Weakly Supervised Text-to-SQL Parsing through Question Decomposition

Anonymous ACL submission

Abstract

Text-to-SQL parsers are crucial in enabling non-experts to effortlessly query relational data. Training such parsers, by contrast, generally requires expert annotation of natural language (NL) utterances paired with corresponding SQL queries. In this work, we propose a *weak supervision* approach for training text-to-SQL parsers. We take advantage of the recently proposed question meaning representation called QDMR, an intermediate between NL and formal query languages. We show that given questions, their QDMR structures (annotated by non-experts or automatically predicted), and the answers, we can automatically synthesize SQL queries that are then used to train text-to-SQL models. Extensive experiments test our approach on five benchmark datasets. The results show that our models perform competitively with those trained on annotated NL-SQL data. Overall, we effectively train text-to-SQL parsers, using zero SQL annotations.

1 Introduction

The development of natural language interfaces to databases has been extensively studied in recent years (Affolter et al., 2019; Kim et al., 2020; Thorne et al., 2021). The current standard is Machine Learning (ML) models which map utterances in natural language (NL) to executable SQL queries (Wang et al., 2020; Rubin and Berant, 2021). These models rely on supervised training examples of NL questions labeled with their corresponding SQL queries. Labeling copious amounts of data is cost-prohibitive as it requires experts that are familiar both with SQL and with the underlying database structure (Yu et al., 2018). Furthermore, it is often difficult to re-use existing training data in one domain in order to generalize to new ones (Suhr et al., 2020). Adapting the model to a new domain requires new NL-SQL training examples, which results in yet another costly round of annotation.

In this paper we propose a *weak supervision* approach for training text-to-SQL parsers. We avoid the use of manually labeled NL-SQL examples and rely instead on data provided by non-expert users. Fig. 1 presents a high-level view of our approach. The input (left corner, in red) is used to automatically synthesize SQL queries (step 3, in green) which, in turn, are used to train an NL-to-SQL model. The supervision signal consists of the question's answer and uniquely, a structured representation of the *question decomposition*, called QDMR. The annotation of both these supervision sources can be effectively crowdsourced to non-experts (Berant et al., 2013; Pasupat and Liang, 2015; Wolfson et al., 2020). In a nutshell, QDMR is a series of computational steps, expressed by semi-structured utterances, that together match the semantics of the original question. The bottom left corner of Fig. 1 shows an example QDMR of the question *"Which authors have more than 10 papers in the PVLDB journal?"*. The question is broken into five steps, where each step expresses a single logical operation (e.g., select papers, filter those in PVLDB) and may refer to previous steps. As QDMR is derived entirely from its question, it is agnostic to the underlying form of knowledge representation and has been used for questions on images, text and databases (Subramanian et al., 2020; Geva et al., 2021; Saparina and Osokin, 2021). In our work, we use QDMR as an intermediate representation for SQL synthesis. Namely, given an input QDMR we implement an automatic procedure mapping it to SQL. The QDMR can either be manually annotated or effectively predicted by a trained model, as shown in our experiments.

We continue to describe the main components of our system, using the aforementioned supervision (Fig. 1). The *SQL Synthesis* component (step 1) attempts to convert the input QDMR into a corresponding SQL query. To this end, *Phrase DB linking* matches phrases in the QDMR with rele-

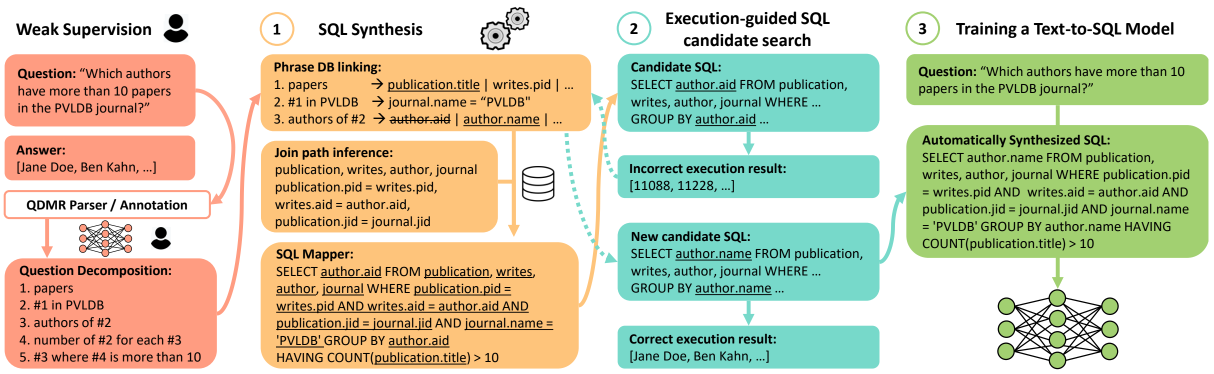

Figure 1: Our pipeline for training a Text-to-SQL model on data synthesized using weak supervision.

vant columns and values in the database. Next, SQL *join paths* are automatically inferred given the database schema structure. Last, the QDMR, DB-linked columns and inferred join paths are converted to SQL by the *SQL Mapper*. In step 2, we rely on question-answer supervision to filter out incorrect candidate SQL. Thus, our *Execution-guided SQL Search* returns the first candidate query which executes to the correct answer.

Given our synthesis procedure, we evaluate its ability to produce accurate SQL, using weak supervision. To this end, we run our synthesis on 9,313 examples of questions, answers and QDMRs from five standard NL-to-SQL benchmarks (Zelle and Mooney, 1996; Li and Jagadish, 2014; Yaghmazadeh et al., 2017; Yu et al., 2018). Overall, our solution successfully synthesizes SQL queries for 77.8% of examples, thereby demonstrating its applicability to a broad range of target databases.

Next, we show our synthesized queries to be an effective alternative to training on expert annotated SQL. We compare a text-to-SQL model, trained on the queries synthesized from questions, answers and QDMRs, to one trained using gold SQL. As our model of choice we use T5-large, which is widely used for sequence-to-sequence modeling tasks (Raffel et al., 2020). Following past work (Shaw et al., 2021; Herzig et al., 2021), we fine-tune T5 to map text to SQL. We experiment with the SPIDER and GEO880 datasets (Yu et al., 2018; Zelle and Mooney, 1996) and compare model performance based on the training supervision. When training on manually *annotated QDMRs*, the weakly supervised models achieve 91% to 97% of the accuracy of models trained on gold SQL. We further extend our approach to use automatically *predicted QDMRs*, requiring zero annotation of in-domain QDMRs. Notably, when training on predicted QDMRs models still reach 86% to 93% of the fully supervised versions accuracy. In addition, we evaluate cross-database generalization of models trained on SPIDER (Suhr et al., 2020). We test our models on four additional datasets and show that the weakly supervised models are generally better than the fully supervised ones in terms of cross-database generalization. Overall, our findings show that weak supervision, in the form of question, answers and QDMRs (annotated or predicted) is nearly as effective as gold SQL when training text-to-SQL parsers.

Our codebase and data are publicly available.[1]

2 Background

Weakly Supervised ML The performance of supervised ML models hinges on the quantity and quality of their training data. In practice, labeling large-scale datasets for new tasks is often cost-prohibitive. This problem is further exacerbated in semantic parsing tasks (Zettlemoyer and Collins, 2005), as utterances need to be labeled with formal queries. *Weak supervision* is a broad class of methods aimed at reducing the need to manually label large training sets (Hoffmann et al., 2011; Ratner et al., 2017; Zhang et al., 2019). An influential line of work has been dedicated to weakly supervised semantic parsing, using question-answer pairs, referred to as *learning from denotations* (Clarke et al., 2010; Liang et al., 2011). Past work has shown that non-experts are capable of annotating answers over knowledge graphs (Berant et al., 2013) and tabular data (Pasupat and Liang, 2015). This approach could potentially be extended to databases by sampling subsets of its tables, such that question-answer examples can be manually annotated. A key issue in learning text-to-SQL parsers from denotations is the vast search space of potential candidate queries. Therefore, past work has focused on con-

[1] `https://anonymized`

straining the search space, which limited applicability to simpler questions over single tables (Wang et al., 2019). Here, we handle arbitrary SQL by using QDMR to constrain the search space.

Question Decomposition QDMR expresses the meaning of a question by breaking it down into simpler sub-questions. Given a question x, its QDMR s is a sequence of reasoning steps $s^1, ..., s^{|s|}$ required to answer x. Each step s^k is an intermediate question which represents a relational operation, such as projection or aggregation. Steps may contain phrases from x, tokens signifying a query operation (e.g., *"for each"*) and references to previous steps. Operation tokens indicate the structure of a step, while its arguments are the references and question phrases. A key advantage of QDMR is that it can be annotated by non-experts and at scale (Wolfson et al., 2020). Moreover, unlike SQL, annotating QDMR requires zero domain expertise as it is derived entirely from the original question.

3 Weakly Supervised SQL Synthesis

Our input data contains examples of question x_i, database D_i, the answer a_i, and s_i, which is the QDMR of x_i. The QDMR is either annotated or predicted by a trained model f, such that $f(x_i) = s_i$. For each example, we attempt to synthesize a SQL query $\hat{Q}_i$, that matches the intent of x_i and executes to its answer, $\hat{Q}_i(D_i) = a_i$. The successfully synthesized examples $\langle x_i, \hat{Q}_i \rangle$ are then used to train an NL-to-SQL model.

3.1 Synthesizing SQL from QDMR

Given QDMR s_i and database D_i, we automatically map s_i into SQL. Alg. 1 describes the synthesis process, where candidate query $\hat{Q}_i$ is incrementally synthesized by iterating over the QDMR steps. Given step s_i^k, its phrases are automatically linked to columns and values in D_i. Then, relevant join paths are inferred between the columns. Last, each step is automatically mapped to SQL based on its QDMR operator and its arguments (see Table 1).

Algorithm 1 SQL Synthesis

procedure SQLSYNTH($\mathbf{s}$: QDMR, D: database)
  mapped $\leftarrow$ []
  for s^k in $\mathbf{s} = \langle s^1, ..., s^n \rangle$ **do**
    cols $\leftarrow$ PHRASECOLUMNLINK(D, s^k)
    refs $\leftarrow$ REFERENCEDSTEPS(s^k)
    join $\leftarrow$ []
    for s^j in *refs* **do**
      other_cols $\leftarrow$ *synth*$[j].cols$
      join $\leftarrow$ *join* + JOINP(D, *cols*, *other_cols*)
    op $\leftarrow$ OPTYPE(s^k)
    $\hat{Q}$ $\leftarrow$ MAPSQL(*op, cols, join, refs, mapped*)
    mapped$[k] \leftarrow \langle s^k, cols, \hat{Q} \rangle$
  return *mapped*$[n].\hat{Q}$

3.1.1 Phrase DB Linking

As discussed in §2, a QDMR step may have a phrase from x_i as its argument. When mapping QDMR to SQL these phrases are linked to corresponding columns or values in D_i. For example, in Table 1 the phrases *"ships"* and *"injuries"* are linked to columns `ship.id` and `death.injured` respectively. We perform phrase-column linking automatically by ranking all columns in D_i and returning the top one. The ranked list of columns is later used in §3.2 when searching for a correct assignment to all phrases in the QDMR. To compute phrase-column similarity, we tokenize both the phrase and column, then lemmatize their tokens using the WordNet lemmatizer.[2] The similarity score is the average GloVe word embeddings similarity (Pennington et al., 2014) between the phrase and column tokens. All columns in D_i are then ranked based on their word overlap and similarity with the phrase: (1) we return columns whose lemmatized tokens are identical to those in the phrase; (2) we return columns who share (non stop-word) tokens with the phrase, ordered by phrase-column similarity; (3) we return the remaining columns, ordered by similarity.

We assume that literal values in D_i, such as strings or dates, appear verbatim in the database as they do in the question. Therefore, using string matching, we can identify the columns containing all literal values mentioned in s_i. If a literal value appears in multiple columns, they are all returned as potential links. This assumption may not always hold in practice due to DB-specific language, e.g., the phrase *"women"* may correspond to the condition `gender = 'F'`. Consequently, we measure the effect of DB-specific language in §4.2.

3.1.2 Join Path Inference

In order to synthesize SQL (Codd, 1970), we infer join paths between the linked columns returned in §3.1.1. Following past work (Guo et al., 2019; Suhr et al., 2020), we implement a heuristic returning the shortest join path connecting two sets of columns. To compute join paths, we convert D_i into a graph where the nodes are its tables and edges exist for every foreign-key constraint connecting two tables. The JOINP procedure in Alg. 1 joins the tables of

[2] https://www.nltk.org/

QDMR Step	Phrase-DB Linking	SQL
1. ships	1. `SELECT(ship.id)`	`SELECT ship.id FROM ship;`
2. injuries	2. `SELECT(death.injured)`	`SELECT death.injured FROM death;`
3. number of #2 for each #1	3. `GROUP(count, #2, #1)`	`SELECT COUNT(death.injured) FROM ship, death WHERE death.caused_by_ship_id = ship.id GROUP BY ship.id;`
4. #1 where #3 is highest	4. `SUPERLATIVE(max, #1, #3)`	`SELECT ship.id FROM ship, death WHERE death.caused_by_ship_id = ship.id GROUP BY ship.id ORDER BY COUNT(death.injured) DESC LIMIT 1;`
5. the name of #4	5. `PROJECT(ship.name, #4)`	`SELECT ship.name FROM ship, death WHERE death.caused_by_ship_id = ship.id AND ship.id IN (#4);`

Table 1: Mapping the QDMR of the question *"What is the ship name that caused most total injuries?"* to SQL.

x: **"What are the populations of states through which the Mississippi river runs?"**

s: the Mississippi river; states #1 runs through; the populations of #2

```
1. SELECT(river.river_name = 'Mississippi')
2. PROJECT(state.state_name, #1)
3. PROJECT(state.population, #2)
```

```
1. SELECT river.river_name FROM river WHERE
   river.river_name = 'Mississippi';
2. SELECT state.state_name FROM state, river
   WHERE river.traverse = state.state_name AND
   river.river_name IN (#1);
3. SELECT state.population FROM state, river
   WHERE river.traverse = state.state_name AND
   state.state_name IN (#2);
```

Figure 2: Previously mapped QDMR steps (with operations and arguments) used as nested SQL queries.

x: **"What papers were written by H. V. Jagadish and Yunyao Li?"**

s: papers; #1 by H. V. Jagadish; #2 by Yunyao Li

```
1. SELECT(publication.title)
2. FILTER(#1, author.name = 'H. V. Jagadish')
3. FILTER(#2, author.name = 'Yunyao Li')
```

```
1. SELECT publication.title FROM author, publication;
2. SELECT publication.title FROM author, publica-
   tion, writes WHERE publication.pid= writes.pid AND
   writes.aid = author.aid AND author.name = 'H. V. Ja-
   gadish';
3. SELECT publication.title FROM author, publica-
   tion, writes WHERE publication.pid = writes.pid AND
   writes.aid = author.aid AND author.name = 'Yunyao Li'
   AND publication.title IN (#2);
```

Figure 3: Handling Self-joins in QDMR mapping. The two `FILTER` steps have conflicting assignments to the same column and are identified as a self-join. This is resolved by using a nested query in the SQL of step 3.

columns mentioned in step s^k (*cols*) with those mentioned in the previous steps which s^k refers to (*other_cols*). If multiple shortest paths exist, it returns the first path which contains either $c_i \in cols$ as its start node or $c_j \in other\_cols$ as its end node. Step 3 of Table 1 underlines the join path between the `death` and `ship` tables.

3.1.3 QDMR to SQL Mapper

The MAPSQL procedure in Alg. 1 maps QDMR steps into executable SQL. First, the QDMR operation of each step is inferred from its utterance template using the OPTYPE procedure of Wolfson et al. (2020). Then, following the previous DB linking phase, the arguments of each step are either the linked columns and values or references to previous steps (second column of Table 1). MAPSQL uses the step operation type and arguments to automatically map s^k to SQL query $\hat{Q}^k$. Each operation has a unique mapping to SQL as shown in Table 2. The mapping is performed incrementally for each step, while using parts of the mapped SQL of previous steps (stored in the *mapped* array) when these are referenced. Furthermore, our mapping procedure is able to handle complex SQL that may involve nested queries (Fig. 2) and self-joins (Fig. 3).

3.2 Execution-guided SQL Candidate Search

At this point we have $\hat{Q}_i$, which is a potential SQL candidate. However, this candidate may be incorrect due to a wrong phrase-column linking or due to its original QDMR structure. To mitigate these issues, we search for accurate SQL candidates using the answer supervision.

Following phrase DB linking (§3.1.1), each phrase is assigned its top ranked column in D_i. However, this assignment may potentially be wrong. In step 1 of Fig. 1 the phrase *"authors"* is incorrectly linked to `author.aid` instead of `author.name`. Given our weak supervision, we do not have access to the gold phrase-column linking and rely instead on the gold answer a_i. Namely, we iterate over phrase-column assignments and synthesize their corresponding SQL. Once an assignment whose SQL executes to a_i has been found, we return it as our result. We iterate over assignments that cover only the top-k ranked columns for each phrase, shown to work very well in practice (§4.2).

Failing to find a correct candidate SQL may be due to QDMR structure rather than phrase-column linking. As s_i is derived entirely from the question it may fail to capture database-specific language. E.g., in the question *"How many students enrolled during the semester?"* the necessary aggregate operation may change depending on the database structure. If D_i has the column `course.num_enrolled`, the query should *sum* its entries for all courses in the semester. Conversely, if D_i has the column `course.student_id`, the corresponding

QDMR Operation	SQL Mapping
`SELECT(t.col)`	`SELECT t.col FROM t;`
`FILTER(#x, =, val)`	`SELECT #x[SELECT] FROM #x[FROM] WHERE #x[WHERE] AND t.col = val;`
`PROJECT(t.col, #x)`	`SELECT t.col FROM t, #x[FROM] WHERE Join(t, #x[FROM]) AND #x[SELECT] IN (#x);`
`AGGREGATE(count, #x)`	`SELECT COUNT(#x[SELECT]) FROM #x[FROM] WHERE #x[WHERE];`
`GROUP(avg, #x, #y)`	`SELECT AVG(#x[SELECT]) FROM #x[FROM], #y[FROM] WHERE Join(#x[FROM], #y[FROM]) AND #x[WHERE] AND #y[WHERE] GROUP BY #y[SELECT];`
`SUPERLATIVE(max, k, #x, #y)`	`SELECT #x[SELECT] FROM #x[FROM], #y[FROM] WHERE Join(#x[FROM], #y[FROM]) AND #x[WHERE] AND #y[WHERE] ORDER BY #y[SELECT] DESC k;`

Table 2: QDMR to SQL mapping rules for six operations. The full set of mapping rules, for all QDMR operations, is provided in the Appendix A. #x denotes a previously mapped SQL query while #x[CLAUSE] denotes its relevant SQL clause. For example, #x[FROM] returns all tables in the FROM clause of SQL query #x.

query would need to *count* the number of enrolled students. We account for these structural mismatches by implementing three additional search heuristics which modify the structure of a candidate $\hat{Q}_i$. If the candidate executes to the correct result following modification, it is returned by the search process. These modifications are described in detail in Appendix B. Namely, they include the addition of a `DISTINCT` clause, converting QDMR `FILTER` steps into `SUPERLATIVE` and switching between the `COUNT` and `SUM` operations.

4 Experiments

Our experiments target two main research questions. First, given access to weak supervision of question-answer pairs and QDMRs, we wish to measure the percentage of SQL queries that can be automatically synthesized. Therefore, in §4.2 we measure the SQL synthesis *coverage* using 9,313 examples taken from five benchmark datasets. Second, in §4.3 we use the synthesized SQL to train NL-to-SQL models and compare their performance to those trained on gold SQL annotations.

4.1 Setting

Datasets We evaluate both the SQL synthesis coverage and NL-to-SQL accuracy using five NL-to-SQL datasets (see Table 3). The first four datasets contain questions over a single database: ACADEMIC (Li and Jagadish, 2014) has questions over the Microsoft Academic Search database; GEO880 (Zelle and Mooney, 1996) concerns US geography; IMDB and YELP (Yaghmazadeh et al., 2017) contain complex questions on a film and restaurants database, respectively. The SPIDER dataset (Yu et al., 2018) measures *domain generalization* between databases, and therefore contains questions over 160 different databases. For QDMR data we use the BREAK dataset (Wolfson et al., 2020). The only exception is 259 questions of IMDB and YELP, outside of BREAK, which we (authors) annotate with corresponding QDMRs and release with our code. See Appendix C for license.

Training We fine-tune the T5-large sequence-to-sequence model (Raffel et al., 2020) for both NL-to-SQL and QDMR parsing (§4.2). Namely, for each task we fine-tune the pre-trained model on its specific data. For NL-to-SQL, we fine-tune on mapping utterances $x_i; cols(D_i)$ to SQL, where sequence $cols(D_i)$ is a serialization of all columns in database D_i, in an arbitrary order. In QDMR parsing, input questions are mapped to output QDMR strings. We use the T5 implementation by HuggingFace (Wolf et al., 2020) and train using the Adam optimizer (Kingma and Ba, 2014). Following fine-tuning on the dev sets, we adjust the batch size to 128 and the learning rate to $1e$-4 (after experimenting with $1e$-5, $1e$-4 and $1e$-3). All models were trained on an NVIDIA GeForce RTX 3090 GPU.

4.2 SQL Synthesis Coverage

Our first challenge is to measure our ability to synthesize accurate SQL using weak supervision. To evaluate SQL synthesis we define its *coverage* as the percentage of examples where it successfully produces SQL $\hat{Q}$ which executes to the correct answer. To ensure our synthesis procedure is domain independent, we test it on examples from five different datasets, spanning across 164 DBs (Table 3).

Annotated QDMRs The upper rows of Table 3 list the SQL synthesis coverage when using manually annotated QDMRs from BREAK. Overall, we evaluate on 9,313 QDMR annotated examples, reaching a coverage of 77.8%. Synthesis coverage for single-DB datasets tends to be slightly higher than that of SPIDER, which we attribute to its larger size and diversity. To further ensure the quality of synthesized SQL, we manually validate a random

Dataset	DB #	Examples	Synthesized	Coverage %
ACADEMIC	1	195	155	79.5
GEO880	1	877	736	83.9
IMDB	1	131	116	88.5
YELP	1	128	100	78.1
SPIDER dev	20	1,027	793	77.2
SPIDER train	140	6,955	5,349	76.9
Total:	**164**	**9,313**	**7,249**	**77.8**
SPIDER pred.	20	1,027	797	77.6

Table 3: SQL synthesis coverage scores across datasets.

Error	Description	%
Nonstandard QDMR	Annotated QDMR contains a step that does not follow any of the pre-specified operation templates	42
DB-specific language	Phrase entails an implicit condition, e.g., *"female workers"* → `emp.gender = 'F'`	23
Phrase-column link.	The correct phrase-column assignment falls outside of the top-k candidates (§3.2)	13
Gold SQL	An error due to an incorrectly labeled gold SQL	6

Table 4: SQL synthesis error analysis.

subset of 100 queries out of the 7,249 that were synthesized. Our analysis reveals 95% of the queries to be correct interpretations of their original question. In addition, we evaluate synthesis coverage on a subset of 8,887 examples whose SQL denotations are not the empty set. As SQL synthesis relies on answer supervision, discarding examples with empty denotations eliminates the false positives of spurious SQL which incidentally execute to an empty set. Overall, coverage on examples with non-empty denotations is nearly identical, at 77.6% (see Appendix D). We also perform an error analysis on a random set of 100 failed examples, presented in Table 4. SQL synthesis failures are mostly due to QDMR annotation errors or implicit database-specific conditions. E.g., in GEO880 the phrase *"major river"* should implicitly be mapped to the condition `river.length > 750`. As our SQL synthesis is domain-general, it does not memorize any domain-specific jargon or rules.

Predicted QDMRs While QDMR annotation can be crowdsourced to non-experts (Wolfson et al., 2020), moving to a new domain may incur annotating new in-domain examples. Our first step to address this issue is to evaluate the coverage of SQL synthesis on predicted QDMRs, for out-of-domain questions. As question domains in SPIDER dev differ from those in its training set, it serves as our initial testbed. We further explore this setting in §4.3.4. Our QDMR parser (§4.1) is fine-tuned on BREAK for 10 epochs and we select the model with highest exact string match (EM) on BREAK dev. Evaluating on the hidden test set reveals our model scores 42.3 normalized EM,[3] on par with the state-of-the-art on BREAK.[4] The predicted QDMRs, are then used in SQL synthesis together with examples $\langle x_i, a_i, D_i \rangle$. In Table 3, the last row shows that coverage on SPIDER dev is nearly identical to that of manually annotated QDMRs (77.6% to 77.2%).

4.3 Training NL to SQL Models

Next, we compare NL-to-SQL models trained on our synthesized data to training on expert annotated SQL. Given examples $\langle x_i, D_i \rangle$ we test the following settings: (1) A *fully supervised* training set, that uses gold SQL annotations $\{\langle x_i, Q_i, D_i \rangle\}_{i=1}^{n}$. (2) A *weakly supervised* training set, where given answer a_i and QDMR s_i, we synthesize queries $\hat{Q}_i$. As SQL synthesis coverage is less than 100%, the process returns a subset of $m < n$ examples $\{\langle x_i, \hat{Q}_i, D_i \rangle\}_{i=1}^{m}$ on which the model is trained.[5]

4.3.1 Training Data

We train models on two NL-to-SQL datasets: SPIDER (Yu et al., 2018) and GEO880 (Zelle and Mooney, 1996). As our weakly supervised training sets, we use the synthesized examples $\langle x_i, \hat{Q}_i, D_i \rangle$, described in §4.2, (using annotated QDMRs). We successfully synthesized 5,349 training examples for SPIDER and 547 examples for GEO880 train.

4.3.2 Models and Evaluation

Models We fine-tune T5-large for NL-to-SQL, using the hyperparameters from §4.1. We choose T5 as it is agnostic to the structure of its input sequences. Namely, it has been shown to perform competitively on different text-to-SQL datasets, regardless of their SQL conventions (Shaw et al., 2021; Herzig et al., 2021). This property is particularly desirable in our cross-database evaluation (§4.3.3), where we test on multiple datasets.

We train and evaluate the following models:

- **T5-SQL-G** trained on $\{\langle x_i, Q_i, D_i \rangle\}_{i=1}^{n}$, using *gold SQL*, annotated by experts
- **T5-QDMR-G** trained on $\{\langle x_i, \hat{Q}_i, D_i \rangle\}_{i=1}^{m}$ with $\hat{Q}_i$ that were synthesized using weak supervision
- **T5-SQL-G$_{\textbf{part}}$** trained on $\{\langle x_i, Q_i, D_i \rangle\}_{i=1}^{m}$, using gold SQL. This models helps us measure the degree to which the smaller size of the synthesized training data and its different query structure (compared to gold SQL) affects performance

[3] The metric is a strict lower bound on performance.

[4] https://leaderboard.allenai.org/break

[5] In practice, we do not train directly on $\hat{Q}_i$ but on s_i following its phrase-column linking. This representation is then automatically mapped to SQL to evaluate its execution.

Model	Supervision	Training set	Exec. %
T5-SQL-G	$\langle x_i, Q_i, D_i \rangle$	7,000	68.0 ± 0.3
T5-SQL-G_{part}	$\langle x_i, Q_i, D_i \rangle$	5,349	66.4 ± 0.8
T5-QDMR-G	$\langle x_i, a_i, s_i, D_i \rangle$	5,349	65.8 ± 0.3
T5-QDMR-P	$\langle x_i, a_i, D_i \rangle$*	5,075	62.9 ± 0.8

Table 5: SPIDER trained model results on the dev set. *Supervision for T5-QDMR-P also includes 700 annotated QDMRs of SPIDER train questions.

Model	ACADEMIC	GEO880	IMDB	YELP
T5-SQL-G	8.2 ± 1.3	33.6 ± 2.5	19.8 ± 3.6	22.7 ± 1.2
T5-SQL-G_{part}	4.9 ± 1.5	32.4 ± 1.3	20.9 ± 0.8	20.7 ± 1.4
T5-QDMR-G	10.7 ± 0.7	40.4 ± 1.8	27.1 ± 3.6	16.2 ± 4.7
T5-QDMR-P	8.2 ± 0.4	39.7 ± 1.4	23.6 ± 5.5	16.7 ± 3.7

Table 6: SPIDER trained models zero-shot performance on cross-database (XSP) examples.

Evaluation Metric Due to our SQL being automatically synthesized, its syntax is often different from that of the gold SQL (see Appendix E.2). As a result, the ESM metric of Yu et al. (2018) does not fit our evaluation setup. Instead, we follow Suhr et al. (2020) and evaluate NL-to-SQL models using the *execution accuracy* of output queries. We define execution accuracy as the percentage of output queries which, when executed on the database, result in the same set of tuples (rows) as a_i.

4.3.3 Training on Annotated QDMRs

We begin by comparing the models trained using annotated QDMRs to those trained on gold SQL. Meanwhile, the discussion of T5-QDMR-P, trained using predicted QDMRs, is left for §4.3.4. The results in Tables 5-7 list the average accuracy and standard deviation of three model instances, trained using separate random seeds.

SPIDER & XSP Evaluation Tables 5-6 list the results of the SPIDER trained models. All models were trained for 150 epochs and evaluated on the dev set of 1,034 examples. When comparing T5-QDMR-G to the model trained on gold SQL, it achieves 96.8% of its performance (65.8 to 68.0). The T5-SQL-G_{part} model, trained on the same 5,349 examples as T5-QDMR-G, performs roughly on par, scoring +0.6 points (66.4 to 65.8).

As SPIDER is used to train cross-database models, we further evaluate our models performance on *cross-database semantic parsing* (XSP) (Suhr et al., 2020). In Table 6 we test on four additional NL-to-SQL datasets (sizes in parenthesis): ACADEMIC (183), GEO880 (877), IMDB (113) and YELP (66). For ACADEMIC, IMDB and YELP we removed examples whose execution result in an empty set. Otherwise, the significant percentage of such examples would result in false positives of predictions which incidentally execute to an empty set. In practice, evaluation on the full datasets remains mostly unchanged and is provided in Appendix E. Similarly to Suhr et al. (2020), the results in Table 6 show that SPIDER trained models struggle to generalize to XSP examples. However, T5-QDMR-G performance is generally better on XSP examples, which further indicates that QDMR and answer supervision is effective, compared to gold SQL. Example predictions are shown in Appendix E.2.

Model	Supervision	Train. set	Exec. %
T5-SQL-G	$\langle x_i, Q_i, D_i \rangle$	547	82.1 ± 1.9
T5-SQL-G_{part}	$\langle x_i, Q_i, D_i \rangle$	454	79.4 ± 0.4
T5-QDMR-G	$\langle x_i, a_i, s_i, D_i \rangle$	454	74.5 ± 0.2
T5-QDMR-P	$\langle x_i, a_i, D_i \rangle$	432	70.4 ± 0.2

Table 7: GEO880 trained models results on the test set. Supervision for T5-QDMR-P does not include any in-domain annotated QDMRs.

GEO880 Table 7 lists the execution accuracy of models trained on GEO880. Models were trained for 300 epochs, fine-tuned on the dev set and then evaluated on the 280 test examples. We note that T5-QDMR-G achieves 90.7% of the performance of T5-SQL-G (74.5 to 82.1). The larger performance gap, compared to SPIDER models, may be partly to due to the dataset size. As GEO880 has 547 training examples, fewer synthesized SQL to train T5-QDMR-G on (454) may have had a greater effect on its accuracy.

4.3.4 Training on Predicted QDMRs

We extend our approach by replacing the annotated QDMRs with the predictions of a trained QDMR parser (a T5-large model, see §4.1). In this setting, we now have two sets of questions: (1) questions used to train the QDMR parser; (2) questions used to synthesize NL-SQL data. We want these two sets to be as separate as possible, so that training the QDMR parser would not require new *in-domain* annotations. Namely, the parser must generalize to questions in the NL-SQL domains while being trained on as few of these questions as possible.

SPIDER Unfortunately, SPIDER questions make up a large portion of the BREAK training set, used to train the QDMR parser. We therefore experiment with two alternatives to minimize the in-domain QDMR annotations of NL-SQL questions. First, is to train the parser using few-shot QDMR annotations for SPIDER. Second, is to split SPIDER to questions used as the NL-SQL data, while the rest are used to train the QDMR parser.

In Table 5, T5-QDMR-P is trained on 5,075 queries, synthesized using predicted QDMRs (and answer supervision) for SPIDER train questions. The predictions were generated by a QDMR parser trained on a subset of BREAK, excluding all SPIDER questions save 700 (10% of SPIDER train). Keeping few in-domain examples minimizes additional QDMR annotation while preserving the predictions quality. Training on the predicted QDMRs, instead of the annotated ones, resulted in accuracy being down 2.9 points (65.8 to 62.9) while the model achieves 92.5% of T5-SQL-G performance on SPIDER dev. On XSP examples, T5-QDMR-P is competitive with T5-QDMR-G (Table 6).

In Table 8, we experiment with training T5-QDMR-P without in-domain QDMR annotations. We avoid any overlap between the questions and domains used to train the QDMR parser and those used for SQL synthesis. We randomly sample 30-40 databases from SPIDER and use their corresponding questions exclusively as our NL-SQL data. For training the QDMR parser, we use BREAK while discarding the sampled questions. We experiment with 3 random samples of SPIDER train, numbering 1,348, 2,028 and 2,076 examples, with synthesized training data of 1,129, 1,440 and 1,552 examples respectively. Results in Table 8 show that, on average, T5-QDMR-P achieves 95.5% of the performance of T5-SQL-G. This indicates that even without any in-domain QDMR annotations, data induced from answer supervision and out-of-domain QDMRs is effective in training NL-to-SQL models, compared to gold SQL.

GEO880 For predicted QDMRs on GEO880, we train the QDMR parser on BREAK while discarding all of its 547 questions. Therefore, the parser was trained without any in-domain QDMR annotations for GEO880. SQL synthesis using the predicted QDMRs resulted in 432 queries. In Table 7, T5-QDMR-P reaches 85.7% of T5-SQL-G performance while being trained using question-answer supervision and no in-domain QDMR annotations.

Model	Supervision	Train. set	DB #	Exec. %
T5-SQL-G	$\langle x_i, Q_i, D_i \rangle$	1,348	30	48.4
T5-SQL-G$_{\text{part}}$	$\langle x_i, Q_i, D_i \rangle$	1,129	30	47.4
T5-QDMR-P	$\langle x_i, a_i, D_i \rangle$	1,129	30	46.2
T5-SQL-G	$\langle x_i, Q_i, D_i \rangle$	2,028	40	54.7
T5-SQL-G$_{\text{part}}$	$\langle x_i, Q_i, D_i \rangle$	1,440	40	51.3
T5-QDMR-P	$\langle x_i, a_i, D_i \rangle$	1,440	40	52.1
T5-SQL-G	$\langle x_i, Q_i, D_i \rangle$	2,076	40	56.2
T5-SQL-G$_{\text{part}}$	$\langle x_i, Q_i, D_i \rangle$	1,552	40	53.7
T5-QDMR-P	$\langle x_i, a_i, D_i \rangle$	1,552	40	53.8

Table 8: SPIDER models results on the dev set. T5-QDMR-P is trained without using any QDMR annotations for training set questions. We train separate models on the three randomly sampled training sets.

5 Related Work

For a thorough review of NL interfaces to databases see Affolter et al. (2019); Kim et al. (2020). Research on parsing text-to-SQL gained significant traction in recent years with the introduction of large supervised datasets for training models and evaluating their performance (Zhong et al., 2017; Yu et al., 2018). Recent approaches relied on specialized architectures combined with pre-trained language models (Guo et al., 2019; Wang et al., 2020; Lin et al., 2020; Yu et al., 2021; Deng et al., 2021; Scholak et al., 2021). As our solution synthesizes NL-SQL pairs (using weak supervision) it can be used to train supervised NL-to-SQL models.

Also related is the use of intermediate meaning representations (MRs) in mapping NL-to-SQL. In contrast to QDMR, past MRs were either annotated by experts (Yaghmazadeh et al., 2017; Kapanipathi et al., 2020), or were directly induced from such annotations as a way to simplify the target MR (Guo et al., 2019; Herzig et al., 2021). Similarly to us, Saparina and Osokin (2021) map QDMR to SPARQL. Contrastly, our SQL synthesis does not rely on the annotated linking of question phrases to DB elements (Lei et al., 2020). We further train models without gold QDMR annotations and test our models on four datasets in addition to SPIDER.

6 Conclusions

This work presents a weakly supervised approach for generating NL-SQL training data, using answer and QDMR supervision. We implemented an automatic SQL synthesis procedure, capable of generating effective training data for dozens of target databases. Experiments on multiple NL-to-SQL benchmarks demonstrate the efficacy of our synthesized training data. Namely, our weakly-supervised models achieve 91%-97% of the performance of fully supervised models trained on annotated SQL. Further constraining our models supervision to few or zero in-domain QDMRs still reaches 86%-93% of the fully supervised models performance. Overall, we provide an effective solution to train text-to-SQL parsers while requiring zero SQL annotations.

References

Katrin Affolter, Kurt Stockinger, and A. Bernstein. 2019. A comparative survey of recent natural language interfaces for databases. *The VLDB Journal*, 28:793 – 819.

J. Berant, A. Chou, R. Frostig, and P. Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Empirical Methods in Natural Language Processing (EMNLP)*.

J. Clarke, D. Goldwasser, M. Chang, and D. Roth. 2010. Driving semantic parsing from the world's response. In *Computational Natural Language Learning (CoNLL)*, pages 18–27.

Edgar F Codd. 1970. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387.

Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. Structure-grounded pretraining for text-to-sql. In *NAACL*.

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.

Mor Geva, Tomer Wolfson, and Jonathan Berant. 2021. Break, perturb, build: Automatic perturbation of reasoning paths through question decomposition. *ArXiv*, abs/2107.13935.

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 *Association for Computational Linguistics (ACL)*.

Jonathan Herzig, Peter Shaw, Ming-Wei Chang, Kelvin Guu, Panupong Pasupat, and Yuan Zhang. 2021. Unlocking compositional generalization in pre-trained models using intermediate representations. *ArXiv*, abs/2104.07478.

R. Hoffmann, C. Zhang, X. Ling, L. S. Zettlemoyer, and D. S. Weld. 2011. Knowledge-based weak supervision for information extraction of overlapping relations. In *Association for Computational Linguistics (ACL)*, pages 541–550.

Pavan Kapanipathi, I. Abdelaziz, Srinivas Ravishankar, S. Roukos, Alexander G. Gray, Ramón Fernández Astudillo, Maria Chang, Cristina Cornelio, S. Dana, Achille Fokoue, Dinesh Garg, A. Gliozzo, Sairam Gurajada, Hima Karanam, Naweed Khan, Dinesh Khandelwal, Youngsuk Lee, Yunyao Li, Francois Luus, Ndivhuwo Makondo, Nandana Mihindukulasooriya, Tahira Naseem, Sumit Neelam, L. Popa, Revanth Reddy, R. Riegel, G. Rossiello, Udit Sharma, G P Shrivatsa Bhargav, and M. Yu. 2020. Question answering over knowledge bases by leveraging semantic parsing and neuro-symbolic reasoning. *ArXiv*, abs/2012.01707.

Hyeonji Kim, Byeong-Hoon So, Wook-Shin Han, and Hongrae Lee. 2020. Natural language to sql: Where are we today? *Proc. VLDB Endow.*, 13:1737–1750.

D. Kingma and J. Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

Wenqiang Lei, Weixin Wang, Zhixin Ma, Tian Gan, Wei Lu, Min-Yen Kan, and Tat-Seng Chua. 2020. Re-examining the role of schema linking in text-to-sql. In *EMNLP*.

Fei Li and Hosagrahar Visvesvaraya Jagadish. 2014. Nalir: an interactive natural language interface for querying relational databases. In *International Conference on Management of Data, SIGMOD*.

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

Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2020. Bridging textual and tabular data for cross-domain text-to-SQL semantic parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4870–4888, Online. Association for Computational Linguistics.

P. Pasupat and P. Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Association for Computational Linguistics (ACL)*.

J. Pennington, R. Socher, and C. D. Manning. 2014. GloVe: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

A. Ratner, Stephen H. Bach, Henry R. Ehrenberg, Jason Alan Fries, Sen Wu, and C. Ré. 2017. Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*.

Ohad Rubin and Jonathan Berant. 2021. SmBoP: Semi-autoregressive bottom-up semantic parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 311–324, Online. Association for Computational Linguistics.

Irina Saparina and Anton Osokin. 2021. Sparqling database queries from intermediate question decompositions. In *EMNLP*.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2021. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? In *ACL/IJCNLP*.

Sanjay Subramanian, Ben Bogin, Nitish Gupta, Tomer Wolfson, Sameer Singh, Jonathan Berant, and Matt Gardner. 2020. Obtaining faithful interpretations from compositional neural networks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5594–5608, Online. Association for Computational Linguistics.

Alane Suhr, Ming-Wei Chang, Peter Shaw, and Kenton Lee. 2020. Exploring unexplored generalization challenges for cross-database semantic parsing. In *ACL*.

James Thorne, Majid Yazdani, Marzieh Saeidi, F. Silvestri, S. Riedel, and A. Halevy. 2021. From natural language processing to neural databases. *Proc. VLDB Endow.*, 14:1033–1039.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and M. Richardson. 2020. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *ArXiv*, abs/1911.04942.

Bailin Wang, Ivan Titov, and Mirella Lapata. 2019. Learning semantic parsers from denotations with latent structured alignments and abstract programs. In *EMNLP*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Tomer Wolfson, Mor Geva, Ankit Gupta, Matt Gardner, Yoav Goldberg, Daniel Deutch, and Jonathan Berant. 2020. Break it down: A question understanding benchmark. *Transactions of the Association for Computational Linguistics*, 8:183–198.

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

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir R. Radev, Richard Socher, and Caiming Xiong. 2021. Grappa: Grammar-augmented pre-training for table semantic parsing. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Empirical Methods in Natural Language Processing (EMNLP)*.

M. Zelle and R. J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pages 1050–1055.

L. S. Zettlemoyer and M. Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Uncertainty in Artificial Intelligence (UAI)*, pages 658–666.

Zhen-Yu Zhang, Peng Zhao, Yuan Jiang, and Z. Zhou. 2019. Learning from incomplete and inaccurate supervision. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

V. Zhong, C. Xiong, and R. Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

A QDMR to SQL Mapping Rules

Table 9 lists all of the QDMR operations along with their mapping rules to SQL. For a thorough description of QDMR semantics please refer to Wolfson et al. (2020).

B SQL Candidate Search Heuristics

We further describe the execution-guided search process for candidate SQL queries, that was introduced in §3.2. Given the search space of candidate queries, we use four heuristics to find candidates $\hat{Q}_i$ which execute to the correct answer, a_i.

1. Phrase linking search: We avoid iterating over each phrase-column assignment by ordering them according to their phrase-column ranking, as described in §3.1.1. The query $\hat{Q}_i^{(1)}$ is induced from the top ranked assignment, where each

QDMR Operation	SQL Mapping
`SELECT(t.col)`	`SELECT t.col FROM t;`
`SELECT(val)`	`SELECT t.col FROM t WHERE t.col = val;`
`FILTER(#x, =, val)`	`SELECT #x[SELECT] FROM #x[FROM] WHERE #x[WHERE] AND t.col = val;`
`PROJECT(t.col, #x)`	`SELECT t.col FROM t, #x[FROM] WHERE Join(t, #x[FROM]) AND #x[SELECT] IN (#x);`
`AGGREGATE(count, #x)`	`SELECT COUNT(#x[SELECT]) FROM #x[FROM] WHERE #x[WHERE];`
`GROUP(avg, #x, #y)`	`SELECT AVG(#x[SELECT]) FROM #x[FROM], #y[FROM] WHERE Join(#x[FROM], #y[FROM]) AND #x[WHERE] AND #y[WHERE] GROUP BY #y[SELECT];`
`SUPERLATIVE(max, k, #x, #y)`	`SELECT #x[SELECT] FROM #x[FROM], #y[FROM] WHERE Join(#x[FROM], #y[FROM]) AND #x[WHERE] AND #y[WHERE] ORDER BY #y[SELECT] DESC k;`
`COMPARATIVE(#x, #y, >, val)`	`SELECT #x[SELECT] FROM #x[FROM], #y[FROM] WHERE Join(#x[FROM], #y[FROM]) AND #x[WHERE] AND #y[WHERE] AND #y[SELECT] > val;`
`UNION(#x, #y)`	`SELECT #x[SELECT] FROM #x[FROM], #y[FROM] WHERE Join(#x[FROM], #y[FROM]) AND (#x[WHERE] OR #y[WHERE]);`
`UNION_COLUMN(#x, #y)`	`SELECT #x[SELECT], #y[SELECT] FROM #x[FROM], #y[FROM] WHERE Join(#x[FROM], #y[FROM]) AND #x[WHERE] AND #y[WHERE];`
`INTERSECT(t.col, #x, #y)`	`SELECT t.col FROM t, #x[FROM], #y[FROM] WHERE Join(t, #x[FROM], #y[FROM]) AND #x[WHERE] AND t.col IN ( SELECT t.col FROM t, #x[FROM], #y[FROM] WHERE Join(t, #x[FROM], #y[FROM]) AND #y[WHERE] );`
`SORT(#x, #y, asc)`	`SELECT #x[SELECT] FROM #x[FROM], #y[FROM] WHERE Join(#x[FROM], #y[FROM]) AND #x[WHERE] ORDER BY #y[SELECT] ASC;`
`DISCARD(#x, #y)`	`SELECT #x[SELECT] FROM #x[FROM] WHERE #x[WHERE] AND #x[SELECT] NOT IN ( #y );`
`ARITHMETIC(+, #x, #y)`	`( #x ) + ( #y );`

Table 9: QDMR to SQL mapping rules for all QDMR operations. #x denotes a previously mapped SQL query while #x[CLAUSE] denotes its relevant SQL clause. For example, #x[FROM] returns all tables in the FROM clause of SQL query #x. Join, denotes the inferred join paths between sets of tables (see §3.1.2). Note that `AGGREGATE` and `GROUP` steps may use the operations: min, max, count, sum and avg. `SUPERLATIVE` steps may use min, max operations and `COMPARATIVE` steps use the operations: $>$, $<$, $=$, $\neq$, $\geq$, $\leq$. Last, `SORT` steps sort in either ascending (asc) or descending (desc) order and `ARITHMETIC` steps use one of the following: $+$, $-$, $\times$, $\div$.

phrase in s_i is assigned its top ranked column. If $\hat{Q}_i^{(1)}(D_i) \neq a_i$ we continue the candidate search using heuristics 2-4 (described below). Assuming that the additional search heuristics failed to find a candidate $\hat{Q}_i^{(1)'}$ such that $\hat{Q}_i^{(1)'}(D_i) = a_i$, we return to the phrase linking component and resume the process using the candidate SQL induced from the following assignment $\hat{Q}_i^{(2)}$, and so forth. In practice, we limit the number of assignments and review only those covering the top-k most similar columns for each phrase in s_i, where $k = 20$. Our error analysis (Table 4) reveals that only a small fraction of failures are due to limiting k. Step 2 in Fig. 1 represents the iterative process, where $\hat{Q}_i^{(1)}$ executes to an incorrect result while the following candidate $\hat{Q}_i^{(2)}$ correctly links the phrase *"authors"* to column `author.name` and executes to a_i, thereby ending the search.

2. Distinct modification: Given a candidate SQL $\hat{Q}_i$ such that $\hat{Q}_i(D_i) \neq a$, we add `DISTINCT` to its `SELECT` clause. In Table 10 the SQL executes to the correct result, following its modification.

3. Superlative modification: This heuristic automatically corrects semantic mismatches between annotated QDMR structures and the underlying database. Concretely, steps in s_i that represent `PROJECT` and `FILTER` operations may entail an implicit `ARGMAX/ARGMIN` operation. For example for the question *"What is the size of the largest state in the USA?"* in the third row of Table 10. Step (3) of the question's annotated QDMR is the `PROJECT` operation, "state with the largest #2". While conforming to the `PROJECT` operation template, the step entails an `ARGMAX` operation. Using the NLTK part-of-speech tagger, we automatically identify any superlative tokens in the `PROJECT` and `FILTER` steps of s_i. These steps are then replaced with the appropriate `SUPERLATIVE` type steps. In Table 10, the original step (3) is modified to the step "#1 where #2 is highest".

4. Aggregate modification: This heuristics replaces instances of `COUNT` in QDMR steps with `SUM` operations, and vice-versa. In Table 10, the question *"Find the total student enrollment for different affiliation type schools."*, is incorrectly mapped to a candidate query involving a `COUNT` operation on `university.enrollment`. By modifying the aggregate operation to `SUM`, the new $\hat{Q}_i$ correctly executes to a_i and is therefore returned as the output.

Heuristic	Question	Candidate SQL/QDMR	Modified Candidate SQL/QDMR
Phrase linking search	What are the distinct majors that students with treasurer votes are studying?	`SELECT DISTINCT student.major FROM student, voting_record WHERE student.stuid =` **`voting_record.stuid`**	`SELECT DISTINCT student.major FROM student, voting_record WHERE student.stuid =` **`voting_record.treasurer_vote`**
Distinct modification	Find the number of different product types.	`SELECT products.product_type_code FROM products`	`SELECT` **`DISTINCT`** `products.product_type_code FROM products`
Superlative modification	What is the size of the largest state in the USA?	`(1) states in the usa; (2) size of #1;` **`(3) state with the largest #2;`** `(4) size of #3`	`(1) states in the usa; (2) size of #1;` **`(3) #1 where #2 is highest;`** `(4) the size of #3`
Aggregate modification	Find the total student enrollment for different affiliation type schools.	`SELECT university.affiliation,` **`COUNT`** `(university.enrollment) FROM university GROUP BY university.affiliation`	`SELECT university.affiliation,` **`SUM`** `(university.enrollment) FROM university GROUP BY university.affiliation`

Table 10: Examples of the four execution-guided search heuristics used during SQL synthesis.

C Data License

We list the license (when publicly available) and release details of the datasets used in our paper.

The BREAK dataset (Wolfson et al., 2020) is under the MIT License. SPIDER (Yu et al., 2018) is under the CC BY-SA 4.0 License. GEO880 (Zelle and Mooney, 1996) is available under the GNU General Public License 2.0.

The text-to-SQL versions of GEO880 and ACADEMIC (Li and Jagadish, 2014) were made publicly available by Finegan-Dollak et al. (2018) in: https://github.com/jkkummerfeld/text2sql-data/.

The IMDB and YELP datasets were publicly released by Yaghmazadeh et al. (2017) in: goo.gl/DbUBMM.

D SQL Synthesis Coverage

We provide additional results of SQL synthesis coverage. Table 11 lists the coverage results, per dataset, when discarding all examples whose SQL executes to an empty set. Out of the 9,313 original examples, 8,887 examples have non-empty denotations. Coverage scores per dataset remain generally the same as they do when evaluating on all examples. These results further indicate the effectiveness of the SQL synthesis procedure. Namely, this ensures the synthesis results in Table 3 are faithful, despite the potential noise introduced by SQL with empty denotations.

E NL to SQL Models Results

E.1 Evaluation on the Full XSP Datasets

We provide additional results of the models trained on SPIDER. Namely, we evaluate on all examples of the ACADEMIC, IMDB and YELP datasets, including examples whose denotations are empty. Table 12 lists the results of all the models trained on the original training set of SPIDER. In Table 13 we provide the XSP results of the models trained on the random subsets of SPIDER train, used in §4.3.4. Similar to our previous experiments, T5-QDMR-P is generally better than T5-SQL-G in terms of its cross-database generalization.

Dataset	DB #	Examples	Synthesized	Coverage %
ACADEMIC	1	183	148	80.9
GEO880	1	846	707	83.6
IMDB	1	113	101	89.4
YELP	1	66	54	81.8
SPIDER dev	20	978	745	76.2
SPIDER train	140	6,701	5,137	76.7
Total:	**164**	**8,887**	**6,892**	**77.6**
SPIDER pred.	20	978	750	76.7

Table 11: SQL synthesis coverage scores for SQL queries with non-empty denotations. We report the coverage only for non-empty examples to minimize the effect of potentially spurious SQL being synthesized.

E.2 Qualitative Results

Table 14 includes some example predictions of our SPIDER trained models from Tables 5-6. For each example we describe its question and target (gold) SQL annotation, followed by each model's result.

Model	Supervision	Training set	SPIDER dev.	ACADEMIC	GEO880	IMDB	YELP
T5-SQL-G	$\langle x_i, Q_i, D_i \rangle$	7,000	68.0 $\pm$ 0.3	7.9 $\pm$ 1.3	33.6 $\pm$ 2.5	19.1 $\pm$ 2.9	25.3 $\pm$ 1.7
T5-SQL-G_{part}	$\langle x_i, Q_i, D_i \rangle$	5,349	66.4 $\pm$ 0.8	4.9 $\pm$ 1.7	32.4 $\pm$ 1.3	21.1 $\pm$ 0.7	26.1 $\pm$ 1.0
T5-QDMR-G	$\langle x_i, a_i, s_i, D_i \rangle$	5,349	65.8 $\pm$ 0.3	11.2 $\pm$ 1.0	40.4 $\pm$ 1.8	30.3 $\pm$ 3.1	25.8 $\pm$ 5.1
T5-QDMR-P	$\langle x_i, a_i, D_i \rangle$	5,075	62.9 $\pm$ 0.8	8.4 $\pm$ 0.9	39.7 $\pm$ 1.4	27.0 $\pm$ 5.1	28.2 $\pm$ 2.9

Table 12: Model execution accuracy on SPIDER and its performance on cross-database (XSP) examples. Evaluation on ACADEMIC, IMDB and YELP is on the *full datasets*, including examples with empty denotations.

Model	Supervision	Train. set	DB #	SPIDER dev.	ACADEMIC	GEO880	IMDB	YELP
T5-SQL-G	$\langle x_i, Q_i, D_i \rangle$	1,348	30	48.4	2.1	29.6	9.9	22.6
T5-SQL-G_{part}	$\langle x_i, Q_i, D_i \rangle$	1,129	30	47.4	2.6	26.9	14.5	16.9
T5-QDMR-P	$\langle x_i, a_i, D_i \rangle$	1,129	30	46.2	8.4	29.0	16.0	16.9
T5-SQL-G	$\langle x_i, Q_i, D_i \rangle$	2,028	40	54.7	6.3	28.3	18.3	21.0
T5-SQL-G_{part}	$\langle x_i, Q_i, D_i \rangle$	1,440	40	51.3	3.7	21.2	12.2	19.4
T5-QDMR-P	$\langle x_i, a_i, D_i \rangle$	1,440	40	52.1	6.8	27.4	12.2	18.5
T5-SQL-G	$\langle x_i, Q_i, D_i \rangle$	2,076	40	56.2	3.2	25.5	13.0	24.5
T5-SQL-G_{part}	$\langle x_i, Q_i, D_i \rangle$	1,552	40	53.7	2.3	17.8	10.2	22.8
T5-QDMR-P	$\langle x_i, a_i, D_i \rangle$	1,552	40	53.8	6.1	32.3	19.8	21.8

Table 13: Model results on SPIDER dev when trained on predicted QDMRs versus gold SQL. We train separate models on each of the three randomly sampled training sets. Results include the performance on XSP examples where the evaluation on ACADEMIC, IMDB and YELP is on the *full datasets*, including examples with empty denotations.

Question:	Return me the total citations of papers in the VLDB conference in 2005.	
Target SQL:	`select sum ( publication_0.citation_num ) from conference as conference_0, publication as publication_0 where conference_0.name = "VLDB" and publication_0.year = 2005 and conference_0.cid = publication_0.cid;`	
T5-SQL-G:	`select sum(t1.citation_num) from publication as t1 join conference as t2 on t1.cid = t2.cid where t2.name = "VLDB" and t1.year < 2005;`	✓
T5-QDMR-G:	`SELECT SUM(cite.cited) FROM publication, cite, conference WHERE conference.cid = publication.cid AND publication.pid = cite.cited AND conference.cid IN ( SELECT conference.cid FROM conference WHERE conference.name = 'VLDB' ) AND publication.year < 2005;`	✗
T5-QDMR-P:	`SELECT SUM(cite.cited) FROM cite, publication, conference WHERE publication.pid = cite.cited AND conference.cid = publication.cid AND conference.cid IN (SELECT conference.cid FROM publication, conference WHERE conference.cid = publication.cid AND conference.name = 'VLDB' AND publication.year < 2005);`	✗
Question:	Return me the papers written by H. V. Jagadish and Yunyao Li after 2005.	
Target SQL:	`select publication_0.title from author as author_0, author as author_1, publication as publication_0, writes as writes_0, writes as writes_1 where author_0.name = "Yunyao Li" and author_1.name = "H. V. Jagadish" and publication_0.year > 2005 and author_0.aid = writes_0.aid and author_1.aid = writes_1.aid and publication_0.pid = writes_0.pid and publication_0.pid = writes_1.pid;`	
T5-SQL-G:	`select t3.title from author as t1 join writes as t2 on t1.aid = t2.aid join publication as t3 on t2.citation_num = t3.pid where t1.name = "H. V. Jagadish" intersect select t3.title from author as t1 join writes as t2 on t1.aid = t2.aid join journal as t3 on t2.journal = t3.jid where t3.name = "Yunyao Li";`	✗
T5-QDMR-G:	`SELECT publication.abstract FROM author, publication, writes WHERE publication.pid = writes.pid AND writes.aid = author.aid AND author.name = 'h.v.jagadish' AND publication.year > 2005`	✗
T5-QDMR-P:	`SELECT journal.homepage FROM journal, author, publication, writes WHERE journal.jid = publication.jid AND publication.pid = writes.pid AND writes.aid = author.aid AND author.name = 'h.v.jagadish' AND publication.year > 2005`	✗
Question:	What is the shortest river?	
Target SQL:	`SELECT river.river_name FROM river WHERE river.length = (SELECT min (river.length) FROM river);`	
T5-SQL-G:	`select length from river order by length limit 1;`	✗
T5-QDMR-G:	`SELECT river.river_name FROM river WHERE river.river_name IN ( SELECT river.river_name FROM river ) ORDER BY river.length ASC LIMIT 1;`	✓
T5-QDMR-P:	`SELECT river.river_name FROM river WHERE river.river_name IN ( SELECT river.river_name FROM river ) ORDER BY river.length ASC LIMIT 1;`	✓
Question:	Find the actor with most number of films.	
Target SQL:	`select actor_0.name from actor as actor_0, cast as cast_0, movie as movie_0 where actor_0.aid = cast_0.aid and cast_0.msid = movie_0.mid order by count(distinct(movie_0.title)) desc limit 1;`	
T5-SQL-G:	`select t1.name from actor as t1 join cast as t2 on t1.aid = t2.id group by t1.aid order by count(*) desc limit 1;`	✗
T5-QDMR-G:	`SELECT actor.name FROM actor, movie, cast WHERE actor.aid = cast.aid AND cast.msid = movie.mid GROUP BY actor.name ORDER BY COUNT(movie.mid) DESC LIMIT 1;`	✓
T5-QDMR-P:	`SELECT actor.name FROM actor, cast, movie WHERE actor.aid = cast.aid AND cast.msid = movie.mid GROUP BY actor.name ORDER BY COUNT(movie.mid) DESC LIMIT 1;`	✓
Question:	Which business has the most number of checkins?	
Target SQL:	`select business_0.name from business as business_0, checkin as checkin_0 where business_0.business_id = checkin_0.business_id group by business_0.name order by sum(checkin_0.count) desc limit 1;`	
T5-SQL-G:	`select t1.name from business as t1 join checkin as t2 on t1.business_id = t2.business_id group by t2.business_id order by count(*) desc limit 1;`	✗
T5-QDMR-G:	`SELECT business.name FROM checkin, business WHERE business.business_id = checkin.business_id GROUP BY business.name ORDER BY COUNT(checkin.cid) DESC LIMIT 1;`	✗
T5-QDMR-P:	`SELECT business.name FROM checkin, business WHERE business.business_id = checkin.business_id GROUP BY business.name ORDER BY COUNT(checkin.cid) DESC LIMIT 1;`	✗

Table 14: Example predictions of the SPIDER trained models from Tables 5-6. We denote correct and incorrect predictions by ✓and ✗.