Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task

Tao Yu **Rui Zhang** **Kai Yang** **Michihiro Yasunaga**
Dongxu Wang **Zifan Li** **James Ma** **Irene Li**
Qingning Yao **Shanelle Roman** **Zilin Zhang** **Dragomir R. Radev**
Department of Computer Science, Yale University
{tao.yu, r.zhang, k.yang, michihiro.yasunaga, dragomir.radev}@yale.edu

Abstract

We present *Spider*, a large-scale, complex and cross-domain semantic parsing and text-to-SQL dataset annotated by 11 college students. It consists of 10,181 questions and 5,693 unique complex SQL queries on 200 databases with multiple tables, covering 138 different domains. We define a new complex and cross-domain semantic parsing and text-to-SQL task where different complex SQL queries and databases appear in train and test sets. In this way, the task requires the model to generalize well to both new SQL queries and new database schemas. Spider is distinct from most of the previous semantic parsing tasks because they all use a single database and the exact same programs in the train set and the test set. We experiment with various state-of-the-art models and the best model achieves only 14.3% exact matching accuracy on a database split setting. This shows that Spider presents a strong challenge for future research. Our dataset and task are publicly available at https://yale-lily.github.io/spider.

1 Introduction

Semantic parsing (SP) is one of the most important tasks in natural language processing (NLP). It requires both understanding the meaning of natural language sentences and mapping them to meaningful executable queries such as logical forms, SQL queries, and Python code.

Recently, some state-of-the-art methods with Seq2Seq architectures are able to achieve over 80% exact matching accuracy even on some complex benchmarks such as ATIS and GeoQuery. These models seem to have already solved most problems in this field.

However, previous tasks in this field have a simple but problematic task definition because most of these results are predicted by semantic "matching"

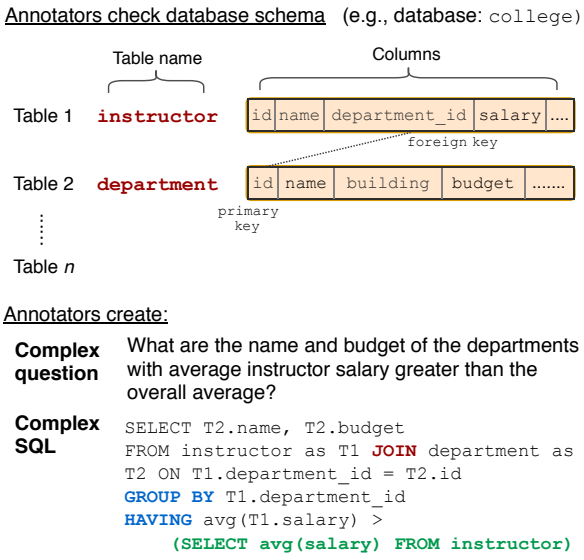

Figure 1: Our corpus annotates complex questions and SQLs. The example contains joining of multiple tables, a GROUP BY component, and a nested query.

rather than semantic parsing. Existing datasets for SP have two shortcomings. First, those that have complex programs (Zelle and Mooney, 1996; Li and Jagadish, 2014; Yaghmazadeh et al., 2017a; Iyer et al., 2017) are too small in terms of the number of programs for training modern data-intensive models and have only a single dataset, meaning that the same database is used for both training and testing the model. More importantly, the number of logic forms or SQL labels is small and each program has about 4-10 paraphrases of natural language problem to expand the size of the dataset. Therefore, the exact same target programs appear in both the train and test sets. The models can achieve decent performances even on very complex programs by memorizing the patterns of question and program pairs during training and decoding the programs exactly the same way as it saw in the training set during testing. Finegan-Dollak et al. (2018) split the dataset by programs so that no two identical programs would be in both

the train and test sets. They show that the models built on this question-splitting data setting fail to generalize to unseen programs. Second, existing datasets that are large in terms of the number of programs and databases such as WikiSQL (Zhong et al., 2017) contain only simple SQL queries and single tables. In order to test a model's real semantic parsing performance on unseen complex programs and its ability to generalize to new domains, an SP dataset that includes a large amount of complex programs and databases with multiple tables is a must.

However, compared to other large, realistic datasets such as ImageNet for object recognition (Deng et al., 2009) and SQuAD for reading comprehension (Rajpurkar et al., 2016), creating such SP dataset is even more time-consuming and challenging in some aspects due to the following reasons. First, it is hard to find many databases with multiple tables online. Second, given a database, annotators have to understand the complex database schema to create a set of questions such that their corresponding SQL queries cover all SQL patterns. Moreover, it is even more challenging to write different complex SQL queries. Additionally, reviewing and quality-checking of question and SQL pairs takes a significant amount of time. All of these processes require very specific knowledge in databases.

To address the need for a large and high-quality dataset for a new complex and cross-domain semantic parsing task, we introduce *Spider*, which consists of 200 databases with multiple tables, 10,181 questions, and 5,693 corresponding complex SQL queries, all written by 11 college students spending a total of 1,000 man-hours. As Figure 1 illustrates, given a database with multiple tables including foreign keys, our corpus creates and annotates complex questions and SQL queries including different SQL clauses such as joining and nested query. In order to generate the SQL query given the input question, models need to understand both the natural language question and relationships between tables and columns in the database schema.

In addition, we also propose a new task for the text-to-SQL problem. Since Spider contains 200 databases with foreign keys, we can split the dataset with complex SQL queries in a way that no database overlaps in train and test, which overcomes the two shortcomings of prior datasets, and defines a new semantic parsing task in which the model needs to generalize not only to new programs but also to new databases. Models have to take questions and database schemas as inputs and predict unseen queries on new databases.

To assess the task difficulty, we experiment with several state-of-the-art semantic parsing models. All of them struggle with this task. The best model achieves only 14.3% exact matching accuracy in the database split setting. This suggests that there is a large room for improvement.

2 Related Work and Existing Datasets

Several semantic parsing datasets with different queries have been created. The output can be in many formats, e.g., logic forms. These datasets include ATIS (Price, 1990; Dahl et al., 1994), GeoQuery (Zelle and Mooney, 1996), and JOBS (Tang and Mooney, 2001a). They have been studied extensively (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005; Wong and Mooney, 2007; Das et al., 2010; Liang et al., 2011; Banarescu et al., 2013; Artzi and Zettlemoyer, 2013; Reddy et al., 2014; Berant and Liang, 2014; Dong and Lapata, 2016). However, they are domain specific and there is no standard label guidance for multiple SQL queries.

Recently, more semantic parsing datasets using SQL as programs have been created. Iyer et al. (2017) and Popescu et al. (2003a) labeled SQL queries for ATIS and GeoQuery datasets. Other existing text-to-SQL datasets also include Restaurants (Tang and Mooney, 2001b; Popescu et al., 2003a), Scholar (Iyer et al., 2017), Academic (Li and Jagadish, 2014), Yelp and IMDB (Yaghmazadeh et al., 2017b), Advising (Finegan-Dollak et al., 2018), and WikiSQL (Zhong et al., 2017). These datasets have been studied for decades in both the NLP community (Warren and Pereira, 1982; Popescu et al., 2003b, 2004; Li et al., 2006; Giordani and Moschitti, 2012; Wang et al., 2017; Iyer et al., 2017; Zhong et al., 2017; Xu et al., 2017; Yu et al., 2018; Huang et al., 2018; Wang et al., 2018; Dong and Lapata, 2018; McCann et al., 2018) and the Database community (Li and Jagadish, 2014; Yaghmazadeh et al., 2017b). We provide detailed statistics on these datasets in Table 1.

Most of the previous work train their models without schemas as inputs because they use a single database for both training and testing. Thus,

they do not need to generalize to new domains. Most importantly, these datasets have a limited number of labeled logic forms or SQL queries. In order to expand the size of these datasets and apply neural network approaches, each logic form or SQL query has about 4-10 paraphrases for the natural language input. Most previous studies follow the standard question-based train and test split (Zettlemoyer and Collins, 2005). This way, the exact same target queries (with similar paraphrases) in the test appear in training set as well. Utilizing this assumption, existing models can achieve decent performances even on complex programs by memorizing database-specific SQL templates. However, this accuracy is artificially inflated because the model merely needs to decide which template to use during testing. Finegan-Dollak et al. (2018) show that template-based approaches can get even higher results. To avoid getting this inflated result, Finegan-Dollak et al. (2018) propose a new, program-based splitting evaluation, where the exact same queries do not appear in both training and testing. They show that under this framework, the performance of all the current state-of-the-art semantic parsing systems drops dramatically even on the same database, indicating that these models fail to generalize to unseen queries. This indicates that current studies in semantic parsing have limitations.

We also want the model to generalize not only to unseen queries but also to unseen databases. Zhong et al. (2017) published the WikiSQL dataset. In their problem definition, the databases in the test set do not appear in the train or development sets. Also, the task needs to take different table schemas as inputs. Therefore, the model has to generalize to new databases. However, in order to generate 80654 questions and SQL pairs for 24241 databases, Zhong et al. (2017) made simplified assumptions about the SQL queries and databases. Their SQL labels only cover single `SELECT` column and aggregation, and `WHERE` conditions. Moreover, all the databases only contain single tables. No `JOIN`, `GROUP BY`, and `ORDER BY`, etc. are included.

Recently, researchers have constructed some datasets for code generation including IFTTT (Quirk et al., 2015), DJANGO (Oda et al., 2015), HEARTHSTONE (Ling et al., 2016), NL2Bash (Lin et al., 2018), and CoNaLa (Yin et al., 2018). These tasks parse natural language descriptions into a more general-purpose programming language such as Python (Allamanis et al., 2015; Ling et al., 2016; Rabinovich et al., 2017; Yin and Neubig, 2017).

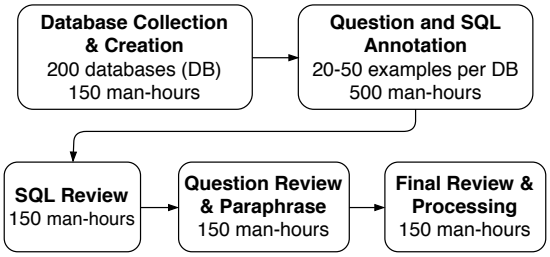

Figure 2: The annotation process of our Spider corpus.

3 Corpus Construction

All questions and SQL queries were written and reviewed by 11 computer science students who were native English speakers.As illustrated in Figure 2, we develop our dataset in five steps, spending around 1,000 hours of human labor in total: §3.1 Database Collection and Creation, §3.2 Question and SQL Annotation, §3.3 SQL Review, §3.4 Question Review and Paraphrase, §3.5 Final Question and SQL Review.

3.1 Database Collection and Creation

Collecting databases with complex schemas is hard. Although relational databases are widely used in industry and academia, most of them are not publicly available. Only a few databases with multiple tables are easily accessible online.

Our 200 databases covering 138 different domains are collected from three resources. First, we collected about 70 complex databases from different college database courses, SQL tutorial websites, online csv files, and textbook examples. Second, we collected about 40 databases from the DatabaseAnswers[1] where contains over 1,000 data models across different domains. These data models contain only database schemas. We converted them into SQLite, populated them using an online database population tool[2], and then manually corrected some important fields so that the table contents looked natural. Finally, we created the remaining 90 databases based on WikiSQL. To ensure the domain diversity, we select about 500 tables in about 90 different domains to create these 90 databases. To create each database, we chose several related tables from WikiSQL

[1] http://www.databaseanswers.org/
[2] http://filldb.info/

dev or test splits, and then created a relational database schema with foreign keys based on the tables we selected. We had to create some intersection tables in order to link several tables together. For most other cases, we did not need to populate these databases since tables in WikiSQL are from Wikipedia, which already had real world data stored.

We manually corrected some database schemas if they had some column names that did not make sense or missed some foreign keys. For table and column names, it is common to use abbreviations in databases. For example, 'student_id' might be represented by 'stu_id'. For our task definition, we manually changed each column name back to regular words so that the system only handled semantic parsing issues.

3.2 Question and SQL Annotation

For each database, we ask eight computer science students proficient in SQL to create 20-50 natural questions and their SQL labels. To make our questions diverse, natural, and reflective of how humans actually use databases, we did not use any template or script to generate question and SQL queries. Our annotation procedure ensures the following three aspects.

A) SQL pattern coverage. We ensure that our corpus contains enough examples for all common SQL patterns. For each database, we ask annotators to write SQL queries that cover all the following SQL components: `SELECT` with multiple columns and aggregations, `WHERE`, `GROUP BY`, `HAVING`, `ORDER BY`, `LIMIT`, `JOIN`, `INTERSECT`, `EXCEPT`, `UNION`, `NOT IN`, `OR`, `AND`, `EXISTS`, `LIKE` as well as nested queries. The annotators made sure that each table in the database appears in at least one query.

B) SQL consistency. Some questions have multiple acceptable SQL queries with the same result. However, giving totally different SQL labels to similar questions can hinder the training of semantic parsing models. To avoid this issue, we designed the annotation protocol so that all annotators choose the same SQL query pattern if multiple equivalent queries are possible. More detail is explained in our appendix.

C) Question clarity. We did not create questions that are (1) vague or too ambiguous, or (2) require knowledge outside the database to answer.

First, ambiguous questions refer to the questions that do not have enough clues to infer which columns to return and which conditions to consider. For example, we would not ask "What is the most popular class at University X?" because the definition of "popular" is not clear: it could mean the rating of the class or the number of students taking the course. Instead, we choose to ask "What is the name of the class which the largest number of students are taking at University X?". Here, "popular" refers to the size of student enrollment. Thus, the "student_enrollment" column can be used in condition to answer this question. We recognize that ambiguous questions appear in real-world natural language database interfaces.

We agree that future work needs to address this issue by having multi-turn interactions between the system and users for clarification. However, our main aim here is to develop a corpus to tackle the problem of handling complex queries and generalizing across databases without multi-turn interactions required, which no existing semantic parsing datasets could do. Moreover, the low performances of current state-of-the-art models already show that our task is challenging enough, without ambiguous questions. In addition, questions are required to contain the specific information to return. Otherwise, we don't know if class id is also acceptable in the previous case. Most of the questions in the existing semantic parsing datasets are ambiguous. This is not a serious problem if we use one single dataset because we have enough data domain specific examples to know which columns are the default. However, it would be a serious problem in cross domain tasks since the default return values differ cross domain and people.

Second, humans sometimes ask questions that require common sense knowledge outside the given database. For instance, when people ask "Display the employee id for the employees who report to John", the correct SQL is

```
SELECT employee_id
FROM employees
WHERE manager_id = (
SELECT employee_id
FROM employees
WHERE first_name = 'John')
```

which requires the common knowledge that "X reports to Y" corresponds to an "employee-

manager" relation. we do not include such questions and leave them as a future research direction.

Annotation tools We open each database on a web-based interface powered by the sqlite_web[3] tool. It allows the annotators to see the schema and content of each table, execute SQL queries, and check the returned results. This tool was extremely helpful for the annotators to write executable SQL queries that reflect the true meaning of the given questions and return correct answers.

3.3 SQL Review

Once the database is labeled with question-query pairs, we ask a different annotator to check if the questions are clear and contain enough information to answer the query. For a question with multiple possible SQL translations, the reviewers double check whether the SQL label is correctly chosen under our protocol. Finally, the reviewers check if all the SQL labels in the current database cover all the common SQL clauses.

3.4 Question Review and Paraphrase

After SQL labels are reviewed, native English speakers review and correct each question. They first check if the question is grammatically correct and natural. Next, they make sure that the question reflects the meaning of its corresponding SQL label. Finally, to improve the diversity in questions, we ask annotators to add a paraphrased version to some questions.

3.5 Final Review

Finally, we ask the most experienced annotator to conduct the final question and SQL review. This annotator makes the final decision if multiple reviewers are not sure about some annotation issues. Also, we run a script to execute and parse all SQL labels to make sure they are correct.

4 Dataset Statistics and Comparison

We summarize the statistics of Spider and other text-to-SQL datasets in Table 1. Compared with other datasets, Spider contains databases with multiple tables and contains SQL queries including many complex SQL components. For example, Spider contains about twice more nested queries and 10 times more `ORDER BY (LIMIT)` and `GROUP BY (HAVING)` components than the total of previous text-to-SQL datasets. Spider has 200 distinct databases covering 138 different domains such as college, club, TV show, government, etc. Most domains have one database, thus containing 20-50 questions, and a few domains such as flight information have multiple databases with more than 100 questions in total. On average, each database in Spider has 27.6 columns and 8.8 foreign keys. The average question length and SQL length are about 13 and 21 respectively. Our task uses different databases for training and testing, evaluating the cross-domain performance. Therefore, Spider is the *only one* text-to-SQL dataset that contains both databases with multiple tables in different domains and complex SQL queries It tests the ability of a system to generalize to not only new SQL queries and database schemas but also new domains.

5 Task Definition

On top of the proposed dataset, we define a text-to-SQL task that is more realistic than prior work. Unlike most of the previous semantic parsing or text-to-SQL tasks, models will be tested on *both different complex SQL queries and different complex databases in different domains* in our task. It aims to ensure that models can only make the correct prediction when they truly understand the semantic meaning of the questions, rather than just memorization. Also, because our databases contain different domains, our corpus tests model's ability to generalize to new databases. In this way, model performance on this task can reflect the real semantic parsing ability.

In order to make the task feasible and to focus on the more fundamental part of semantic parsing, we make the following assumptions:

- In our current task, we do not evaluate model performance on generating values. Predicting correct SQL structures and columns is more realistic and critical at this stage based on the low performances of various current state-of-the-art models on our task. In a real world situation, people need to double check what condition values are and finalize them after multiple times. It is unrealistic to predict condition values without interacting with users. In reality, most people know what values to ask but do not know the SQL logic. A more reasonable way is to ask users to use an interface searching the

[3] https://github.com/coleifer/sqlite-web

Dataset	# Q	# SQL	# DB	# Domain	# Table /DB	ORDER BY	GROUP BY	NESTED	HAVING
ATIS	5,280	947	1	1	32	0	5	315	0
GeoQuery	877	247	1	1	6	20	46	167	9
Scholar	817	193	1	1	7	75	100	7	20
Academic	196	185	1	1	15	23	40	7	18
IMDB	131	89	1	1	16	10	6	1	0
Yelp	128	110	1	1	7	18	21	0	4
Advising	3,898	208	1	1	10	15	9	22	0
Restaurants	378	378	1	1	3	0	0	4	0
WikiSQL	80,654	77,840	26,521	-	1	0	0	0	0
Spider	10,181	5,693	200	138	5.1	1335	1491	844	388

Table 1: Comparisons of text-to-SQL datasets. **Spider** is the *only one* text-to-SQL dataset that contains both databases with multiple tables in different domains and complex SQL queries. It was designed to test the ability of a system to generalize to not only new SQL queries and database schemas but also new domains.

values, then ask more specific questions. Also, other previous work with value prediction uses one single database in both train and test which makes it vulnerable to overfitting. However, SQL queries must include values in order to execute them. For value prediction in our task, a list of gold values for each question is given. Models need to fill them into the right slots in their predicted SQL.

- As mentioned in the previous sections, we exclude some queries that require outside knowledge such as common sense inference and math calculation. For example, imagine a table with birth and death year columns. To answer the questions like "How long is X's life length?", we use `SELECT death_year - birth_year`. Even though this example is easy for humans, it requires some common knowledge of the life length definition and the use of a math operation, which is not the focus of our dataset.

- We assume all table and column names in the database are clear and self-contained. For example, some databases use database specific short-cut names for table and column names such as "stu_id", which we manually converted to "student id" in our corpus.

6 Evaluation Metrics

Our evaluation metrics include Component Matching, Exact Matching, and Execution Accuracy. In addition, we measure the system's accuracy as a function of the difficulty of a query. Since our task definition does not predict value string, our evaluation metrics do not take value strings into account.

We will release the official evaluation script along with our corpus so that the research community can share the same evaluation platform.

Component Matching To conduct a detailed analysis of model performance, we measure the average exact match between the prediction and ground truth on different SQL components. For each of the following components:

- `SELECT` • `WHERE` • `GROUP BY`
- `ORDER BY` • `KEYWORDS` (including all SQL keywords without column names and operators)

we decompose each component in the prediction and the ground truth as bags of several subcomponents, and check whether or not these two sets of components match exactly. To evaluate each `SELECT` component, for example, consider `SELECT avg(col1), max(col2), min(col1)`, we first parse and decompose into a set `(avg, min, col1), (max, col2)`, and see if the gold and predicted sets are the same. Previous work directly compared decoded SQL with gold SQL. However, some SQL components do not have order constraints. In our evaluation, we treat each component as a set so that for example, `SELECT avg(col1), min(col1), max(col2)` and `SELECT avg(col1), max(col2), min(col1)` would be treated as the same query. To report a model's overall performance on each component, we compute F1 score on exact set matching.

Exact Matching We measure whether the predicted query as a whole is equivalent to the gold query. We first evaluate the SQL clauses as described in the last section. The predicted query is correct only if all of the components are correct. Because we conduct a set comparison in each clause, this exact matching metric can handle the "ordering issue" (Xu et al., 2017).

Execution Accuracy[4] Since Exact Matching is possible to provide false negative evaluation when the semantic parser is able to generate novel syntax structures, we also consider Execution Accuracy. For Execution Accuracy, the value is a must in order to execute SQL queries. Instead of generating these values, a list of gold values for each question is given. Models need to select them and fill them into the right slots in their predicted SQL. We exclude value prediction in Component and Exact Matching evaluations and do not provide Execution Accuracy in the current version. However, it is also important to note that Execution Accuracy can create false positive evaluation when a predicted SQL returns the same result (for example, 'NULL') as the gold SQL while they are semantically different. So we can use both to complement each other.

Finally, our evaluation also considers multiple acceptable keys if `JOIN` and `GROUP` are in the query. For example, suppose "stu_id" in one table refers to "stu_id" in another table, `GROUP BY` either is acceptable.

SQL Hardness Criteria To better understand the model performance on different queries, we divide SQL queries into 4 levels: easy, medium, hard, extra hard. We define the difficulty based on the number of SQL components, selections, and conditions, so that queries that contain more SQL keywords (`GROUP BY`, `ORDER BY`, `INTERSECT`, nested subqueries, column selections and aggregators, etc) are considered to be harder. For example, a query is considered as hard if it includes more than two `SELECT` columns, more than two `WHERE` conditions, and `GROUP BY` two columns, or contains `EXCEPT` or nested queries. A SQL with more additions on top of that is considered as extra hard. Figure 3 shows examples of SQL queries in 4 hardness levels.

Easy

What is the number of cars with more than 4 cylinders?

```
SELECT COUNT(*)
FROM cars_data
WHERE cylinders > 4
```

Meidum

For each stadium, how many concerts are there?

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

Hard

Which countries in Europe have at least 3 car manufacturers?

```
SELECT T1.country_name
FROM countries AS T1 JOIN continents
AS T2 ON T1.continent = T2.cont_id
JOIN car_makers AS T3 ON
T1.country_id = T3.country
WHERE T2.continent = 'Europe'
GROUP BY T1.country_name
HAVING COUNT(*) >= 3
```

Extra Hard

What is the average life expectancy in the countries where English is not the official language?

```
SELECT AVG(life_expectancy)
FROM country
WHERE name NOT IN
    (SELECT T1.name
     FROM country AS T1 JOIN
     country_language AS T2
     ON T1.code = T2.country_code
     WHERE T2.language = "English"
       AND T2.is_official = "T")
```

Figure 3: SQL query examples in 4 hardness levels.

7 Methods

In order to analyze the difficulty and demonstrate the purpose of our corpus, we experiment with several state-of-the-art semantic parsing models. As our dataset is fundamentally different from the prior datasets such as Geoquery and WikiSQL, we adapted these models to our task as follows. We created a 'big' column list by concatenating columns in all tables of the database together as an input to all models. Also, for each model, we limit the column selection space for each question example to all column of the database which the question is asking instead of all column names in the whole corpus.

Seq2Seq Inspired by neural machine translation (Sutskever et al., 2014), we first apply a basic sequence-to-sequence model, **Seq2Seq**. Then, we also explore **Seq2Seq+Attention** from (Dong and Lapata, 2016) by adding an attention mechanism (Bahdanau et al., 2015). In addition, we include **Seq2Seq+Copying** by adding an attention-based

[4]Please check our website for the latest updates on the task at https://yale-lily.github.io/spider

	Test					Dev
	Easy	Medium	Hard	Extra Hard	All	All
Example Split						
Seq2Seq	23.6	7.9	5.5	1.3	9.8	10.7
Seq2Seq+Attention (Dong and Lapata, 2016)	33.3	15.8	10.3	2.3	16.2	16.5
Seq2Seq+Copying	30.3	13.2	8.8	3.0	14.4	15.4
SQLNet (Xu et al., 2017)	45.9	20.5	12.4	3.3	21.5	22.0
TypeSQL (Yu et al., 2018)	57.2	39.4	24.1	14.4	35.6	37.5
Database Split						
Seq2Seq	17.3	1.9	1.3	0.5	4.9	3.8
Seq2Seq+Attention (Dong and Lapata, 2016)	18.1	2.6	2.0	1.1	5.6	2.7
Seq2Seq+Copying	15.8	3.4	2.0	1.1	5.4	4.1
SQLNet (Xu et al., 2017)	35.6	13.3	6.6	1.3	14.7	14.3
TypeSQL (Yu et al., 2018)	29.9	7.6	4.0	0.8	10.5	10.6

Table 2: Accuracy of Exact Matching on SQL queries with different hardness levels.

Method	SELECT	WHERE	GROUP BY	ORDER BY	KEYWORDS
Example Split					
Seq2Seq	23.3	4.9	15.3	9.2	17.9
Seq2Seq+Attention	31.1	9.1	28.2	20.8	21.4
Seq2Seq+Copying	28.2	8.3	25.5	21.3	19.0
SQLNet	59.8	32.9	35.9	65.5	76.1
TypeSQL	77.3	52.4	47.0	67.5	78.4
Database Split					
Seq2Seq	13.0	1.5	3.3	5.3	8.7
Seq2Seq+Attention	13.6	3.1	3.6	9.9	9.9
Seq2Seq+Copying	12.0	3.1	5.3	5.8	7.3
SQLNet	44.5	19.8	29.5	48.8	64.0
TypeSQL	36.4	16.0	17.2	47.7	66.2

Table 3: F1 scores of Component Matching on all SQL queries on Test set.

copying operation similar to (Jia and Liang, 2016).

The original model does not take the schema into account because it has the same schema in both train and test. We modify the model so that it considers the table schema information by passing a vocabulary mask that limits the model to decode the words from SQL keywords, table and column names in the current database.

SQLNet introduced by (Xu et al., 2017) uses column attention and employs a sketch-based method and generates SQL as a slot-filling task. This fundamentally avoids the sequence-to-sequence structure when ordering does not matter in SQL query conditions. Because it is originally designed for WikiSQL, we extend its `SELECT` and `WHERE` modules to `ORDER BY` and `GROUP BY` components.

TypeSQL proposed by (Yu et al., 2018) improves upon SQLNet by proposing a different training procedure and utilizing types extracted from either knowledge graph or table content to help model better understand entities and numbers in the question. In our experiment, we use the question type info extracted from database content. Also, we extend their modules to `ORDER BY` and `GROUP BY` components as well. It is the only model that uses database content.

8 Experimental Results and Discussion

We summarize the performance of all models on our test set including accuracy of exact matching in Table 2 and F1 scores of component matching in Table 3.

Data Splits For the final training dataset, we also select and include 752 queries and 1659 questions that follow our annotation protocol from six existing datasets: Restaurants, GeoQuery, Scholar, Academic, IMDB, and Yelp. We report results on two different settings for all models: (1) Example split where examples are randomly split into 8659 train, 1034 dev, 2147 test. Questions for the same database can appear in both train and test. (2) Database split where 206 databases are split into 146 train, 20 dev, and 40 test. All questions

for the same database are in the same split.

Overall Performance The performances of the Seq2Seq-based basic models including Seq2Seq, Seq2Seq+Attention, Seq2Seq+Copying, and Iyer et al. (2017) are very low. However, they are able to generate nested and complex queries because of their general decoding process. Thus, they can get a few hard and extra hard examples correct. But in the vast majority of cases, they predict invalid SQL queries with grammatical errors. The attention and copying mechanisms do not help much either.

In contrast, SQLNet and TypeSQL that utilize SQL structure information to guide the SQL generation process significantly outperform other Seq2Seq models. While they can produce valid queries, however, they are unable to generate nested queries or queries with keywords such as `EXCEPT` and `INTERSECT` because they limit possible SQL outputs in some fixed pre-defined SQL structures.

As Component Matching results in Table 3 shows, all models struggle with `WHERE` clause prediction the most. `WHERE` clause is more likely to have multiple columns and operations, which makes its prediction the most challenging. The most number of prediction errors for each component is from column prediction.

In general, the overall performances of all models are low, indicating that our task is challenging and there is still a large room for improvement.

Example Split vs Database Split As discussed in Section 5, another challenge of the dataset is to generalize to new databases. To study this, in Table 2 and Table 3 we compare model performances under the two settings. For all models, the performance under database split is much lower than that under example split. Especially, TypeSQL utilizes column names as question types, and it outperforms other models with a large margin under the example split. However, its performance drops the most on the database split data set. This indicates that the model does well on complex SQL prediction but fails to generalize to new databases. In addition, we observe that all models perform much poorer on column selection under database split than example split.

Overall, the result shows that our dataset presents a challenge for the model to generalize to new databases.

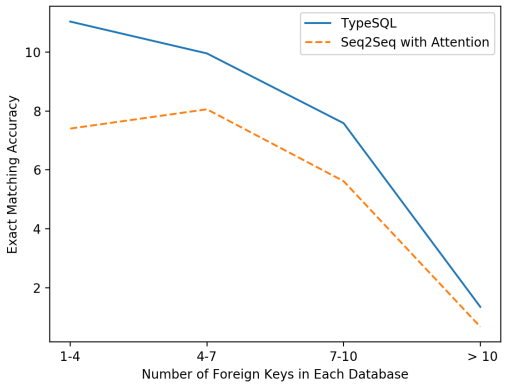

Figure 4: Exact matching accuracy as a function of the number of foreign keys.

Complexity of Database Schema In order to show how the complexity of the database schema affects model performance, Figure 4 plots the exact matching accuracy as a function of the number of foreign keys in a database. The performance decreases as the database has more foreign keys. The first reason is that the model has to choose column and table names from many candidates in a complex database schema. Second, a complex database schema presents a great challenge for the model to capture the relationship between different tables with foreign keys. SQL answers to questions on the database with more number of foreign keys are more likely to join more tables. It indicates that this task requires more effective methods to encode the relation of tables with foreign keys.

9 Conclusion

In this paper we introduce Spider, a large, complex and cross-domain semantic parsing and text-to-SQL dataset, which directly benefits both NLP and DB communities. Based on Spider, we define a new challenging and realistic semantic parsing task. Experimental results on several state-of-the-art models on this task suggest plenty space for improvement.

Acknowledgement

We thank Graham Neubig, Tianze Shi, Catherine Finegan-Dollak, and three anonymous reviewers for their discussion and feedback. We also thank Barry Williams for providing part of our database schemas from the DatabaseAnswers.

References

Miltiadis Allamanis, Daniel Tarlow, Andrew D. Gordon, and Yi Wei. 2015. Bimodal modelling of source code and natural language. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 2123–2132. JMLR.org.

Yoav Artzi and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association forComputational Linguistics*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR*.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*.

Jonathan Berant and Percy Liang. 2014. Semantic parsing via paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1415–1425, Baltimore, Maryland. Association for Computational Linguistics.

Deborah A. Dahl, Madeleine Bates, Michael Brown, William Fisher, Kate Hunicke-Smith, David Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. 1994. Expanding the scope of the atis task: The atis-3 corpus. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, Stroudsburg, PA, USA. Association for Computational Linguistics.

Dipanjan Das, Nathan Schneider, Desai Chen, and Noah A. Smith. 2010. Probabilistic frame-semantic parsing. In *NAACL*.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*.

Li Dong and Mirella Lapata. 2018. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 731–742. Association for Computational Linguistics.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan Dhanalakshmi Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-sql evaluation methodology. In *ACL 2018*. Association for Computational Linguistics.

Alessandra Giordani and Alessandro Moschitti. 2012. Translating questions to sql queries with generative parsers discriminatively reranked. In *COLING (Posters)*, pages 401–410.

Po-Sen Huang, Chenglong Wang, Rishabh Singh, Wen tau Yih, and Xiaodong He. 2018. Natural language to structured query generation via meta-learning. In *NAACL*.

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

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Fei Li and HV Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *VLDB*.

Yunyao Li, Huahai Yang, and HV Jagadish. 2006. Constructing a generic natural language interface for an xml database. In *EDBT*, volume 3896, pages 737–754. Springer.

P. Liang, M. I. Jordan, and D. Klein. 2011. Learning dependency-based compositional semantics. In *Association for Computational Linguistics (ACL)*, pages 590–599.

Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D. Ernst. 2018. Nl2bash: A corpus and semantic parser for natural language interface to the linux operating system. In *LREC*.

Wang Ling, Phil Blunsom, Edward Grefenstette, Karl Moritz Hermann, Tomás Kociský, Fumin Wang, and Andrew Senior. 2016. Latent predictor networks for code generation. In *ACL (1)*. The Association for Computer Linguistics.

Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. 2018. The natural language decathlon: Multitask learning as question answering. *arXiv preprint arXiv:1806.08730*.

Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2015. Learning to generate pseudo-code from source code using statistical machine translation (t). In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ASE '15.

Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. 2004. Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In

Proceedings of the 20th international conference on Computational Linguistics, page 141. Association for Computational Linguistics.

Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. 2003a. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th International Conference on Intelligent User Interfaces*.

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

P. J. Price. 1990. Evaluation of spoken language systems: the atis domain. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27,1990*, pages 91–95.

Chris Quirk, Raymond Mooney, and Michel Galley. 2015. Language to code: Learning semantic parsers for if-this-then-that recipes. 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)*, volume 1, pages 878–888.

Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract syntax networks for code generation and semantic parsing. In *ACL (1)*, pages 1139–1149. Association for Computational Linguistics.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250.

Siva Reddy, Mirella Lapata, and Mark Steedman. 2014. Large-scale semantic parsing without question-answer pairs. *Transactions of the Association for Computational Linguistics*, 2:377–392.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Lappoon R. Tang and Raymond J. Mooney. 2001a. Using multiple clause constructors in inductive logic programming for semantic parsing. In *Proceedings of the 12th European Conference on Machine Learning*, pages 466–477, Freiburg, Germany.

Lappoon R Tang and Raymond J Mooney. 2001b. Using multiple clause constructors in inductive logic programming for semantic parsing. In *ECML*, volume 1, pages 466–477. Springer.

Chenglong Wang, Alvin Cheung, and Rastislav Bodik. 2017. Synthesizing highly expressive sql queries from input-output examples. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 452–466. ACM.

Chenglong Wang, Po-Sen Huang, Alex Polozov, Marc Brockschmidt, and Rishabh Singh. 2018. Execution-guided neural program decoding. In *ICML workshop on Neural Abstract Machines and Program Induction v2 (NAMPI)*.

David HD Warren and Fernando CN Pereira. 1982. An efficient easily adaptable system for interpreting natural language queries. *Computational Linguistics*, 8(3-4):110–122.

Yuk Wah Wong and Raymond J. Mooney. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL-2007)*, Prague, Czech Republic.

Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*.

Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017a. Sqlizer: query synthesis from natural language. *Proceedings of the ACM on Programming Languages*.

Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017b. Sqlizer: Query synthesis from natural language. *Proc. ACM Program. Lang.*, 1(OOPSLA):63:1–63:26.

Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. 2018. Learning to mine aligned code and natural language pairs from stack overflow. In *International Conference on Mining Software Repositories (MSR)*.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *ACL (1)*, pages 440–450. Association for Computational Linguistics.

Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018. Typesql: Knowledge-based type-aware neural text-to-sql generation. In *Proceedings of NAACL*. Association for Computational Linguistics.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *AAAI/IAAI*, pages 1050–1055, Portland, OR. AAAI Press/MIT Press.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *UAI*.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.