KaggleDBQA: Realistic Evaluation of Text-to-SQL Parsers

Chia-Hsuan Lee[◇] **Oleksandr Polozov**[♠] **Matthew Richardson**[♠]
[◇]University of Washington [♠]Microsoft Research, Redmond
chiahlee@uw.edu
{polozov,mattri}@microsoft.com

Abstract

The goal of database question answering is to enable natural language querying of real-life relational databases in diverse application domains. Recently, large-scale datasets such as Spider and WikiSQL facilitated novel modeling techniques for text-to-SQL parsing, improving zero-shot generalization to unseen databases. In this work, we examine the challenges that still prevent these techniques from practical deployment. First, we present KaggleDBQA, a new cross-domain evaluation dataset of real Web databases, with domain-specific data types, original formatting, and unrestricted questions. Second, we re-examine the choice of evaluation tasks for text-to-SQL parsers as applied in real-life settings. Finally, we augment our in-domain evaluation task with *database documentation*, a naturally occurring source of implicit domain knowledge. We show that KaggleDBQA presents a challenge to state-of-the-art zero-shot parsers but a more realistic evaluation setting and creative use of associated database documentation boosts their accuracy by over 13.2%, doubling their performance.

1 Introduction

Text-to-SQL parsing is a form of *database question answering* (DBQA) that answers a user's natural-language (NL) question by converting it into a SQL query over a given relational database. It can facilitate NL-based interfaces for arbitrary end-user applications, thereby removing the need for domain-specific UX or learning query languages. As such, DBQA attracted significant attention in academia and industry, with development of supervised datasets (Yu et al., 2018), large-scale models (Wang et al., 2020b; Zeng et al., 2020), and novel modeling techniques (Yu et al., 2020; Deng et al., 2020).

The key challenge of text-to-SQL parsing is *zero-shot generalization* to unseen domains, *i.e.* to new database schemas and differently distributed NL questions. Large-scale annotated datasets like Spider (Yu et al., 2018) and WikiSQL (Zhong et al., 2017) evaluate *cross-domain generalization* of text-to-SQL parsers by restricting overlap between train and test domains. Such challenging benchmarks facilitate rapid progress in DBQA. State-of-the-art (SOTA) accuracy on Spider rose from 12.4% to 70.5% in just two years since its release, demonstrating the value of well-chosen evaluation settings.

Despite impressive progress in DBQA, deployment of SOTA parsers is still challenging. They often lack robustness necessary to deploy on real-life application domains. While many challenges underlie the gap between SOTA DBQA and its real-life deployment, we identify three specific discrepancies.

First, Spider and WikiSQL datasets normalize and preprocess database schemas or rely on academic example databases that originate with human-readable schemas (Suhr et al., 2020). In contrast, industrial databases feature abbreviated and obscure naming of table, columns, and data values, often accrued from legacy development or migrations. Figure 1 shows a characteristic example. After deployment, text-to-SQL parsers struggle with *schema linking* to domain-specific entities because they do not match the distribution seen in their pre-training (*e.g.* BERT) or supervised training (*e.g.* Spider).

Second, the NL questions of Spider and WikiSQL have high *column mention percentage* (Deng et al., 2020), which makes their language unrealistic. This can be an artifact of rule-generated NL templates (as in WikiSQL) or annotation UIs that prime the annotators toward the schema (as in Spider). Either way, real-world deployment of a text-to-SQL parser optimized on Spider faces a distribution shift in NL, which reduces its realistic performance.

Finally, the standard evaluation setting of cross-domain text-to-SQL parsing assumes no in-domain

Database: `Student Math Score`

Table `FINREV_FED_17`:

🔑 `state_code`	`school_district`	`yr_data`	`t_fed_rev`	`c14`	`c15`	⋮
33	NEW YORK CITY SCHOOL DISTRICT	17	2061297	956851	439209	⋮
47	FAIRFAX CO SCHS	17	126916	21035	36886	⋮

Column Descriptions:

`t_fed_rev`	Total federal revenue through the state to each school district
`c14`	Federal revenue through the state-Title 1 (no child left behind act)
`c15`	Federal revenue through the state - Child Nutrition A

Table `FINREV_FED_17_KEY`:

🔑 `state_code`	`state`	`#_Records`
1	Alabama	137
...	...	...
50	Wisconsin	425
51	Wyoming	48

Example Question: *Which school district received the most of federal revenue through state in Wisconsin?*

Example SQL:

```
SELECT T1.school_district
FROM FINREV_FED_17 as T1 JOIN FINREV_FED_KEY_17 as T2
ON T1.state_code = T2.state_code WHERE T2.state = "Wisconsin"
ORDER BY T1.t_fed_rev DESC LIMIT 1
```

Figure 1: Two table excerpts from the Student Math Score database in KaggleDBQA and an example question-SQL pair. The column names are abbreviated (*e.g.* `t_fed_rev`) or obscure (*e.g.* `c14`, `c25`) but documentation (*e.g.* column descriptions) alleviates this. Source: `https://kaggle.com/loganhenslee/studentmathscores`.

supervision. This simplifies parser evaluation and raises the challenge level for zero-shot generalization. However, it does not leverage knowledge sources commonly present in real-world applications, both explicit (annotated in-domain examples) and implicit (*e.g.* database documentation, SQL queries in the application codebase, or data distributions). A well-chosen alternative evaluation setting would facilitate development of DBQA technologies that match their real-world evaluation.

KaggleDBQA We introduce KaggleDBQA, a new dataset and evaluation setting for text-to-SQL parsers to bridge the gap between SOTA DBQA research and its real-life deployment.[1] It systematically addresses three aforementioned challenges:

- To test database generalization, it includes real-world databases from Kaggle,[2] a platform for data science competitions and dataset distribution. They feature abbreviated and obscure column names, domain-specific categorical values, and minimal preprocessing (Section 3.1).

- To test question generalization, we collected unrestricted NL questions over the databases in KaggleDBQA. Importantly, the annotators were not presented with original column names, and given no task priming (Section 3.2). Out of 400 collected questions, one-third were out of scope for SOTA text-to-SQL parsers. The remaining 272 questions, while expressible, can only be solved to 13.56% accuracy (Section 4).

- Finally, we augment KaggleDBQA with *database documentation*, common metadata for real-world databases and a rich source of implicit domain knowledge. Database documentation includes column and table descriptions, categorical value descriptions (known as *data dictionaries*), SQL examples, and more (Section 3.3). We present a technique to augment SOTA parsers with column and value descriptions, which significantly improves their out-of-domain accuracy (Section 4).

Figure 1 shows a representative example from the dataset. Aligning *"federal revenue"* and `t_fed_rev` is hard without domain knowledge.

In addition to more realistic data and questions, we argue that evaluation of real-world text-to-SQL performance should assume *few-shot* access to ∼10 in-domain question-SQL examples rather than measuring *zero-shot* performance. In practical terms, few-shot evaluation assumes up to 1-2 hours of effort by a target database administrator or application developer, and translates to significant performance benefits. In a few-shot evaluation setting, augmenting a SOTA text-to-SQL parser (RAT-SQL by Wang et al. (2020b)) with database documentation almost doubled its performance from 13.56% to 26.77%. See Section 4.

[1]Available at `https://aka.ms/KaggleDBQA`.
[2]`https://www.kaggle.com`

2 Related Work

Text-to-SQL Semantic Parsing Semantic parsing has been studied extensively for decades (Liang, 2016). Key *in-domain* datasets such as GeoQuery (Zelle and Mooney, 1996) and ATIS (Dahl et al., 1994) acted as initial catalyst for the field by providing an evaluation measure and a training set for learned models. Applying a system to a domain with a different distribution of questions or parses required out-of-domain data or domain transfer techniques. Recently, *cross-domain* datasets WikiSQL (Zhong et al., 2017) and Spider (Yu et al., 2018) proposed a *zero-shot* evaluation methodology that required out-of-domain generalization to unseen database domains. This inspired rapid development of *domain-conditioned* parsers that work "out of the box" such as RAT-SQL (Wang et al., 2020b) and IRNet (Guo et al., 2019). We use the same exact match accuracy metric as these works. Recent work (Zhong et al., 2020) has proposed evaluating SQL prediction via semantic accuracy by computing denotation accuracy on automatically generated databases instead.

Few-shot learning In this paper, we propose a *few-shot* evaluation to inspire future research of practical text-to-SQL parsers. Like zero-shot, few-shot has access to many out-of-domain examples, but it also has access to a small number of in-domain examples as well. Few-shot learning has been applied to text classification in (Mukherjee and Awadallah, 2020), and has also been applied to semantic parsing. Common techniques include meta-learning (Huang et al., 2018; Wang et al., 2020a; Li et al., 2021; Sun et al., 2020) and adversarial learning (Li et al., 2020).

Generalization and Practical usability Recent work has begun to question whether existing datasets are constructed in a way that will lead to models that generalize well to new domains. Suhr et al. (2020) identified a number of challenges with text-to-SQL datasets, one of which is an artificially high overlap between words in a question and words in the tables. This issue appears in Spider and is a byproduct of the fact that question authors view the database schema as they write their question. The Spider-Realistic (Deng et al., 2020) dataset aims to reduce this by explicitly rewriting the questions to avoid overlapping terms. Other works has studied the problem of the gap between academic datasets and their practical usability (de Vries et al., 2020; Radhakrishnan et al., 2020; Zhang et al., 2020), including highlighting the need for data to be real. Our goal was to create an evaluation dataset and metric that minimizes this gap; our dataset is constructed from real data found on Kaggle that has been used for competitions or other analyses.

Another direction of generalization being explored is compositionality. Keysers et al. (2020) used rules to generate a large-scale semantic parsing dataset that specifically tests models for composability.

Leveraging other resources for learning Rastogi et al. (2020) provide NL descriptions for slots and intents to help dialogue state tracking. Logeswaran et al. (2019) use descriptions to facilitate zero-shot learning for entity linking. Weller et al. (2020) use descriptions to develop a system that can perform zero-shot learning on new tasks. We follow by including documentation on each included real-world database. Notably, this documentation was *written for human consumption of the database* rather than prepared for KaggleDBQA, and thus is a natural source of domain knowledge. It provides similar benefits to codebase documentation and comments, which improve source code encoding for AI-assisted software engineering tasks (Panthaplackel et al., 2020; Wei et al., 2019).

3 KaggleDBQA: A Real World Dataset

The goal of the KaggleDBQA evaluation dataset is to more closely reflect the data and questions a text-to-SQL parser might encounter in a real-world setting. As such, it expands upon contemporary cross-domain text-to-SQL datasets in three key aspects: **(i)** its **databases** are pulled from real-world data sources and *not* normalized; **(ii)** its **questions** are authored in environments that mimic natural question answering; **(iii)** its **evaluation** assumes the type of system augmentation and tuning that could be expected from domain experts that execute text-to-SQL parser deployment. We describe each of these components in turn in this section.

3.1 Database Collection

We chose to obtain databases from Kaggle, a popular platform for hosting data science competitions and sharing datasets and code. Their hosted datasets are by definition "real" as they are used by members of the site for research. Competition hosts upload their data unnormalized, and the

Table 1: Comparison of text-to-SQL datasets. We follow the data filtering rules of Suhr et al. (2020) and Deng et al. (2020), which reduces the effective number of examples from the original datasets to make them consistent. %WHERE measures the percentage of examples where all **WHERE**/**HAVING** columns in the SQL query are explicitly mentioned in the NL question. %VAL compares all the values in the SQL queries; %SELECT compares all the **SELECT** columns; %NON SELECT compares all columns except the **SELECT** columns. KaggleDBQA has low column mention percentage and contains databases with multiple tables.

Dataset	# Examples	# DB	# Table/DB	% WHERE	% VAL	% SELECT	% NON-SELECT
ATIS	275	1	25	0.0	95.6	0.0	0.0
GeoQuery	525	1	7	3.8	100.0	32.9	9.1
Restaurants	39	1	3	0.0	100.0	0.0	0.0
Academic	179	1	17	5.2	100.0	15.1	1.7
IMDB	111	1	17	1.6	100.0	7.1	0.8
Yelp	68	1	8	4.2	100.0	5.7	4.1
Scholar	396	1	10	0.0	100.0	0.7	0.2
Advising	281	1	15	4.0	100.0	6.1	3.9
Spider Train	7000	140	5.26	40.8	89.01	52.4	41.6
Spider Dev	1034	20	4.05	39.2	91	48.2	33.1
KaggleDBQA	272	8	2.25	8.7	73.5	24.6	6.8

data content and formatting matches its domain-specific usage (see Figure 1 for an example). To construct KaggleDBQA, we randomly selected 8 Kaggle datasets that satisfied the following criteria: (a) contained a SQLite database; (b) licensed under a republishing-permissive license; (c) had associated documentation that described the meaning of the tables and columns.

3.2 Questions

For each database, we asked five annotators to write ten domain-specific questions that they think someone might be interested in and that can be answered using the database. We use five annotators per database to help guarantee diversity of questions. Each annotated two databases, for a total of 20 annotators and 400 questions.

The annotators are not required to possess SQL knowledge so their questions are more reflective of natural user interests. Importantly, to discourage users from using the same terms from the database schema in their questions, *we replace the original column names with the column descriptions*. When annotating the questions, the annotators are shown a paragraph description of the database, table names, column descriptions and ten sampled rows for each table. We do not provide any constraints or templates other than asking them to avoid using exact phrases from the column headings in the questions. Appendix A.2.3 shows the full guidelines.

Separately, each question is annotated with its SQL equivalent by independent SQL experts. They are given full access to all of the data content and database schema. One-third of the questions were yes/no, percentage, temporal, or unexpressible in SQL and were not considered in our evaluation of SOTA models (see Appendix A.2.2 for details), leaving 272 questions in total.

3.3 Database Documentation

Each database has associated plain-text *documentation* that can assist text-to-SQL parsing. It is commonly found as internal documentation for database administrators or external documentation accompanying a dataset release. The contents vary but often contain an overview of the database domain, descriptions of tables and columns, sample queries, original sources, and more.

While all of these types of information could be leveraged to assist with domain transfer, in this work we focus on the *column descriptions*. They help address the *schema linking* problem of text-to-SQL parsing, *i.e.* aligning entity references in the question with database columns (Wang et al., 2020b). For example, *"federal revenue"* in Figure 1 must be aligned to the column `t_fed_rev` even though its abbreviated name makes alignment non-obvious.

We manually extract the column descriptions from the database documentation and provide the mapping from column to description as part of KaggleDBQA. The descriptions are free text and sometimes contain additional information such as defining the values in an categorical column. Such information could help with the *value-linking* problem (mapping a value in the question to the column

Table 2: Average partial match % of columns descriptions across examples. We check whether 1- to 3-grams in the question are part of any column descriptions.

Type of n-gram	1	2	3
% Cols matched in golden SQL	56.27	21.47	4.80
# Cols matched in golden SQL	1.06	0.37	0.07
# Cols matched not in the SQL	4.69	1.29	0.13

that likely contains it). We leave the entire description as a single field and leave it to future work to explore these uses further. In addition to column descriptions, we also include the original unstructured documentation which can be used for future research on automatically extracting descriptions or leveraging other domain knowledge.

3.4 Few-shot Evaluation Setting

The current cross-domain datasets Spider (Yu et al., 2018) and WikiSQL (Zhong et al., 2017) evaluate models in a *zero-shot* setting, meaning the model is trained on one set of domains and evaluated on a completely disjoint set. This evaluation encourages the development of systems that work well "out of the box" and has spurred great development in cross-domain text-to-SQL systems that are able to generalize to new domains. However, we believe the zero-shot setting is overly-restrictive compared to how text-to-SQL systems are likely to be actually used in practice.

We postulate that it is more realistic to assume a setting where an application author spends 1-2 hours authoring examples and adapting existing database documentation. This time investment is a small fraction of the time required to prepare an application itself and so we believe application authors would devote the time if it resulted in increased text-to-SQL accuracy. In informal experiments, we have found SQL annotators can author 10-20 examples in an hour. Thus, the KaggleDBQA evaluation setting is *few-shot*: 30% of the questions for each domain (6-15 depending on the domain) are designated as *in-domain* and may be used as part of training for that domain, along with documentation. The remaining 70% are used for evaluation.

We report accuracy in both the *few-shot* as well as the standard *zero-shot* (cross-domain) setting in this paper, but consider the few-shot setting to be the primary evaluation setting for KaggleDBQA. Evaluation is conducted on the same 70% portion regardless of setting, to ensure comparable results.

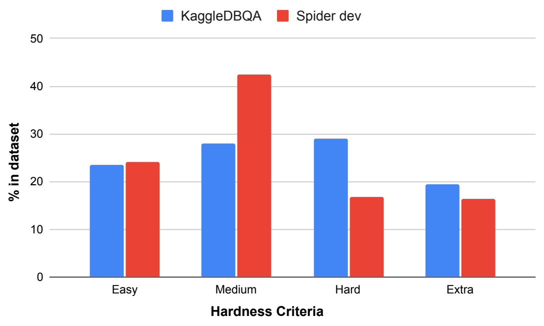

Figure 2: Comparisons of text-to-SQL datasets in terms of SQL structure hardness. KaggleDBQA has more complex SQL query structure than the Spider dev set.

3.5 Dataset Statistics and Comparison

We compare KaggleDBQA with previous benchmark datasets using key metrics in Table 1. KaggleDBQA has the lowest value mention percentage among all datasets, and also exhibits a low overlap between question terms and column names similar to that in all of the datasets besides Spider, making it more in line with what would be expected in a real-world setting where the people asking questions are not familiar with the actual database schema and terminology. This is likely a result of replacing column names with descriptions in the question annotation task.

We also analyze the overlap between question terms and column descriptions in Table 2. Because the descriptions are significantly longer than column names, we require only that they share an n-gram in common (ignoring stop-words) rather than requiring exact match as was done for column mention percent. Unigram overlap is reasonably high (56% of correct columns match the question) but also results in many false-positive matches with other columns. Increasing n-gram size decreases false-positives but also rapidly decreases the correct column match percent. Thus, column descriptions may help guide the model, but are not as strong of a signal as found in Spider which suffers from high exact column name match overlap. This was our intention in asking our annotators to avoid using the descriptions verbatim when writing questions.

To measure the complexity of SQL in KaggleDBQA, we adopt the hardness criteria of Spider and report the numbers in Figure 2. The queries are on average more complex than Spider's, with significantly more hard and extra-hard ones.

4 Experiments

4.1 Baseline Results

We first evaluate KaggleDBQA using models that were developed for the Spider dataset.

EditSQL (Zhang et al., 2019): EditSQL (with BERT) is the highest-performing model on the Spider dataset that also provides an open-source implementation along with a downloadable trained model.[3] The model was built for edit-based multi-turn parsing tasks, but can also be used as a single-turn parser for Spider or KaggleDBQA. It employs a sequence-to-sequence model with a question-table co-attention encoder for schema encoding.

RAT-SQL (Wang et al., 2020b): RAT-SQL (v3 + BERT) is the model with highest accuracy on the Spider leaderboard that also provides an open-source implementation.[4,5] It adds string matching to the encoder through the use of relation-aware self-attention and adopts a tree-based decoder to ensure the correctness of the generated SQL.

Throughout this paper, we use the same *exact-match accuracy* metric introduced by the Spider dataset. Although our primary evaluation setting is few-shot, we first examine the traditional zero-shot setting to present an unbiased comparison with previous results. Table 3 compares the performance of these two models (both trained on Spider). As can be seen, the performance of both models is significantly lower on KaggleDBQA. This echoes the findings of Suhr et al. (2020) who found that a model trained on Spider did not generalize well to other datasets. Also, KaggleDBQA has much fewer column mentions and much more complex SQL than Spider (see Table 1 and Figure 2).

For all further experiments on KaggleDBQA that emulate real-world evaluation, we choose RAT-SQL as the best performing parser.

4.2 RAT-SQL on KaggleDBQA

4.2.1 Moving to the Few-Shot Setting

To apply RAT-SQL to KaggleDBQA's few-shot setting, for each domain we create a model by fine-tuning on its 30% in-domain data. See Appendix A.3 for implementation details. This fine-tuning is always performed as the last step before evaluation.

Table 3: Zero-shot testing results of various open-source models on KaggleDBQA and on the test set of Spider. All numbers are the exact match accuracy evaluated by the Spider official scripts. The Spider results are from the official leaderboard. The KaggleDBQA results are the average of three different runs.

Models	Spider	KaggleDBQA
RAT-SQL (Wang et al., 2020b)	65.60	13.56
EditSQL (Zhang et al., 2019)	53.40	11.73

As Table 4 shows, fine-tuning on a small amount of in-domain data dramatically increases overall accuracy from 13.56% to 17.96% (rows (a) and (e)), Although the few-shot setting is our primary setting, we also present results in the zero-shot setting to compare to previous work (Table 4 rows (e)-(h)). However, in the remainder of the paper we will be focusing on the few-shot setting.

4.2.2 Leveraging Database Documentation

The database schemas in KaggleDBQA are obscure, making the task difficult without leveraging the database documentation. We consider only the column descriptions, but other portions of the documentation may prove useful in future work. The best approach for incorporating column descriptions into a text-to-SQL model is model-specific. RAT-SQL makes use of relations between question tokens and schema terms to assist with *schema-linking*. We extend the same functionality to column descriptions by appending the column descriptions to the column names (separated by a period) and recomputing matching relations. The concatenated column name is also presented to the transformer encoder for schema encoding.

Simply adding these descriptions results in mismatch between the training set (Spider) which does not have descriptions, and the evaluation set (KaggleDBQA) which does. To alleviate it, we first augment the schemas in Spider with artificial descriptions. For column c of table t, the description for c is "*the c of the t*". We then retrain RAT-SQL on Spider with these artificial descriptions.

Since the artificial descriptions simply restate information from the schema, the model may not learn to leverage them for any further information about schema linking and simply treat them as noise. Therefore, we also evaluate RAT-SQL adapted to the general domain of KaggleDBQA so that it (a)

[3] https://github.com/ryanzhumich/editsql

[4] As of one month before paper authoring. Current SOTA systems are also based on RAT-SQL and add less than 5% accuracy, thus will likely behave similarly.

[5] https://github.com/microsoft/rat-sql

Table 4: Exact match accuracy and standard error on KaggleDBQA, mean of three runs with different random seeds.

With *fine-tuning*									
Models	Nuclear	Crime	Pesticide	MathScore	Baseball	Fires	WhatCD	Soccer	Avg
(a) RAT-SQL	28.78	35.18	11.76	3.50	14.81	30.66	10.68	8.33	17.96 ± 0.5%
(b) *w.* desc	22.72	29.62	12.74	3.50	11.11	33.33	19.04	8.33	17.55 ± 0.6%
(c) *w. adaptation*	28.78	44.44	16.66	8.76	16.04	37.33	16.66	13.87	22.82 ± 0.1%
(d) *w.* desc + *adaptation*	36.35	44.44	21.56	7.01	22.22	41.33	27.38	13.87	26.77 ± 0.4%

Without *fine-tuning*									
Models	Nuclear	Crime	Pesticide	MathScore	Baseball	Fires	WhatCD	Soccer	Avg
(e) RAT-SQL	22.72	25.92	8.82	0.00	12.34	17.33	4.76	16.66	13.56 ± 0.1%
(f) *w.* desc	24.24	20.37	7.84	0.00	9.87	13.33	7.14	16.66	12.43 ± 0.1%
(g) *w. adaptation*	25.75	38.88	12.74	3.50	7.40	20.00	9.52	16.66	16.80 ± 0.8%
(h) *w.* desc + *adaptation*	30.29	25.92	17.64	3.50	16.04	25.33	11.9	16.66	18.41 ± 0.4%

experiences useful descriptions and (b) adapts to the language distribution of KaggleDBQA. We evaluate the benefits of this *adaptation* using leave-one-out: for each domain in KaggleDBQA, we fine-tune the model on all other domains except for the target (with the same fine-tuning parameters as for few-shot learning). Adapting in this way is predictive of the performance of a novel domain with similar characteristics.

As with the other few-shot results, the model is then fine-tuned on the few examples of target domain data. *Adaptation* and *fine-tuning* are two separate training processes. *Adaptation* is meant to adapt to the real-world distribution. *Fine-tuning* is meant to adjust for in-domain knowledge. The most effective setting for a target database in our experiments is to conduct *adaptation* first, followed by *fine-tuning*.

Table 4 (row (d)) shows the results. Using column descriptions in the context of adaptation increases model accuracy from 17.96% to 26.77%. Ablations show that adaptation and descriptions each contribute approximately half of this gain (row (c)). Descriptions provide no benefit without adaptation (row (b)), likely due to the train-test mismatch between artificial descriptions and real ones. Without any artificial descriptions, accuracy drops even further so they are critical to leveraging in-domain knowledge. Overall, incorporating in-domain data (*i.e.* a few-shot setting and database documentation) nearly doubles model accuracy from 13.56% to 26.77% on KaggleDBQA.

Table 5: Exact match accuracy and standard error on schema-normalized KaggleDBQA, average of three runs with different random seeds.

With *fine-tuning*	
Models	Avg
(a) RAT-SQL	17.96 ± 0.5%
(b) *w.* desc	17.55 ± 0.6%
(c) *w.* normalization	23.09 ± 0.9%
(e) *w. adaptation*	22.82 ± 0.1%
(f) *w.* desc + *adaptation*	26.77 ± 0.4%
(g) *w.* normalization + *adaptation*	25.60 ± 0.9%
(h) *w.* desc + normalization + *adaptation*	27.83 ± 0.7%

4.3 Column Normalization

One of the major challenges in KaggleDBQA is that column names are often obscure or abbreviated. A natural question is whether this creates difficulty because the model struggles to understand the meaning of a column or because it leads to a low overlap between question and column terms. In an attempt to tease these factors apart, we created a *normalized* version of KaggleDBQA by replacing the obscure column names with normalized column names such as one might find in the Spider dataset. This was done manually using column descriptions to help clarify each column and without introducing any extra knowledge into the column names except for the expansion of abbreviations (*e.g.* `t_fed_rev` → `total federal revenue`).

In Table 5 we give the results of evaluation on the normalized KaggleDBQA, following the same setup as Table 4. Normalization provides a significant boost in performance (row (c) vs. row (a)). The trend is similar to Table 4. Without adaptation, models with descriptions are not better than those without (row (b) vs. row (a), row (d) vs. row (c)). After adaptation, the train-test mismatch is partly mitigated and the performance improves (row (f) vs. row (e), row (h) vs. row (g)). Normalization and descriptions provide complementary knowledge augmentation, jointly improving accuracy by 5% (row (h) vs. row (e)), more than either alone.

Normalization helps clarify the obscure column names of KaggleDBQA. However, the other chal-

Table 6: Examples where description-augmented ("desc.") models solve a question that unaugmented models ("no desc.") do not. Both models are adapted and fine-tuned. Both omit values, as per the official Spider metric.

Database `USWildFires`	**Column Descriptions**
`STAT_CAUSE_CODE`	Code for the (statistical) cause of the fire
`STAT_CAUSE_DESCR`	Description of the (statistical) cause of the fire.
`FIRE_SIZE`	Estimate of acres within the final perimeter of the fire
Question	What's the most common cause of the fire (code) in the database?
no desc.	`SELECT Fires.STAT_CAUSE_DESCR FROM Fires GROUP BY Fires. STAT_CAUSE_DESCR ORDER BY Count(*)DESC LIMIT 1`
desc.	`SELECT Fires.STAT_CAUSE_CODE FROM Fires GROUP BY Fires. STAT_CAUSE_CODE ORDER BY Count(*)DESC LIMIT 1`
Question	What is the total area that has been burned until now?
no desc.	`SELECT Sum(*)FROM Fires`
desc.	`SELECT Sum(Fires.FIRE_SIZE)FROM Fires`
Database `Pesticide`	**Column Descriptions**
`origin`	Code indicating sample origin (1=U.S. 2=imported 3=unknown)
`country`	Country of origin if the sample was imported
Question	How many samples come from other countries?
no desc.	`SELECT sampledata15.country FROM sampledata15`
desc.	`SELECT Count(*)FROM sampledata15 WHERE sampledata15.origin =`'⬚'

Table 7: Distribution of error types in each domain over 10 randomly-selected erroneous examples.

Error Types	Nuclear	Crime	Pesticide	MathScore	Baseball	Fires	WhatCD	Soccer	%
Entity-column matching	0	2	2	3	0	2	0	0	15.00%
Incorrect Final Column	3	2	5	3	4	4	4	2	33.75%
Missing Constraint	5	3	3	1	5	5	2	2	32.50%
Incorrect Constraint	4	2	2	2	6	0	7	2	31.25%
Understanding Error	0	1	0	4	0	1	2	3	13.75%
Ambiguous Columns	0	2	2	0	1	1	0	0	7.50%
Equivalent	1	0	2	0	0	0	0	0	3.75%

lenges such as low column mention percentage and in-domain schema conventions still leave significant room for improvement. We provide the full experimental results on normalized tables in the Appendix.

4.4 Error Analysis

Table 6 shows examples of improvements due to descriptions. First, column descriptions help the parser correctly identify columns to select. For instance, it chooses `STAT_CAUSE_CODE` over `STAT_CAUSE_DESCR` when asked for "the most common cause of the fire (code)". Second, they clarify necessary constraints. For instance, when asked "how many samples come from other countries?", the parser chooses the correct `origin` column rather than superficially-matching `country` in the clause `WHERE sampledata15.origin = "2"`.

Table 7 shows a distribution of error types in KaggleDBQA using 10 randomly-selected erroneous predictions for each domain. The error categories mostly follow Suhr et al. (2020), modulo (a) removing unobserved categories, (b) separating semantically equivalent predictions into their own "Equivalent" category, and (c) categorizing significant structural errors as "Understanding Errors". We also provide more characteristics of each database in Table 8 in an attempt to understand the difference in performance across databases. Our model performs worst on the databases with the most columns (*Pesticide*, *Baseball* and *Soccer*). The only database with lower accuracy is *MathScore* which has multiple tables and a relatively small fine-tuning set.

The most common error types and their examples are summarized in Table 9. **(i)** The most common type is "Incorrect Final Column" (33.75%), illustrating the difficulty of schema linking in KaggleDBQA even with documentation and fine-tuning. **(ii)** 32.5% of the errors are in "Missing Constraints". In KaggleDBQA questions, users sometimes use implications instead of directly mentioning the desired constraint, *e.g.* "in preparation" for `Status = "Under Construction"`. **(iii)** 31.25% of the errors are in "Incorrect Constraint", *e.g.* failing to parse "highest" into the top-1 result in

Table 8: Statistics of each database in KaggleDBQA.

	Nuclear	Crime	Pesticide	MathScore	Baseball	Fires	WhatCD	Soccer
#Tables	1	1	2	3	5	1	2	2
#Columns	15	6	34	15	44	19	10	37
#Fine-tuning Examples	10	9	16	9	12	12	13	6
#Test Examples	22	18	34	19	27	25	28	12

Table 9: The most common error types of our best model and their representative examples.

	33.75%: Incorrect Final Column
Question	What is the latitudinal band that is most likely to experience wildfires in the USA?
Predicted	SELECT STAT_CAUSE_DESCR FROM Fires GROUP BY STAT_CAUSE_DESCR ORDER BY Count(*)Desc LIMIT 1
Gold	SELECT LATITUDE FROM Fires GROUP BY LATITUDE ORDER BY count(*)DESC LIMIT 1
	32.5%: Missing Constraint
Question	How many nuclear power plants are in preparation to be used in Japan?
Predicted	SELECT Count(*)FROM nuclear_power_plants WHERE Country =’□’
Gold	SELECT count(*)FROM nuclear_power_plants WHERE Country = "Japan"AND Status = "Under Construction"
	31.25%: Incorrect Constraint
Question	Which state gets the highest revenue?
Predicted	SELECT NDECoreExcel_Math_Grade8.state FROM FINREV_FED_17 JOIN NDECoreExcel_Math_Grade8 GROUP BY NDECoreExcel_Math_Grade8.state ORDER BY Sum(FINREV_FED_17.t_fed_rev)Asc
Gold	SELECT T2.state FROM FINREV_FED_KEY_17 as T2 JOIN FINREV_FED_17 as T1 ON T1.state_code = T2.state_code GROUP BY T2.state ORDER BY sum(t_fed_rev) DESC LIMIT 1
	15%: Entity-column matching
Question	Which type of crime happens the most in Salford?
Predicted	SELECT Type FROM GreaterManchesterCrime WHERE Location LIKE ’□’ GROUP BY Type ORDER BY Count(*)Desc LIMIT 1
Gold	SELECT Type FROM GreaterManchesterCrime WHERE LSOA LIKE "%Salford%"GROUP BY Type ORDER BY count(*)DESC LIMIT 1
	13.75%: Understanding Error
Question	How many downloads of ep and album respectively?
Predicted	SELECT Sum(totalSnatched), Sum(totalSnatched)FROM torrents WHERE releaseType =’□’
Gold	SELECT sum(totalSnatched)FROM torrents WHERE releaseType = "ep"UNION SELECT sum(totalSnatched)FROM torrents WHERE releaseType = "album"

descending order. **(iv)** 15% of the errors are in “Entity-column matching”, *e.g.* aligning “Salford” to `Location` rather than `LSOA`. This illustrates the difficulty of *value linking*, partly mitigated by value descriptions for categorical columns in the database documentation.

5 Conclusion & Future Work

KaggleDBQA provides two resources to facilitate real-world applications of text-to-SQL parsing. First, it encourages an evaluation regime that bridges the gap between academic and industrial settings, leveraging in-domain knowledge and more realistic database distribution. We encourage adopting this regime for established text-to-SQL benchmarks. Second, it is a new dataset of more realistic databases and questions, presenting a challenge to state-of-the-art parsers. Despite the addition of domain knowledge in the form of database documentation, our baselines reach only 26.77% accuracy, struggling to generalize to harder questions. We hope that better use of documentation and new modeling and domain adaptation techniques will help further advance state of the art. The KaggleDBQA dataset is available at https://aka.ms/KaggleDBQA.

Ethical Considerations

Dataset Collection The data collection process was pre-approved by IRB. Each annotator agreed to a consent form before having access to the labeling task. Each annotator was rewarded with a $20 e-gift card for the approximately one hour of their time. The authors of this paper acted as the SQL annotators and incurred no additional compensation. The databases collected for KaggleDBQA were individually reviewed to ensure they were properly licensed for re-distribution. For other details of dataset construction, please refer to Section 3.

Aside from email addresses, no personal information of annotators was collected during our study. Email addresses were not shared and were promptly deleted after compensation had been provided. The association between annotator and annotation was deleted before any analysis or distribution was conducted.

Language Distribution KaggleDBQA only includes question annotations and databases in English, thus evaluating multi-lingual text-to-SQL models on it will require translation. The set of annotators included both native and second-language speakers of English, all fluent.

Usage of DBQA Technology Our goal with KaggleDBQA is to encourage the development of DBQA that will work in real-world settings. The actual deployment of a text-to-SQL parser must be conducted with appropriate safeguards in place to ensure users understand that the answers may be incorrect, especially if those answers are to be used in decision making.

References

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 *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*.

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

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-SQL in cross-domain database with intermediate representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535, Florence, Italy. Association for Computational Linguistics.

Po-Sen Huang, Chenglong Wang, Rishabh Singh, Wen-tau Yih, and Xiaodong He. 2018. Natural language to structured query generation via meta-learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 732–738, New Orleans, Louisiana. Association for Computational Linguistics.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, Dmitry Tsarkov, Xiao Wang, Marc van Zee, and Olivier Bousquet. 2020. Measuring compositional generalization: A comprehensive method on realistic data. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Zechang Li, Yuxuan Lai, Yansong Feng, and Dongyan Zhao. 2020. Domain adaptation for semantic parsing. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 3723–3729. ijcai.org.

Zhuang Li, Lizhen Qu, Shuo Huang, and Gholamreza Haffari. 2021. Few-shot semantic parsing for new predicates. *arXiv preprint arXiv:2101.10708*.

Percy Liang. 2016. Learning executable semantic parsers for natural language understanding. *Communications of the ACM*, 59(9):68–76.

Lajanugen Logeswaran, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, Jacob Devlin, and Honglak Lee. 2019. Zero-shot entity linking by reading entity descriptions. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3449–3460, Florence, Italy. Association for Computational Linguistics.

Subhabrata Mukherjee and Ahmed Awadallah. 2020. Uncertainty-aware self-training for few-shot text classification. *Advances in Neural Information Processing Systems*, 33.

Sheena Panthaplackel, Milos Gligoric, Raymond J Mooney, and Junyi Jessy Li. 2020. Associating natural language comment and source code entities. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8592–8599.

Karthik Radhakrishnan, Arvind Srikantan, and Xi Victoria Lin. 2020. Colloql: Robust cross-domain text-to-sql over search queries. *arXiv preprint arXiv:2010.09927*.

Abhinav Rastogi, Xiaoxue Zang, Srinivas Sunkara, Raghav Gupta, and Pranav Khaitan. 2020. Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8689–8696.

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.

Yibo Sun, Duyu Tang, Nan Duan, Yeyun Gong, Xiaocheng Feng, Bing Qin, and Daxin Jiang. 2020. Neural semantic parsing in low-resource settings with back-translation and meta-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8960–8967.

Harm de Vries, Dzmitry Bahdanau, and Christopher Manning. 2020. Towards ecologically valid research on language user interfaces. *arXiv preprint arXiv:2007.14435*.

Bailin Wang, Mirella Lapata, and Ivan Titov. 2020a. Meta-learning for domain generalization in semantic parsing. *arXiv preprint arXiv:2010.11988*.

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. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, Online. Association for Computational Linguistics.

Bolin Wei, Ge Li, Xin Xia, Zhiyi Fu, and Zhi Jin. 2019. Code generation as a dual task of code summarization. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 6559–6569.

Orion Weller, Nicholas Lourie, Matt Gardner, and Matthew Peters. 2020. Learning from task descriptions. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1361–1375, Online. 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. 2020. Grappa: Grammar-augmented pre-training for table semantic parsing. *arXiv preprint arXiv:2009.13845*.

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.

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

Jichuan Zeng, Xi Victoria Lin, Steven C.H. Hoi, Richard Socher, Caiming Xiong, Michael Lyu, and Irwin King. 2020. Photon: A robust cross-domain text-to-SQL system. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 204–214, Online. Association for Computational Linguistics.

Rui Zhang, Tao Yu, Heyang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019. Editing-based SQL query generation for cross-domain context-dependent questions. 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 5338–5349, Hong Kong, China. Association for Computational Linguistics.

Yusen Zhang, Xiangyu Dong, Shuaichen Chang, Tao Yu, Peng Shi, and Rui Zhang. 2020. Did you ask a good question? a cross-domain question intention classification benchmark for text-to-sql. *arXiv preprint arXiv:2010.12634*.

Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic evaluation for text-to-sql with distilled test suite. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 396–411.

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

Table 10: Evaluation results on KaggleDBQA using 100% of the evaluation data. All numbers are the exact match accuracy evaluated by the Spider official scripts. Here we report the average score of three runs with different random seeds.

Models	Nuclear	Crime	Pesticide	MathScore	Baseball	Fires	WhatCD	Soccer	Avg
RATSQL	22.91	23.45	8.00	0.00	11.11	25.22	4.76	11.11	13.32
w. desc	21.87	20.98	9.99	0.00	11.11	18.01	6.50	11.11	12.44
w. adaptation	20.83	33.33	12.66	3.57	11.11	24.32	8.93	12.96	15.96
w. desc + adaptation	29.16	25.88	18.00	3.57	16.23	30.62	10.53	12.96	18.37

Table 11: The original user question distribution. This reflects the natural information need from users.

Question Types	#	Example
Yes/No Percentage	51	Has there been a recent surge in violent crime in Manchester? What percentage of August crime detections resulted in prosecution of a suspect?
Time-related	46	Divide the day into 3 slots (6am to 4pm, 4pm to 11pm, 11pm to 6am), which has the highest amount of crime conducted per hour?
SQL-unexpressible	31	Which states had the highest percentage change in average scores over the last few years?
SQL-expressible	272	Which LSOA has had the most instances of bicycle theft this month?

A Appendix

A.1 Evaluation on Full Testing Data

We show the zero shot testing and out-of-domain adaptation results in Table 10. In contrast to Table 4, they are evaluated using the full set of testing data.

A.2 Details of Dataset Construction

A.2.1 Example Page of User Instructions

For each user, we show two different HTML files that contain different instructions of the task, database overview, table name(s), column descriptions, ten sampled rows of the database content.

A.2.2 Question Types

Question annotators were allowed to write any type of question without restriction. While this represents a natural distribution of questions one might expect to encounter in a realistic setting, some types do not appear in the Spider training set and thus pose particular difficulty with current text-to-SQL systems. We remove these from the official evaluation but still include them in the dataset for future work on these types of questions. Table 11 summarizes the distribution over these types of questions.

A.2.3 SQL annotation Guidelines

We also establish few guidelines and follow them throughout the annotation process:

1. If the referred column is categorical, use "=" operator with the value from the database (e.g., *Where is the area with the largest number of sexual offenses crime events?* → `SELECT Location FROM GreaterManchesterCrime WHERE Type = "Violence and sexual offences"GROUP BY Location ORDER BY count(*)DESC LIMIT 1`). If it is free-form text use "LIKE" operator with a term from the question (e.g., *What were the closing odds for a draw in matches with VfB Stuttgart?* → `SELECT DRAW_CLOSING FROM betfront WHERE MATCH LIKE "%VfB Stuttgart%"`).

2. Sometimes ID columns are paired with their name realizations (e.g., `state_code` and `state`). We choose to return ID whenever users do not explicitly ask for the name realizations.

3. Duplicate rows can sometimes yield an incorrect result. However, it is not possible for models to know in advance unless they encode database content. So we use the `DISTINCT` operator when necessary to return the correct answer or it is explicitly asked for by the user (e.g., *What are titles for each unique entry?*).

A.3 Implementation Details

For all our experiments we use the RAT-SQL official implementation and the pre-trained BERT-Large from Google. [6] We follow the original settings to get the pre-fine-tuned/pre-adapted models.

[6]We use the BERT-Large, Uncased (Whole Word Masking) model from https://storage.googleapis.com/bert_models/2019_05_30/wwm_uncased_L-24_H-1024_A-16.zip

Table 12: Exact match accuracy and standard error on schema-normalized KaggleDBQA, average of three runs with different random seeds.

With *fine-tuning*									
Models	Nuclear	Crime	Pesticide	MathScore	Baseball	Fires	WhatCD	Soccer	Avg
(a) RAT-SQL	25.75	44.44	23.52	7.01	19.74	33.33	22.61	8.33	23.09 ± 0.9%
(b) *w.* desc	25.75	40.73	19.60	3.50	20.98	28.00	25.00	8.33	21.48 ± 1.0%
(c) *w. adaptation*	30.30	46.29	19.60	12.27	19.74	41.33	21.42	13.88	25.60 ± 0.9%
(d) *w.* desc + *adaptation*	33.33	49.99	28.43	8.76	22.21	37.33	26.18	16.44	27.86 ± 0.7%
Without *fine-tuning*									
Models	Nuclear	Crime	Pesticide	MathScore	Baseball	Fires	WhatCD	Soccer	Avg
(e) RAT-SQL	30.29	35.18	15.68	0.05	12.34	22.66	5.95	25.00	19.04 ± 0.6%
(f) *w.* desc	24.23	25.92	13.72	0.00	0.08	13.33	0.07	13.87	13.35 ± 0.9%
(g) *w. adaptation*	25.75	40.73	21.56	14.02	14.81	25.33	10.69	25.00	22.23 ± 0.7%
(h) *w.* desc + *adaptation*	34.84	37.03	23.52	8.76	18.51	24.00	16.66	21.96	23.16 ± 0.5%

For adaptation and fine-tuning, we decrease the learning rate of BERT parameters by 50 times to 6e-8 to avoid overfitting. We keep the learning rate of non-BERT parameters the same at 7.44e-4. We also increase the dropout rate of the transformers from 0.1 to 0.3 to provide further regularization.