# Exploring Unexplored Generalization Challenges for Cross-Database Semantic Parsing

**Alane Suhr**[‡][*], **Ming-Wei Chang**[†], **Peter Shaw**[†], and **Kenton Lee**[†]

[‡] Cornell University Department of Computer Science and Cornell Tech
New York, NY 10044
suhr@cs.cornell.edu

[†]Google Research
{mingweichang, petershaw, kentonl}@google.com

## Abstract

We study the task of cross-database semantic parsing (XSP), where a system that maps natural language utterances to executable SQL queries is evaluated on databases unseen during training. Recently, several datasets, including Spider, were proposed to support development of XSP systems. We propose a challenging evaluation setup for cross-database semantic parsing, focusing on variation across database schemas and in-domain language use. We re-purpose eight semantic parsing datasets that have been well-studied in the setting where in-domain training data is available, and instead use them as additional evaluation data for XSP systems instead. We build a system that performs well on Spider, and find that it struggles to generalize to our re-purposed set. Our setup uncovers several generalization challenges for cross-database semantic parsing, demonstrating the need to use and develop diverse training and evaluation datasets.

## 1 Introduction

Semantic parsing is the task of mapping natural language utterances to formal meaning representations, and has been studied in tasks including instruction following, evaluating sentence meaning, and building interfaces to knowledge bases. In this paper, we focus on the task of mapping from natural language utterances to SQL queries executable in a database. Most prior work in mapping from natural language to SQL queries train and test the system on a single database. We refer to this setup as *single-database semantic parsing* (**SSP**). Well-studied datasets used in the SSP setting include GeoQuery (Zelle and Mooney, 1996) and ATIS (Hemphill et al., 1990; Dahl et al., 1994).

However, semantic parsing systems should be able to generalize to new domains and databases, as it is often cost-prohibitive to collect a sufficient number of training examples for all possible databases. Several datasets, including Spider (Yu et al., 2018), were proposed to evaluate this dimension of generalization. These datasets include

Advising (Finegan-Dollak et al., 2018)

**NL**: *For EECS 478, how many credits is it?*

**SQL**:
```
select distinct credits from
course where department ='EECS'
and number = 478;
```

GeoQuery (Zelle and Mooney, 1996)

**NL**: *How many people live in mississippi?*

**SQL**:
```
select population from state where
state_name = 'mississippi';
```

ATIS (Hemphill et al., 1990; Dahl et al., 1994)

**NL**: *Flights from Phoenix to Milwaukee*

**SQL**:
```
select distinct T1.flight_id
from airport_service as T2,
airport_service as T3, city as
T4, city as T5, flight as T1
where T4.city_code = T2.city_code
and T4.city_name = 'Phoenix'
and T5.city_code = T3.city_code
and T5.city_name = 'Milwaukee'
and T1.from_airport =
T2.airport_code and T1.to_airport =
T3.airport_code;
```

Spider (Yu et al., 2018)

**NL**: *List the emails of the professionals who live in the state of Hawaii or the state of Wisconsin.*

**SQL**:
```
select email_address from
professionals where state =
'Hawaii' or state = 'Wisconsin';
```

Figure 1: Examples of generalization challenges revealed in the cross-database semantic parsing (XSP) setting. The top three examples are from datasets originally studied in the single-database (SSP) setting. Without in-domain training data, generalization is more difficult, requiring identifying entities, mapping unfamiliar phrases and entities to the database, and using new and complex database schemas. In contrast, existing XSP evaluation data such as Spider simplifies some of these challenges, for example by including utterances that closely match their paired SQL query.

[*] Work done during an internship at Google.

examples grounded in multiple databases, distinguishing between training databases and evaluation databases. We refer to this setup as *cross-database semantic parsing* (**XSP**).

While these datasets have been valuable in understanding and addressing some of the additional generalization challenges introduced by XSP, current evaluation of XSP systems has been limited to datasets designed for XSP. This limits the types of generalization challenges studied to those introduced by these datasets. Existing XSP evaluation data such as Spider simplifies some of these challenges, for example by including utterances that closely match their paired SQL query, as shown in last row of Figure 1.

This setup misses an important opportunity for studying cross-database semantic parsing: evaluating on challenging datasets designed for single-database semantic parsing, like GeoQuery and ATIS. While the in-domain challenges of these datasets are relatively well-understood, generalization challenges introduced by studying these datasets in an XSP context have not been addressed.

In this paper, we propose a more holistic analysis and evaluation setup for XSP. We propose to evaluate a semantic parsing system not only on evaluation data designed for XSP, but also on datasets that have only been studied in the SSP setting. Our repurposed evaluation set includes eight well-studied datasets like ATIS, but in a completely new setting. Instead of training on the original training data for these datasets, we train a single model on training data designed for the XSP setting, and evaluate the trained model on each evaluation dataset. These datasets were collected at different times, by different researchers, and with different motivations. This results in a wide variety of language usage, database structures, and SQL styles across datasets, further stressing a system's ability to adapt to unseen datasets. These variations pose many new generalization challenges for cross-database semantic parsing models, where in-domain examples are not available at training time. Our proposed XSP evaluation setup addresses several evaluation challenges posed by these dataset variations.

With our proposed setup, we are able to analyze the potential limitations of current cross-database semantic parsing models. We uncover and attempt to address several new forms of generalization in cross-dataset semantic parsing. We develop a neural semantic parsing model is competitive all public systems on the Spider development set, and evaluate its ability to generalize to the evaluation datasets. First, we observe that the datasets originally designed for SSP become much more difficult under the XSP setting, with a notable drop in performance from both the Spider development results. Second, we experiment with several techniques that improve generalization to the eight evaluation datasets. Finally, we provide in-depth qualitative analysis on our results. Our results and analysis demonstrate a need for diverse training and evaluation datasets for XSP. Our code and experimental setup is available at `https://github.com/google-research/language/tree/master/language/xsp`.

## 2 Background and Related Work

We focus on the task of semantic parsing for databases. A natural language utterance $\overline{u}$ is a sequence $\langle u_1, \ldots, u_{|\overline{u}|}\rangle$, where each $u_i$ is a natural language token. The task is to map $\overline{u}$ to an executable formal query $\overline{y} = \langle y_1, \ldots, y_{|\overline{y}|}\rangle$ executable in a database $\mathcal{D}$, where each $y_i$ is a SQL query token.

**Single-database Semantic Parsing (SSP)** In SSP, all data is grounded in the same knowledge database. The training data consists of $N$ pairs of utterances and SQL queries $\{\overline{x}^{(l)}, \overline{y}^{(l)}\}_{l=1}^{N}$ grounded in database $\mathcal{D}$. The evaluation data contains $M$ unseen pairs of utterances and SQL queries $\{\overline{x}^{(l)}, \overline{y}^{(l)}\}_{l=1}^{M}$, also grounded in $\mathcal{D}$.

SSP has been studied using a number of datasets including ATIS (Hemphill et al., 1990; Dahl et al., 1994) and GeoQuery (Zelle and Mooney, 1996). Many prior approaches in SSP assume access to database contents at inference time. At test time, this allows the system to resolve the columns containing novel entities by performing a database look-up; for example, by labeling entity mentions in the input utterance with the columns in which they appear (Dong and Lapata, 2016; Iyer et al., 2017; Suhr et al., 2018).

**Cross-database Semantic Parsing (XSP)** In the XSP setting, examples from the evaluation databases are not seen at training time (Yu et al., 2018, 2019b,a). Previously, the cross-*domain* semantic parsing task focused mostly on databases consisting of a single table (Pasupat and Liang, 2015; Iyyer et al., 2017; Zhong et al., 2017). However, the cross-*database* setting requires generalizing to unseen domains and novel database schemas.

In XSP, the $N$ training examples are $\{\overline{x}^{(l)}, \overline{y}^{(l)}, \mathcal{D}_i^{(l)}\}_{l=1}^N$ and the $M$ evaluation examples are $\{\overline{x}^{(l)}, \overline{y}^{(l)}, \mathcal{D}_j^{(l)}\}_{l=1}^N$, where each $\mathcal{D}$ is a database. Importantly, the set of training and evaluation datasets do not overlap. In addition to the generalization challenges posed by SSP, this setting adds several challenges, including generalizing to new schema structures, domain-specific phrases, and database conventions.

Unlike SSP, prior work in XSP does not assume that the system has access to database contents at model inference time (Yu et al., 2018). Preprocessing steps that perform database look-ups are unavailable at inference time. Instead, the model only has access to the database schema for each evaluation example. This setting requires additional generalization, where the model must be able to map unfamiliar entities to columns in domains unseen during training.

**Other Related Work** Semantic parsing has been widely studied for tasks including sentence understanding (Zettlemoyer and Collins, 2005, 2007; Banarescu et al., 2013), instruction following (Chen and Mooney, 2011; Artzi and Zettlemoyer, 2013; Long et al., 2016; Givoli and Reichart, 2019), and knowledge base querying (Popescu et al., 2004; Poon, 2013; Iyer et al., 2017). Related to the task of semantic parsing is code generation (Oda et al., 2015; Ling et al., 2016; Yin et al., 2018; Lin et al., 2018; Iyer et al., 2018). While our experiments are performed on English-langauge data, a limited amount of existing work has explored semantic parsing in languages besides English (Wong and Mooney, 2006; Min et al., 2019).

Annotating SQL queries for new domains can be expensive. Several prior works present approaches to reduce this cost, for example by having crowdworkers paraphrase generated examples (Wang et al., 2015; Zhong et al., 2017), give feedback (Iyer et al., 2017), interact with a system (Artzi and Zettlemoyer, 2011; Thomason et al., 2015; Labutov et al., 2018), or a combination (Herzig and Berant, 2019).

Research in SSP and code generation has led to innovations including constrained decoding and grammar-based decoding (Xiao et al., 2016; Yin and Neubig, 2017; Krishnamurthy et al., 2017; Lin et al., 2019). SSP has also been studied alongside additional generalization challenges, including to new compositional structures (Finegan-Dollak et al., 2018) and with additional context (Miller et al., 1996; Zettlemoyer and Collins, 2009; Suhr et al., 2018). Recent works evaluating in the XSP setting have explored methods of jointly embedding an utterance and the database schema (Shaw et al., 2019; Bogin et al., 2019a), interactive learning (Yao et al., 2019), and using intermediate output representations and new inference methods (Herzig and Berant, 2018; Guo et al., 2019; Zhang et al., 2019; Bogin et al., 2019b; Lin et al., 2019). We incorporate several such methods proposed into our proposed system.

## 3 Evaluating on Re-purposed Data

We propose to study the task of XSP by training on datasets designed for XSP, and evaluating on datasets originally designed for SSP. In our full model, we use both the Spider[1] (Yu et al., 2018) and WikiSQL (Zhong et al., 2017) datasets for training. For evaluation, in addition to the Spider development set,[2] we use eight English-language SSP datasets curated by Finegan-Dollak et al. (2018) covering a variety of domains, for example flights, geography, and movies.[3] For each dataset, we evaluate on as much data as possible, excluding test sets. Table 1 describes our evaluation datasets.

Developing evaluation metrics for these repurposed evaluation sets is challenging because of the diversity of SQL styles across different databases. Yu et al. (2018)'s proposed evaluation metric compares components of the predicted and correct query, allowing for variation in the exact form of the query, for example using different table aliases. However, it does not capture all possible SQL syntax, and fails to cover some of the gold queries in our evaluation datasets. For example, it does not handle assigning an alias to the results of an intermediate `SELECT` statement. Moreover, it does not measure equivalence of values, meaning

[1]In addition to introducing Spider, Yu et al. (2018) propose to use a number of SSP datasets, including GeoQuery, as additional training examples for systems evaluated on Spider. However, these SSP datasets were not previously used as evaluation data in the XSP setting. During training, we use only the original Spider data, and discard this additional training data used by some Spider systems.

[2]WikiSQL contains much more simplified language, SQL queries, and databases than Spider. Therefore, we focus on Spider as part of our proposed XSP evaluation setup.

[3]Finegan-Dollak et al. (2018) re-split these datasets to evaluate generalization to novel query structures. However, this work still operates in the SSP setting, where in-domain training examples are available. Our setup uses the original splits of the data, rather than the structure-based splits (Table 1).

| Original Task | Dataset | Splits | # Examples (All/Filtered) | % Col. Mentioned |
|---|---|---|---|---|
| SSP | ATIS (Hemphill et al., 1990; Dahl et al., 1994) | dev | 486/289 | 0.2 |
| | GeoQuery (Zelle and Mooney, 1996) | train/dev | 598/532 | 32.4 |
| | Restaurants (Tang and Mooney, 2000) | splits 0–9 | 378/ 27 | 0.0 |
| | Academic (Li and Jagadish, 2014) | splits 0–9 | 196/180 | 8.2 |
| | IMDB (Yaghmazadeh et al., 2017) | splits 0–9 | 131/107 | 4.6 |
| | Yelp (Yaghmazadeh et al., 2017) | splits 0–9 | 128/ 54 | 7.0 |
| | Scholar (Iyer et al., 2017) | train/dev | 599/394 | 1.0 |
| | Advising (Finegan-Dollak et al., 2018) | train/dev | 2858/309 | 0.5 |
| XSP | Spider (Yu et al., 2018) | dev | 1034/ – | 72.4 |

Table 1: Basic statistics for our evaluation datasets. We use all ten cross-validation sets for Restaurants, Academic, IMDB, and Yelp. *Filtered* refers to the focused subset of evaluation data where relative performance of systems is more meaningful, as we removed examples that yield empty tables and those that are likely impossible to solve due to dataset conventions. *% Col. Mentioned* shows the estimated proportion of examples in the evaluation set where all columns compared against entities in the gold query are explicitly mentioned in the utterance.

predictions correct according to this metric need not execute correctly.

We propose to use variation of execution accuracy as our main metric. Execution accuracy over an evaluation set is the proportion of predicted queries which, when executed against the database, result in a table equivalent to the correct query's result. If the correct query requires ordering on the final table, we require the tables be exactly the same; if it does not, we consider result tables equivalent if they contain the same set of rows. We supplement the results with additional baselines and data filtering to address the problem of over-crediting spurious predictions. We report the empty-table prior for each dataset, demonstrating how well a model could perform if predicting incorrect queries that result in empty tables. We create a filtered subset where relative performance of systems is more meaningful, including attempting to remove examples that are impossible to solve without in-domain training data. Our heuristics include removing examples with correct queries that result in empty tables, and where the correct query contains a value token that is not copiable from the input.[4] For example, in Restaurants, the phrase *good restaurant* always requires constraining the SQL query to restaurants with a rating greater than 2.5, even when the rating is not explicitly mentioned.

## 4 Generalization Challenges

Single-database semantic parsing requires recognizing unseen, in-domain entities, understanding new compositional structures, and generating executable representations. Cross-database semantic parsing introduces additional challenges, which we analyze and discuss below. We find that with existing XSP datasets, these challenges have been relatively under-explored. In our proposed setup, where we evaluate on datasets designed for SSP, these challenges become more prevalent.

### 4.1 Language Variation Across Domains

Generalizing to a new domain requires understanding domain-particular language, including entity mentions and their types, and how to map domain-specific phrases to SQL queries.

**Identifying Entities** In the XSP setting, identifying spans of tokens comprising relevant entities in the utterance is difficult, especially without access to the database contents. For example, in some databases, first and last names are stored in separate columns, so the corresponding tokens should appear in different parts of the SQL query. In other databases, a single column is used to store names. Even if a model is trained on databases where names are always stored in a single column, it still must generalize to databases where first and last names are stored in separate columns. This becomes more challenging with domain-specific entities. For example, in the Advising example in Figure 1, the span *EECS 478* refers to two distinct database entities, rather than a single entity. This requires taking into account the database schema, for example by considering that the `course` table has distinct columns for `department` and `number`.

**Mapping Entities to Columns** Mapping a natural language utterance to an executable SQL query requires correctly identifying which columns and

[4]Details are included in Appendix A.

tables each entity should be compared with. Consider the following example (from GeoQuery):

**NL**: *what states are next to the mississippi*
**SQL**:
```
select traverse
from river where
river_name = 'mississippi';
```

To correctly identify that the entity *mississippi* refers to a `river_name` in the `river` table, the system must have some domain knowledge. *mississippi* appears twice in the database: as a state and as a river. Even in the SSP setup, if the system has access to database contents, this entity mention's type is ambiguous without reasoning about its context in the utterance. In the XSP setup, this problem becomes even more difficult. Database contents are not available at model inference time, so an exhaustive search over the database for matching columns is not possible. Without in-domain training data, the model must still be able to choose the most likely column match for each mentioned entity.

However, sometimes the column name is explicitly mentioned in the utterance, making the matching problem much easier, as demonstrated in the Spider example of Figure 1. We measure how prevalent the challenge of mapping from entities to column names is in our XSP setup. In each evaluation set, we estimate the proportion of examples whose entity mentions can be resolved using exact string matching between the utterance and the schema.[5] Yavuz et al. (2018) perform a similar analysis manually on the WikiSQL dataset, estimating that roughly 54.1% of examples can be solved using exact match. The rightmost column in Table 1 compares all eight evaluation datasets and the Spider development set. In all eight evaluation datasets originally developed for SSP, fewer than half of examples explicitly mention column names for all entities in the utterance. In contrast, all column names are explicitly mentioned in least 72.4% of examples in the Spider development set. These results demonstrate that addressing this challenge is critical to XSP on completely unseen domains.

**Domain-Specific Phrases** Generalizing to new domains requires mapping domain-specific phrases to implementations in SQL. Consider the following examples (from GeoQuery):

[5]More details on this analysis are available in Appendix A.

**NL**: *what is the smallest city in arkansas*
**SQL**:
```
select city_name from
city where population =
(select min (population)
from city where state_name =
'arkansas') and state_name =
'arkansas'
```

**NL**: *what is the smallest state that borders texas*
**SQL**:
```
select state_name from state
where area = (select min (area)
from state where state_name in
(select border from border_info
where state_name = 'texas'))
and state_name in (select
border from border_info where
state_name = 'texas')
```

When *smallest* describes a city, it requires sorting by the `city.population` column, but when used to describe a state, it requires sorting by the `state.area` column, even though the `state` table also has a `population` column. Another phrase whose implementation may change in a new database is *how many*. This phrase is often mapped to the `count` operator, but is sometimes mapped to specific database columns. For example, in Figure 1, *how many credits* maps to the `credits` table in Advising, and *how many people* maps to the `population` table in GeoQuery. To scope the problem, Yu et al. (2018) avoid including examples in Spider that require commonsense reasoning, including examples of domain-specific phrases. However, understanding domain-specific phrases is an important capability for a domain-general semantic parsing system.

## 4.2 Novel Database and Query Structures

Cross-database semantic parsing requires generalizing to new database schemas, including larger tables and compositions of SQL components. Four of our evaluation datasets have at least ten tables in the database, with the largest database being ATIS with 32 tables.[6] Figure 1 demonstrates that generating queries for large databases such as ATIS often requires reasoning about the relationships between many tables. In contrast, our training databases are relatively small, with one table per example in WikiSQL and an average of 5.1 per database in Spider. The queries themselves also vary in complexity. Finegan-Dollak et al. (2018) show that our eight target evaluation datasets range from using 1.4 to 6.4 tables per SQL query. We estimate that in Spider, an average query uses around 1.7 tables, which

[6]Finegan-Dollak et al. (2018) and Yu et al. (2018) provide comprehensive statistics on the databases and gold queries in our evaluation domains.

is more than only one target dataset (GeoQuery). Generalizing to new databases and datasets in our setting requires generating queries that use more tables than the training data.

### 4.3 Dataset Conventions

In some evaluation datasets, the system must not only reason about the input utterance and schema, but about dataset-specific conventions that are not specified in the inputs. Consider the following example (from Scholar):

**NL**: *papers on semantic parsing*
**SQL**:
```
select distinct T1.paperid from
keyphrase as T2, paper as T1,
paperkeyphrase as T3, where
T2.keyphrasename = 'semantic
parsing' and T3.keyphraseid =
T2.keyphraseid and T1.paperid =
T3.paperid;
```

The annotated SQL query for this utterance returns the `paperid` column from the `paper` table. However, the `paper` table also includes a column named `title`. The utterance does not specify whether the final column should be `paperid` or `title`. While both columns may seem like reasonable options, the dataset's convention is that a list of papers should be presented using the `paperid` column, and a query selecting the `title` column will have an incorrect execution result. Such conventions are difficult, if not impossible, to learn without any in-domain training data. Unfortunately, these cases occur in nearly all target datasets. We do not focus on addressing this type of generalization, and instead report how pervasive this problem is during error analysis. A possible direction for future work is to assume access to a small number of in-domain training examples and perform few-shot learning.

## 5 Model and Learning

Our model takes as input an utterance $\overline{x}$ and a database schema $\mathcal{S}$. Similar to Guo et al. (2019), we serialize $\mathcal{S}$ into a sequence of wordpieces $\overline{s} = \overline{t}_0 + \overline{t}_1 + \cdots + \overline{t}_{|\mathcal{S}|}$. Each $\overline{t}_i$ is a serialization of table $\mathcal{T}_i$, where $\overline{t}_i = \langle \texttt{TAB} \rangle + \overline{T}_i + \overline{c}_{i,0} + \overline{c}_{i,1} + \dots \overline{c}_{i,|\mathcal{T}_i|}$. `TAB` is a token noting the beginning of a table schema serialization. $\overline{T}_i$ is the tokenization of table $\mathcal{T}_i$'s name. Each $\overline{c}_{i,j}$ is a serialization of a column $\mathcal{C}_{i,j}$, where $\overline{c}_{i,j} = \texttt{CT}_{i,j} + \overline{C}_{i,j}$. $\texttt{CT}_{i,j}$ is a token denoting the type of the column's contents as provided by the database schema, for example numerical or text. $\overline{C}_{i,j}$ is the tokenization of the column's name. The ordering of table schemas in $\overline{s}$ and table columns in each $\overline{t}_i$ is arbitrary.[7] The input to the encoder is the concatenation of the query wordpieces and the serialized schema, represented as the sequence of tokens $\overline{x} = \langle \texttt{CLS} \rangle + \overline{u} + \langle \texttt{SEP} \rangle + \overline{s}$. The inputs to the encoder are embedded and passed to a pretrained Transformer encoder such as BERT (Devlin et al., 2019). The decoder is an autoregressive Transformer decoder (Vaswani et al., 2017) that attends over the outputs of the encoder and the generated prefix.

We use a training set $\{\overline{x}^{(l)}, \overline{y}^{(l)}, \mathcal{S}^{(l)}\}_{l=1}^{N}$ consisting of pairs of natural language utterances, gold SQL queries, and database schemas. We train the encoder and decoder end-to-end, minimizing the token-level cross-entropy loss of the gold query $\overline{y}^{(l)}$. We update the parameters of the pre-trained encoder during training. For training data we use training sets developed for XSP. Importantly, to ensure we are evaluating the *cross*-database setting, our training data does not include examples from the evaluation databases. During inference, we use beam search and execute the highest-probability, syntactically correct prediction. We impose a maximum execution time of 45 seconds for predictions. More details on the model, learning, and evaluation setup are available in Appendix B.

### 5.1 Generalization Strategies

While using pre-trained language models can help encode natural language text, we need other strategies to reason jointly about the language and the database schema in completely unseen domains. We focus on generalizing to domain-specific language and novel database structures.

**Value Copying** Similar to previous work (Jia and Liang, 2016; Gu et al., 2016; Gulcehre et al., 2016; See et al., 2017), we use a copy mechanism in the decoder. At each output step, the decoder generates a distribution over possible actions, including selecting a symbol from the output vocabulary, and copying a token from the input $\overline{x}$. We only allow copying of certain token types, and mask out invalid copying actions, including independent wordpieces from $\overline{u}$ and `TAB` and column-type tokens. For table and column tokens, the name of

[7] To discourage over-fitting to an arbitrary ordering of schema elements, we duplicate each Spider training example seven times with randomly permuted orderings. Duplicating seven times results in the number of Spider training examples roughly matching the number of WikiSQL training examples (Section 5.1).

the corresponding table or column is recovered by post-processing the predicted sequence $\overline{y}$.

Previous approaches on Spider do not evaluate execution accuracy over the databases. Because the main metric does not require values in the predicted and gold queries to be the same, many approaches simplify the problem by using a placeholder token for all values during training. However, correctly generating values is critical for correctly executing predicted queries. To the best of our knowledge, our approach is the first to evaluate on Spider with execution accuracy and to generate SQL queries without placeholder values.

**Multiple Data Sources** We train with training data from Spider (Yu et al., 2018) and WikiSQL (Zhong et al., 2017). Spider includes examples of complex SQL queries grounded in multi-table databases, while queries in WikiSQL are compositionally simple and grounded in single web tables. We use WikiSQL to improve generalization to domain-specific data, as it covers a large variety of domains. WikiSQL contains many more tables than Spider, and prior work estimates that roughly half of WikiSQL examples require using domain knowledge to map from entity mentions to column names (Yavuz et al., 2018).

**Different Output Space** Guo et al. (2019) demonstrated improvements on Spider by deterministically mapping SQL to an intermediate representation, SemQL, and learning to predict outputs in this space. SemQL does not require predicting all of the tables in the `FROM` clause of the SQL query, or explicitly predicting the columns on which tables are joined. Instead of reasoning about foreign keys, the model predicts queries in the SemQL space, which are deterministically transformed to a final SQL query. In most cases, SemQL queries can be mapped back to SQL using database foreign key relations. We implement this aspect of SemQL as a mapping from SQL to a representation with an under-specified `FROM` clause, which we call $\text{SQL}^{\text{UF}}$. Conversion from SQL to $\text{SQL}^{\text{UF}}$ removes tables from the `FROM` clause(s) of the SQL query implicitly referenced via a column elsewhere in the query, and removes `JOIN` clauses. Conversion from $\text{SQL}^{\text{UF}}$ to SQL restores these tables, and joins between tables are inferred by greedily identifying a path that connects all tables in the `FROM` clause, given foreign key relations.[8] Examples of $\text{SQL}^{\text{UF}}$ are shown in Appendix C.

## 6 Experiments

**Comparison to Existing XSP Systems** Our best model performs well on the Spider development set. Table 3 compares our system with top systems on the Spider leaderboard[9]. On the development set, our model performs competitively with contemporaneous systems. Table 3 shows that Spider performance correlated to the choice of the pre-trained models. Public $\text{BERT}_{\text{LARGE}}$ is better than $\text{BERT}_{\text{BASE}}$. To further improve performance, we experiment with an enhanced pre-trained model $\text{BERT}_{\text{LARGE+}}$ following the recipe proposed by Liu et al. (2019). The $\text{BERT}_{\text{LARGE+}}$ model is trained with 8K batch size and 100k training steps, and in contrast to RoBERTa, is only trained on the Wiki+Books Corpus used in Devlin et al. (2019). Training our model to predict value placeholders (– Value Copying) instead of copying values from the input results in a performance drop, showing a benefit of modeling values even when ignored by the metric.[10]

**XSP on Unseen Datasets** Table 2 shows results on all evaluation data, including datasets originally studied in the SSP setting. We report results on the filtered set (Section 3) as well as the full set of these datasets. A large portion of the examples in datasets such as Restaurants and Advising yield empty execution results. This shows the need to also evaluate on the filtered set, where incorrect spurious predictions are much less likely to result in the same table as a gold query with an empty table result. Second, while execution accuracy on Spider is relatively high, performance on the other evaluation datasets is much lower.

We find that all three techniques for addressing generalization challenges are effective. First, including WikiSQL in the training data results in better performance than only using Spider training data. We hypothesize that this is due to the addi-

[8]Like SemQL, this conversion is not possible if foreign key relations between predicted tables are not provided or if a given table is referenced more than once in a `FROM` clause. This can also result in a lossy or ambiguous conversion if there are multiple foreign key relations between a pair of tables.

[9]`https://yale-lily.github.io/spider`. In Table 3, we include non-anonymized leaderboard submissions, and for anonymous systems, the most recent submission for duplicate systems.

[10]About 55% of examples in Spider do not require copying values from the input utterance to the gold query.

| Dataset | Metric | # Examples | Our best | –WikiSQL | –SQL$^{UF}$ | – Value copying | Empty Prior |
|---|---|---|---|---|---|---|---|
| ATIS | *Execution* | 289 ( 486) | 0.8 (11.9) | 0.5 (11.9) | 0.8 (11.9) | 0.1 (10.8) | 0.0 (11.9) |
| GeoQuery | | 532 ( 598) | 41.6 (40.0) | 35.6 (35.0) | 34.7 (33.4) | 2.2 ( 5.6) | 0.0 ( 4.0) |
| Restaurants | | 27 ( 378) | 3.7 (45.2) | 3.7 (46.3) | 0.0 (46.6) | 0.0 (51.1) | 0.0 (51.6) |
| Academic | | 180 ( 196) | 8.2 (12.1) | 6.1 ( 9.4) | 5.7 ( 9.0) | 2.8 ( 7.7) | 0.0 ( 4.1) |
| IMDB | | 107 ( 131) | 24.6 (33.3) | 24.3 (32.3) | 23.1 (32.3) | 0.0 (14.3) | 0.0 (13.0) |
| Yelp | | 54 ( 128) | 19.8 (49.2) | 16.7 (47.9) | 14.8 (47.9) | 4.9 (53.1) | 0.0 (41.4) |
| Scholar | | 394 ( 599) | 0.5 ( 6.8) | 0.4 ( 7.4) | 0.5 ( 8.6) | 0.2 ( 7.8) | 0.0 ( 9.3) |
| Advising | | 309 (2858) | 2.3 (35.2) | 1.2 (35.7) | 1.4 (37.3) | 0.0 (38.0) | 0.0 (38.3) |
| Spider | *Execution* | 1034 | 69.0 | 68.4 | 65.1 | 33.9 | 4.7 |
| | *Exact Set Match* | | 65.0 | 65.1 | 60.5 | 54.1 | – |

Table 2: Execution accuracy on the XSP task for the eight evaluation datasets and Spider, comparing our best system with baselines and independent ablations. For Spider, we also report performance using Exact Set Match, the official Spider metric. Results are averaged over three trials. The full set results are reported in parentheses. The empty prior represents the baseline accuracy of returning empty set for all queries. The accuracies on the re-purposed datasets are much lower than the Spider performance.

| System | Exact Set Match (Dev.) |
|---|---|
| Top Leaderboard Systems (As of May 1, 2020) | |
| RYANSQL v2 + BERT (Choi et al., 2020) | 70.6 |
| RYANSQL + BERT (Choi et al., 2020) | 66.6 |
| RATSQL v2 + BERT (Anonymous) | 65.8 |
| IRNet++ + XLNET (Anonymous) | 65.5 |
| IRNet + BERT (Guo et al., 2019) | 61.9 |
| RASQL + BERT (Anonymous) | 60.8 |
| GIRN + BERT (Anonymous) | 60.2 |
| CNSQL (Anonymous) | 58.0 |
| EditSQL + LSL + BERT (Anonymous) | 57.9 |
| GNN + Bertrand-DR (Kelkar et al., 2020) | 57.9 |
| Ours with BERT$_{\text{LARGE+}}$ | 65.8 |
| Ours with BERT$_{\text{LARGE}}$ | 63.2 |
| Ours with BERT$_{\text{BASE}}$ | 60.4 |
| Ours with BERT$_{\text{LARGE+}}$ - Value Copying | 55.1 |

Table 3: Performance on the Spider development set using Spider's official evaluation metric (Exact Set Match), ordered by the development set performance. For our systems, we report the Exact Set Match of the best of three trials. While the focus of our paper is not on Spider performance, our system still performs well.

tional domains in WikiSQL, as well as the larger proportion of examples that require mapping from entities in the utterance to column names (Yavuz et al., 2018). Using SQL$^{UF}$ also improves performance, as it produces queries coherent with respect to the schema, for example only selecting columns from tables where the column exists. Finally, using value placeholders significantly reduces execution accuracy in all datasets. While masking values decreases Exact Set Match on Spider by 10.9%, its effect on execution accuracy can be devastating both for Spider and the eight evaluation datasets. This demonstrates the need to consider execution results when evaluating semantic parsing systems.

**Error Analysis** For each evaluation dataset, we analyze twenty random predictions from the filtered subset. Examples of the most common error types are shown in Figure 2, along with the proportion of analyzed predictions in the eight target datasets that contain the error type. Appendix D discusses the complete results of error analysis.

40% of errors are caused by comparing an entity to the wrong column, for example searching for `'James Bond'` in the `director.name` column when it actually refers to a `movie.title`. This usually requires using domain knowledge identify to columns that are likely to contain the mentioned entity (Section 4.1). 31.1% of errors are caused by missing constraints specified in the utterance, for example by failing to use a relevant entity in the predicted query. 28.8% of errors are also caused by incorrectly identifying entity spans, for example by treating `FIN 340` as a single entity rather than two separate entities in the database (Section 4.1). Another common error is predicting the wrong final column. While choosing what to return is difficult for the model due to understanding domain-specific phrases such as *how many* (20.0% of errors; Section 4.1), sometimes the errors are due to dataset conventions (26.9% of errors; Section 4.3). For example, the `paperid` column should be selected instead of the `title` column in Scholar. Such dataset conventions could be learned through few-shot learning, where a small number of in-domain training examples are available.

Our system is required to generalize to larger databases than it was trained on, including more complex compositions of tables (Section 4.2). For

40.0% → Entity-column matching (IMDB$_{\text{XSP}}$)

**NL**: *List "James Bond" directors*

**Pred.**:
```
select director.name
from directed_by join
director on directed_by.did
= director.did where
director.name = 'James Bond';
```
**Gold**:
```
select T1.name from directed_by
as T2, director as T1, movie
as T3 where T1.did = T2.did
and T3.mid = T2.mid and
T3.title = 'James Bond';
```

31.3% → Missing constraint (Academic$_{\text{XSP}}$)

**NL**: *return me the year of "Making database systems usable"*

**Pred**:
```
select publication.year from
publication;
```
**Gold**:
```
select T1.year from publication
as T1 where T1.title = 'Making
database systems usable';
```

28.8% → Entity identification and copying (Advising$_{\text{XSP}}$)

**NL**: *What's the number of times FIN 340 has been offered?*

**Pred.**:
```
select count(*) from course join
course_offering on course.course_id
= course_offering.course_id where
course.name = 'FIN 340';
```
**Gold**:
```
select count(distinct
T1.offering_id) from course as
T2, course_offering as T1 where
T2.course_id = T1.course_id
and T2.department = 'FIN' and
T2.number = 340;
```

26.9% → Ambiguous final column (Scholar$_{\text{XSP}}$)

**NL**: *papers from 2014*

**Pred.**:
```
select distinct paper.title from
paper where paper.year = 2014;
```
**Gold**:
```
select distinct T1.paperid from
paper as T1 where T1.year = 2014;
```

20.0% → Wrong final column (GeoQuery$_{\text{XSP}}$)

**NL**: *how many people live in austin*

**Pred.**:
```
select count(*) from city where
city.state_name = 'austin';
```
**Gold**:
```
select T1.population from city as
T1 where T1.city_name = 'austin';
```

Figure 2: The most common error types made by our best system, including an example. The subscript indicates the results are in the XSP setting. Each prediction may be annotated with more than one error type.

example, while SQL$^{\text{UF}}$ can be used to represent most gold queries in most evaluation datasets (shown in Appendix C), in ATIS, only 17.3% of gold queries are covered by SQL$^{\text{UF}}$. Most of the uncovered examples require mapping two columns, `to_airport` and `from_airport`, in the same table `flight` to the same foreign key `airport_service.airport_code`. This compositional structure is not covered by SQL$^{\text{UF}}$, but is critical to perform well on ATIS.

## 7 Discussion

We study the task of cross-database semantic parsing (XSP), where a system that maps natural language utterances to executable SQL queries is evaluated on databases unseen at training time. While this task has been studied through datasets developed specifically for XSP, we propose a more holistic evaluation for XSP, where we also evaluate on datasets originally studied in a setting where in-domain training data is available. We identify several new generalization challenges that arise when evaluating in our proposed setup, including identifying entities, mapping entities and domain-specific phrases to a database schema, and generalizing to more complex database schemas. Using a model that performs well on evaluation data designed for XSP, we are able to move towards addressing some of the generalization challenges on these additional evaluation sets without any in-domain training data. Our results and analysis demonstrate the need for developing more holistic evaluation of cross-database semantic parsing using a more diverse set of language and databases.

Several significant generalization challenges remaining, including improving commonsense and in-domain reasoning and table schema understanding capabilities. Some examples in our filtered evaluation set still require reasoning about dataset conventions that are difficult to acquire without in-domain training examples. Future work could also make the stronger assumption that a small number of in-domain training examples are available, and train and evaluate in a few-shot setting.

## Acknowledgments

The first author is supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1650441. We thank the Google Research Language and Cornell NLP groups for their comments and feedback during the project's development. We also thank Jing Li for pre-training the BERT$_{\text{LARGE+}}$ model, and Philip Massey, Zuyao Li, Angelica Chen, Karl Pichotta, and Francesco Piccinno for their contributions to the codebase. Finally, we thank the anonymous reviewers for their comments and suggestions.

## References

Yoav Artzi and Luke Zettlemoyer. 2011. Bootstrapping semantic parsers from conversations. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 421–432, Edinburgh, Scotland, UK. Association for Computational Linguistics.

Yoav Artzi and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1:49–62.

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*, pages 178–186, Sofia, Bulgaria. Association for Computational Linguistics.

Ben Bogin, Jonathan Berant, and Matt Gardner. 2019a. Representing schema structure with graph neural networks for text-to-SQL parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4560–4565, Florence, Italy. Association for Computational Linguistics.

Ben Bogin, Matt Gardner, and Jonathan Berant. 2019b. Global reasoning over database structures for text-to-SQL parsing. 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 3650–3655, Hong Kong, China. Association for Computational Linguistics.

David L. Chen and Raymond J. Mooney. 2011. Learning to interpret natural language navigation instructions from observations. pages 859–865.

DongHyun Choi, Myeong Cheol Shin, EungGyun Kim, and Dong Ryeol Shin. 2020. RYANSQL: Recursively applying sketch-based slot fillings for complex text-to-sql in cross-domain databases.

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*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

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 (Volume 1: Long Papers)*, pages 33–43, Berlin, Germany. Association for Computational Linguistics.

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

Ofer Givoli and Roi Reichart. 2019. Zero-shot semantic parsing for instructions. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4454–4464, Florence, Italy. Association for Computational Linguistics.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1631–1640, Berlin, Germany. Association for Computational Linguistics.

Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. 2016. Pointing the unknown words. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 140–149, Berlin, Germany. Association for Computational Linguistics.

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.

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

Jonathan Herzig and Jonathan Berant. 2018. Decoupling structure and lexicon for zero-shot semantic parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1619–1629, Brussels, Belgium. Association for Computational Linguistics.

Jonathan Herzig and Jonathan Berant. 2019. Don't paraphrase, detect! rapid and effective data collection for semantic parsing. 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 3801–3811, Hong Kong, China. Association for Computational Linguistics.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 963–973, Vancouver, Canada. Association for Computational Linguistics.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2018. Mapping language to code in programmatic context. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1643–1652, Brussels, Belgium. Association for Computational Linguistics.

Mohit Iyyer, Wen-tau Yih, and Ming-Wei Chang. 2017. Search-based neural structured learning for sequential question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1821–1831, Vancouver, Canada. Association for Computational Linguistics.

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)*, pages 12–22, Berlin, Germany. Association for Computational Linguistics.

Amol Kelkar, Rohan Relan, Vaishali Bhardwaj, Saurabh Vaichal, and Peter Relan. 2020. Bertrand-DR: Improving text-to-SQL using a discriminative re-ranker.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations*.

Jayant Krishnamurthy, Pradeep Dasigi, and Matt Gardner. 2017. Neural semantic parsing with type constraints for semi-structured tables. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1516–1526, Copenhagen, Denmark. Association for Computational Linguistics.

Igor Labutov, Bishan Yang, and Tom Mitchell. 2018. Learning to learn semantic parsers from natural language supervision. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1676–1690, Brussels, Belgium. Association for Computational Linguistics.

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

Kevin Lin, Ben Bogin, Mark Neumann, Jonathan Berant, and Matt Gardner. 2019. Grammar-based neural text-to-SQL generation. *arXiv preprint arXiv:1905.13326*.

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 *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Wang Ling, Phil Blunsom, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiský, Fumin Wang, and Andrew Senior. 2016. Latent predictor networks for code generation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 599–609, Berlin, Germany. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Reginald Long, Panupong Pasupat, and Percy Liang. 2016. Simpler context-dependent logical forms via model projections. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1456–1465, Berlin, Germany. Association for Computational Linguistics.

Scott Miller, David Stallard, Robert Bobrow, and Richard Schwartz. 1996. A fully statistical approach to natural language interfaces. In *34th Annual Meeting of the Association for Computational Linguistics*, pages 55–61, Santa Cruz, California, USA. Association for Computational Linguistics.

Qingkai Min, Yuefeng Shi, and Yue Zhang. 2019. A pilot study for Chinese SQL semantic parsing. 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 3643–3649, Hong Kong, China. Association for Computational Linguistics.

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. In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 574–584. IEEE.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the*

*7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1470–1480, Beijing, China. Association for Computational Linguistics.

Hoifung Poon. 2013. Grounded unsupervised semantic parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 933–943, Sofia, Bulgaria. Association for Computational Linguistics.

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 *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, pages 141–147, Geneva, Switzerland. COLING.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

Peter Shaw, Philip Massey, Angelica Chen, Francesco Piccinno, and Yasemin Altun. 2019. Generating logical forms from graph representations of text and entities. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 95–106, Florence, Italy. Association for Computational Linguistics.

Alane Suhr, Srinivasan Iyer, and Yoav Artzi. 2018. Learning to map context-dependent sentences to executable formal queries. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2238–2249, New Orleans, Louisiana. Association for Computational Linguistics.

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

Jesse Thomason, Shiqi Zhang, Raymond Mooney, and Peter Stone. 2015. Learning to interpret natural language commands through human-robot dialog. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

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

Yuk Wah Wong and Raymond Mooney. 2006. Learning for semantic parsing with statistical machine translation. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 439–446, New York City, USA. Association for Computational Linguistics.

Chunyang Xiao, Marc Dymetman, and Claire Gardent. 2016. Sequence-based structured prediction for semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1341–1350, Berlin, Germany. Association for Computational Linguistics.

Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, , and Thomas Dillig. 2017. SQLizer: Query synthesis from natural language. In *International Conference on Object-Oriented Programming, Systems, Languages, and Applications, ACM*, pages 63:1–63:26.

Ziyu Yao, Yu Su, Huan Sun, and Wen-tau Yih. 2019. Model-based interactive semantic parsing: A unified framework and a text-to-SQL case study. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5450–5461, Hong Kong, China. Association for Computational Linguistics.

Semih Yavuz, Izzeddin Gur, Yu Su, and Xifeng Yan. 2018. What it takes to achieve 100% condition accuracy on WikiSQL. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1702–1711, Brussels, Belgium. Association for Computational Linguistics.

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 *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, pages 476–486. IEEE.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450, Vancouver, Canada. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, Youxuan Jiang, Michihiro Yasunaga, Sungrok Shim, Tao Chen, Alexander Fabbri, Zifan Li, Luyao Chen, Yuwen Zhang, Shreya Dixit, Vincent Zhang, Caiming Xiong, Richard Socher, Walter Lasecki, and Dragomir Radev. 2019a. CoSQL: A conversational text-to-SQL challenge towards cross-domain natural language interfaces to databases. 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 1962–1979, Hong Kong, China. Association for Computational Linguistics.

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

Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Heyang Er, Irene Li, Bo Pang, Tao Chen, Emily Ji, Shreya Dixit, David Proctor, Sungrok Shim, Jonathan Kraft, Vincent Zhang, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019b. SParC: Cross-domain semantic parsing in context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4511–4523, Florence, Italy. 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 Thirteenth National Conference on Artificial Intelligence - Volume 2*, pages 1050–1055.

Luke Zettlemoyer and Michael Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 678–687, Prague, Czech Republic. Association for Computational Linguistics.

Luke Zettlemoyer and Michael Collins. 2009. Learning context-dependent mappings from sentences to logical form. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 976–984, Suntec, Singapore. Association for Computational Linguistics.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, UAI'05, pages 658–666, Arlington, Virginia, United States. AUAI Press.

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 5341–5352, Hong Kong, China. Association for Computational Linguistics.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.

## A Data Details

**Measuring Exact Column Match in Utterances** For each example, we identify columns used for direct comparison with values in the correct SQL query (not considering columns used to link two tables, order or group results, or in the top-level `SELECT` statement). We then heuristically identify whether any of the used column names appear in the utterance by canonicalizing the column name (e.g., replacing underscores with spaces) and performing a basic substring match. Because slight variants of column names may appear in the utterance, our reported results show an lower bound.

**Heuristically Filtering Datasets** We use several heuristics to filter evaluation data. Although we cannot automatically filter out all examples where database conventions are required to select the correct final column, we found that a good heuristic is filtering out examples that require selecting more than one final column. For example, in Advising, when an utterance asks for a list of classes, the labeled query always selects four columns from the `course` table: `department`, `name`, `number`, and `semester`. We remove all examples where a numerical or text value is not copiable from the input utterance, except for the numbers `0` and `1`, which are often not copied from the input (for example, limiting table results in `LIMIT 1`). We also remove all examples that result in an empty table, and examples where the gold query returns a count, and the resulting table is `[0]`.

## B Experimental Details

To choose model and learning hyperparameters, we began with the hyperparameters of Shaw et al. (2019), and performed a small number of experiments to improve performance on Spider.

**Model** For our encoder, we use a pre-trained BERT model (Devlin et al., 2019). All input tokens use the same segment ID. We use absolute positional embeddings. The word embedding size for all tokens is 128. We use wordpiece tokenization for the input utterance. To tokenize column and table names, we replace underscores with spaces, and then apply wordpiece tokenization.

The outputs of the encoder are transformed using a linear layer before being used by the decoder. We use a two-layer Transformer decoder (Vaswani et al., 2017) with eight attention heads. We use a gated copying mechanism, supervising gating decisions during training.

**Learning** During training, we use a maximum input size of 512 tokens. During training and inference, we use a maximum decoding length of 100. For training, we create seven examples per original Spider training example, with randomized permutations of table ordering, and column ordering within the table spans. We use a batch size of 32 and train for 30,000 update steps. We apply teacher-forcing during training. During training, we apply a dropout at a rate of 0.3 on the embeddings of decoder tokens and within the decoder Transformer. We use Adam optimizer (Kingma and Ba, 2014). We increase learning rate linearly from 0 to 0.00008 until 5,625 steps, then decrease it linearly to 0.0 by the end of training. The encoder begins as a pre-trained BERT model, whose parameters we freeze for the first 2,100 updates.

**Inference** For each evaluation example, we perform beam search against our trained model with a beam size of 100. Among the 10 most probable predictions from the beam search, we choose the highest-probability prediction that is a syntactically valid SQL query. To check syntactic validity, we test each prediction's execution against an emptied copy of the database (emptied to ensure we are not using database contents) and pass over queries which are inexecutable. We test the top 10 items in the beam to limit evaluation time, and find that in nearly all cases, if a syntactically correct prediction exists in the beam, it appears as one of the top 10 items. To correctly resolve predicted SQL$^{\text{UF}}$ queries, we use gold-standard foreign keys for all target databases.

Some queries are highly inefficient, and can take minutes to execute due to the sizes of the databases. To reduce the execution time of gold queries, we add database indices where possible. Even with these indices, some predicted queries still take a long time to execute. To make evaluation tractable, we use a timeout of 45 seconds per predicted query. If the query has not executed after 45 seconds, we return the empty table. Figure 3 shows the influence of the timeout threshold on the model performance for our best model on each evaluation dataset. By 45 seconds, the vast majority of predictions can execute, and execution accuracy has stabilized.

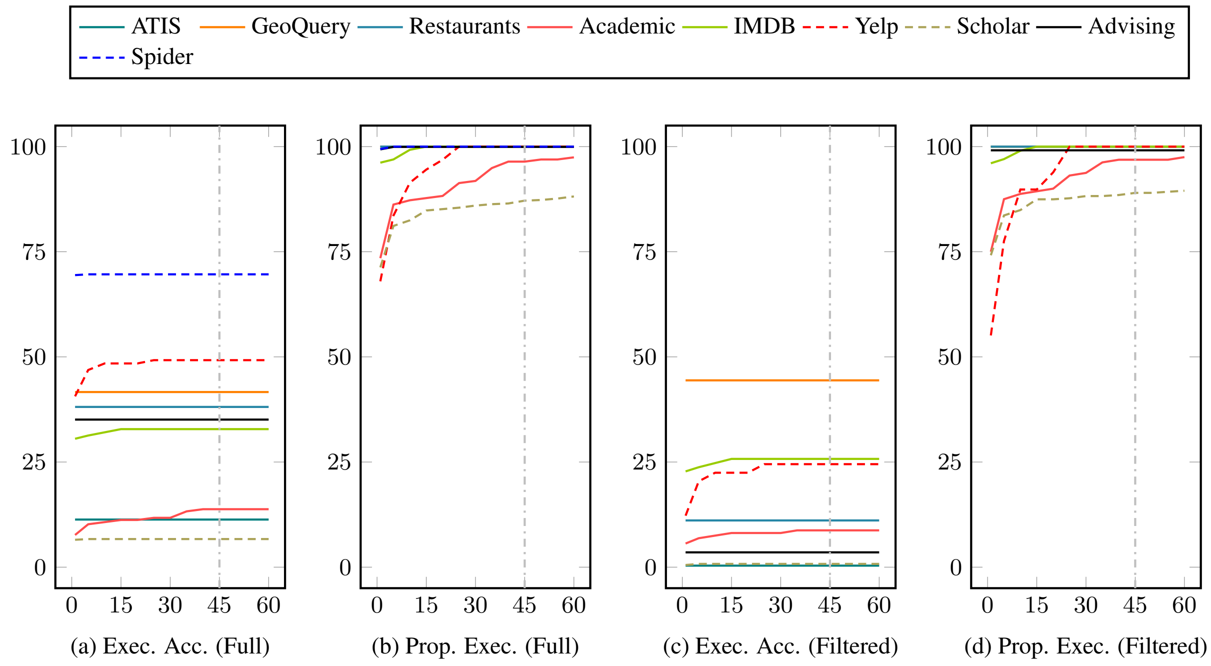

Figure 3: Influence of the timeout threshold on model performance for each dataset evaluation dataset. The $x$-axis shows the maximum execution time (in seconds) we allow before terminating query execution. In our experiments, we use a timeout of 45 seconds. We show how the execution accuracy is influenced by the cutoff time for the full (a) and filtered (c) evaluation sets. We also show the proportion of queries which finished execution for the full (b) and filtered (d) evaluation sets. Results are generated using the model which achieved the highest execution accuracy on the Spider development set. By 45 seconds, the majority of queries return an execution result, and execution accuracy has stabilized.

| Dataset | % of Gold Queries Covered by SQL$^{\text{UF}}$ |
|---|---|
| ATIS | 17.3 |
| GeoQuery | 97.5 |
| Restaurants | 100.0 |
| Academic | 85.7 |
| IMDB | 94.7 |
| Yelp | 81.3 |
| Scholar | 92.2 |
| Advising | 87.4 |
| Spider | 97.4 |

Table 4: Estimates of coverage of SQL$^{\text{UF}}$ across each evaluation dataset.

## C SQL$^{\text{UF}}$ Examples

Examples of our intermediate representation SQL$^{\text{UF}}$ are shown in Figure 4. Table 4 shows an estimate of the proportion of all gold queries that can be generated by our system.

## D Supplementary Error Analysis

Table 5 shows the rate of occurrence of eight types of errors in each evaluation dataset, out of twenty random incorrect predictions. Below, we give more detailed descriptions of these error types, as well as examples from our model predictions.

**Entity Understanding** We consider two types of errors: entity identification errors, and column matching errors. Entity identification errors include copying the wrong span of tokens that comprise an entity into the output. Column matching errors include comparing an entity to the wrong database column type. Examples for these errors are included in Figure 2.

**Final Column** We consider two types of final column errors: incorrect predictions, and ambiguous predictions. In incorrect predictions, the final column type is obviously incorrect with respect to the utterance. In ambiguous predictions, the final column is a reasonable prediction with respect to the utterance, but due to the dataset conventions, is incorrect. Examples for these errors are included in Figure 2.

**Database Understanding** We consider two types of database understanding errors: syntactically incorrect predictions, and predictions which do not compose the tables correctly. Although the second category execute successfully, their result-

| | |
|---|---|
| SQL: | SELECT people.name FROM people JOIN films ON people.id = film.person_id WHERE films.id = 5 |
| $\text{SQL}^{\text{UF}}$: | SELECT people.name **UF** WHERE films.id = 5 |
| SQL: | SELECT people.name FROM people JOIN films ON people.id = film.person_id |
| $\text{SQL}^{\text{UF}}$: | SELECT people.name **UF** films |
| SQL: | SELECT cities.state, count(*) FROM cities GROUP BY cities.state |
| $\text{SQL}^{\text{UF}}$: | SELECT cities.state, count(*) **UF** GROUP BY cities.state |
| SQL: | SELECT count(*) FROM cities |
| $\text{SQL}^{\text{UF}}$: | SELECT count(*) **UF** cities |
| SQL: | SELECT student_id FROM student_course_registrations UNION SELECT student_id FROM student_course_attendance |
| $\text{SQL}^{\text{UF}}$: | SELECT student_course_registrations.student_id **UF** UNION SELECT student_course_attendance.student_id **UF** |
| SQL: | SELECT table_1.id, table_3.id FROM table_1 JOIN table_2 ON table_1.table_2_id = table_2.id JOIN table_3 ON table_2.table_3_id |
| $\text{SQL}^{\text{UF}}$: | SELECT table_1.id, table_3.id **UF** table_2 |

Figure 4: Examples of $\text{SQL}^{\text{UF}}$, which uses under-specified `FROM` clauses. Tables are omitted from the `FROM` clause unless a column belonging to the given table is not mentioned elsewhere in the query. Under certain assumptions, reconstruction of the original SQL is possible given schema information.

ing tables are incorrect due to how the database is structured. For example, in Restaurants, although the `restaurant` table has a `city_name` column, the correct way to construct a query corresponding to an utterance like *how many places for french food are there in palo alto?* is to traverse the `location` table instead.

**Query Implementation** We consider two types of errors related to incorrectly implementing the utterance's intent in SQL: missing and incorrect constraints. Missing constraints involve ignoring a constraint mentioned in the utterance, such as the paper title constraint in the Academic example in Figure 2. Incorrect constraints are incorrect for reasons besides those described above, such as entity-column matching. For example (from GeoQuery):

| | |
|---|---|
| **NL**: | *what is the smallest state in the usa* |
| **Pred.**: | select state.state_name from state where state.country_name = 'usa' order by state.population limit 1; |
| **Gold**: | select T1.state_name from state as T1 where T1.area = (select min(T2.area) from state as T2); |

Our model's prediction incorrectly orders by population rather than the state's area.

## E Additional Results on Spider

Table 6 shows the performance on the Spider (Yu et al., 2018) development set for our single best model split by hardness level. Table 7 shows the F1 over query components of our best model on the Spider development set.

| Error Type | Spider | Restaurants | IMDB | ATIS | Academic | Scholar | Yelp | GeoQuery | Advising |
|---|---|---|---|---|---|---|---|---|---|
| Entity understanding | | | | | | | | | |
| *Identification* | 4 | 9 | 3 | 2 | 5 | 9 | 5 | 5 | 8 |
| *Column match* | 6 | 9 | 6 | 20 | 4 | 5 | 8 | 4 | 8 |
| Final column | | | | | | | | | |
| *Incorrect* | 5 | 0 | 3 | 3 | 8 | 4 | 1 | 5 | 8 |
| *Ambiguous* | 0 | 0 | 0 | 16 | 3 | 9 | 11 | 0 | 4 |
| Database understanding | | | | | | | | | |
| *Syntax error* | 0 | 2 | 15 | 0 | 0 | 1 | 1 | 0 | 1 |
| *Table composition* | 6 | 6 | 0 | 1 | 2 | 0 | 1 | 2 | 1 |
| Query implementation | | | | | | | | | |
| *Missing constraint* | 2 | 6 | 2 | 9 | 10 | 4 | 4 | 3 | 12 |
| *Incorrect constraint* | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 0 |

Table 5: For each evaluation dataset, we analyzed twenty random incorrect predictions of our best model. We categorized each into one or more error categories, including errors of understanding entities mentioned in the utterance, generating the correct top-level column selection, understanding the database structure, and correctly implementing the constraints of the utterance in SQL. We report the number of predictions, out of the twenty analyzed per dataset, which had each error type.

| System | Easy | Medium | Hard | Extra Hard |
|---|---|---|---|---|
| Ours with BERT$_{\text{LARGE}+}$ | 83.2 | 71.1 | 57.5 | 34.7 |

Table 6: Spider's official evaluation metric results for our best model split by hardness level on the Spider development set.

| Component | F1 |
|---|---|
| `SELECT` | 89.2 |
| `SELECT` (no `AGG`) | 90.4 |
| `WHERE` | 71.7 |
| `WHERE` (no `OP`) | 76.3 |
| `GROUP` (no `HAVING`) | 82.0 |
| `GROUP` | 79.4 |
| `ORDER` | 82.9 |
| `AND/OR` | 98.1 |
| `IEUN` | 46.5 |
| `KEYWORDS` | 89.5 |

Table 7: Per-component F1 of our best model on the Spider development set.