dataset for future use, assuming at least some of them are valid queries.

Model	SEDE-Dev		SEDE-Test	
	PCM-F1	PCM-EM	PCM-F1	PCM-EM
T5-Base	46.8	**4.0**	49.4	3.6
with schema	46.4	3.4	48.9	**4.5**
T5-Large	**48.2**	**4.0**	50.6	4.1
with schema	47.1	3.7	**51.0**	3.3

Table 5: Results on SEDE development and test sets.

5 Experiments

In this section, we describe our experimental setup, test how strong baselines perform on SEDE, and analyze their errors.

5.1 Experimantal Setup

Most models in the Spider leaderboard[7] use a grammar-based decoder designed for Spider, and as a result, they cannot be used as-is on SEDE, which uses a larger grammar. Thus, following Shaw et al. (2020), we use a general-purpose pre-trained sequence-to-sequence model, T5 (Raffel et al., 2020), which was shown to be competitive with Spider's state-of-the-art models.

Since all queries in SEDE come from a single schema which is seen during training time, it is not clear if allowing the model to access the schema during encoding and decoding is helpful. We thus experiment with two versions. In the first one, T5, the input is simply the utterance $\bar{u}$. In the second, T5 *with schema*, the input is the utterance $\bar{u}$ followed by a separator token, and then the serialized schema. We follow Suhr et al. (2020) and serialize the schema by listing all tables in the schema and all the columns for each table, with a separator token between each column and table. Naturally, we did not evaluate T5 (without schema) on Spider since encoding the schema is crucial in a zero-shot

[7] https://yale-lily.github.io/spider

Category & Utterance	**Gold**	**Predicted**
Under-specification and Hidden assumptions *Positive scored questions without answers for c++ tags*	`SELECT id as "" Post Link "", *` `FROM posts` `WHERE` `  answercount = 0` `  AND tags NOT LIKE '%c++%'` `  AND score > 0` `  AND ClosedDate is null` `ORDER BY score DESC`	`SELECT id as [Post Link], tags,` `  score, viewcount, CreationDate` `FROM posts` `WHERE` `  tags NOT LIKE '%c++%'` `  AND answercount = 0` `  AND posttypeid = 1` `  AND score > 0` `  AND CreationDate > '2018-01-01'` `ORDER BY CreationDate DESC`
Dates Manipulation *Percentage of votes - depending on day after posting (only questions)*	`SELECT` `  DATEDIFF(day, p.CreationDate,` `  v.CreationDate) AS Days,` `  COUNT(v.Id) AS Count,` `  COUNT(v.Id) * 100.0` `  / SUM(COUNT(v.Id)) OVER ()` `  AS Percentage ...`	`SELECT` `  DATEDIFF(day, p.CreationDate,` `  v.CreationDate) AS Days,` `  COUNT(v.Id) AS Count,` `  COUNT(v.Id) * 100.0` `  / COUNT(v.Id)` `  AS Percentage ...`
Parameters *Top users in a tag by score and answer count*	`SELECT TOP 100 ...` `WHERE ...` `  AND t.TagName = 'tagName'`	`SELECT TOP ##num?100## Users.id` `  AS [User Link], ...` `WHERE tags.TagName = '##tagname##'`

Table 6: Error analysis of gold queries vs. predicted queries for some selected dataset characteristics mentioned in 3. For brevity, in some of the examples we show only relevant parts of the query.

setup. We perform textual pre-processing to the queries in SEDE before training (i.e. remove non UTF-8 characters and SQL comments, normalize spaces and new lines, normalize apostrophes, remove comments, etc.). We show results for experiments considering the titles alone, and ignore their given description, which are given in 14.6% of the examples. We have found that if we concatenate the description to the title, we get slightly worse results.

We use the SentencePiece (Kudo and Richardson, 2018) tokenizer, with its default vocabulary, for all models. We fine-tune the model to minimize the token-level cross-entropy loss against the gold SQL query for 60 epochs with the AdamW (Loshchilov and Hutter, 2019) optimizer and a learning rate of $5e^{-5}$. We choose the best model based on the performance on the validation set for each dataset, using Exact-Match (EM) for Spider and PCM-F1 for SEDE. For inference, we use beam-search (of size 6) and choose the highest-probability generated SQL query. We show results for both T5-Base and T5-Large.

For each experiment we measure PCM-F1 together with a modified version of it, PCM-EM (PCM exact match), that returns an accuracy of 1 for a given prediction if and only if the PCM-F1 value for that prediction is 1. For Spider, we use the officially provided script to measure the EM metric.

5.2 Main Results

We show experiments results for SEDE in Table 5 and for Spider in Table 4. The results indicate that the performance gap between SEDE and Spider is large: while T5-Large reaches a score of 63.2 EM on Spider's validation set, not very far from the state-of-the-art (a difference of 8.6 points), and a PCM-F1 of 86.3, when trained on SEDE, it only receives 48.2 and 50.6 PCM-F1 on the validation and test set of SEDE, respectively. This supports our main claim, that single-schema datasets could still impose a substantial challenge when tested in a realistic setup. We also notice in Table 4 that large improvements in EM do not necessarily imply a large increase in PCM-F1, since PCM-F1 numbers are already high for Spider in any of the tested models, implying that the model is generating SQL queries that are close to the exact gold SQL, only different by a small change (e.g. value or column name).

Comparing experiments with and without encoding the schema shows that encoding the schema does not significantly improve results in this single-domain setup. We also observe that PCM-EM is close to 0 in all experiments, supporting our motivation to create a loosened evaluation metric.

5.3 PCM-F1 Validation

In order to validate the correctness of our proposed evaluation metric, we compare PCM-EM with the

more established EM metric of Spider. There are two differences in the way EM is calculated compared to PCM-EM: (1) EM anonymizes all values in the queries and (2) EM ignores the `ON` expressions in the `JOIN` clauses. For those reasons, we define PCM-F1-NoValues and PCM-EM-NoValues, modified versions of PCM-F1 and PCM-EM, respectively, such that all values in the SQL are anonymized and the `ON` expressions are ignored. Table 4 shows that EM and PCM-EM-NoValues are only different by up to 0.7 points for all models, showing that PCM-F1 is well calibrated with Spider's EM.

5.4 Error Analysis

Next, we analyze errors and successful outputs of the model. Table 6 shows examples of gold vs. predicted queries by our model, with respect to some of the introduced challenges mentioned in 3.2.

We can see from the first example that the model is often wrong whenever the question is not specified well: In this example, this happens in the `SELECT`, `WHERE` and `ORDER` fields. In the `SELECT` clause, the model predicts extra columns in comparison to the gold query, most likely as it has learned to do so for similar questions. In addition, since the desired order of the results are not mentioned in the utterance, it leads to a different predicted `ORDER BY` clause. A hidden assumption the author had added to the query is taking into account only open questions (i.e. questions with no close date: `ClosedDate is null`). The model, which could not deduce this assumption from the utterance alone, predicts a wrong filter expression `CreationDate > '2018-01-01'`.

The second example shows how the model correctly uses the `DATEDIFF` function to manipulate dates, although it predicted a wrong computation of the percentage (i.e. without the `SUM` function).

The last example shows how the model generates a SQL query with parameters, for the number of required users (with a predicted default value of 100) and for the tag name. In this case, the predicted query is possibly better than the gold one as it uses a reusable parameter instead of a fixed one.

6 Conclusion

In this work, we take a significant step towards improving and evaluating Text-to-SQL models in a real world setting, by releasing SEDE, a dataset comprised of real-world complex and diverse SQL queries with their utterances, naturally written by real users. We show that there's a large gap between the performance of strong Text-to-SQL baselines on SEDE compared to the commonly studied dataset Spider, and hope that the release of this challenging dataset will encourage research on improving generalization for real-world SQL prediction.

Acknowledgments We thank Kevin Montrose and the rest of the Stack Exchange team for providing the raw query log.

References

Xiang Deng, Ahmed Hassan Awadallah, Chris Meek, Alex Polozov, Huan Sun, and Matthew Richardson. 2020. Structure-grounded pretraining for text-to-sql. Technical Report 2010.12773, Arxiv.

Ahmed Elgohary, Saghar Hosseini, and Ahmed Hassan Awadallah. 2020. Speak to your parser: Interactive text-to-sql with natural language feedback.

Ahmed Elgohary, Christopher Meek, Matthew Richardson, Adam Fourney, Gonzalo Ramos, and Ahmed Hassan Awadallah. 2021. NL-EDIT: Correcting semantic parse errors through natural language interaction. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5599–5610, Online. Association for Computational Linguistics.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-SQL evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360, Melbourne, Australia. Association for Computational Linguistics.

Charles T. Hemphill, John J. Godfrey, and George R. Doddington. 1990. The ATIS spoken language systems pilot corpus. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27,1990*.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. 2020. TaPas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333, Online. Association for Computational Linguistics.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. *CoRR*, abs/1704.08760.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Fei Li and H. V. Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *Proc. VLDB Endow.*, 8(1):73–84.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization.

Phuong Minh Nguyen, Vu Tran, and Minh Le Nguyen. 2021. Phrasetransformer: Self-attention using local context for semantic parsing.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1470–1480, Beijing, China. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2020. Compositional generalization and natural language variation: Can a semantic parsing approach handle both?

Peng Shi, Patrick Ng, Zhiguo Wang, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Cicero Nogueira dos Santos, and Bing Xiang. 2020. Learning contextual representations for semantic parsing with generation-augmented pre-training.

Alane Suhr, Ming-Wei Chang, Peter Shaw, and Kenton Lee. 2020. Exploring unexplored generalization challenges for cross-database semantic parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8372–8388, Online. Association for Computational Linguistics.

Lappoon R. Tang and Raymond J. Mooney. 2000. Automated construction of database interfaces: Integrating statistical and relational learning for semantic parsing. EMNLP '00, page 133–141, USA. Association for Computational Linguistics.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020a. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, Online. Association for Computational Linguistics.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020b. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers.

Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1332–1342, Beijing, China. Association for Computational Linguistics.

Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. Sqlizer: Query synthesis from natural language. *Proc. ACM Program. Lang.*, 1(OOPSLA).

Ziyu Yao, Yu Su, Huan Sun, and Wen-tau Yih. 2019. Model-based interactive semantic parsing: A unified framework and a text-to-SQL case study. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5447–5458, Hong Kong, China. Association for Computational Linguistics.

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, bailin wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, richard socher, and Caiming Xiong. 2021. GraPPa: Grammar-augmented pre-training for table semantic parsing. In *International Conference on Learning Representations*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2019. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, AAAI'96, page 1050–1055. AAAI Press.

Kai Zhao and Liang Huang. 2014. Type-driven incremental semantic parsing with polymorphism.

Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic evaluation for text-to-SQL with distilled test suites.

In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 396–411, Online. Association for Computational Linguistics.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning.