Weakly Supervised Mapping of Natural Language to SQL through Question Decomposition

Tomer Wolfson, Jonathan Berant, and Daniel Deutch. 2021. Weakly Supervised Mapping of Natural Language to SQL through Question Decomposition. In Proceedings of ACM Conference (Conference'17). ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnn.nnnnnn

1 INTRODUCTION

The development of Natural Language Interfaces to Databases (NLIDBs) has been extensively studied in the literature [1, 27, 32, 54]. Existing solutions may roughly be categorized to either rule-based systems [31, 45, 47] or models based on Machine Learning (ML) [58, 66, 70]. An inherent advantage of ML-based systems is that they mitigate the need for laborious manual crafting of rules mapping Natural Language (NL) to SQL. In addition, ML-based NLIDBs tend to outperform rule-based solutions, achieving state-of-the-art performance on standard NL-to-SQL benchmarks [13, 44, 65]. However, machine learning models require training data which, in the context of NLIDBs, consists of labeled pairs of NL questions and their corresponding SQL queries. Generating copious amounts of NL-SQL data is often cost-prohibitive as it requires expert annotators familiar with SQL. Furthermore, a reliable NLIDB is expected to generalize to new domains and databases. Since the mapping task may differ significantly across domains, it is often difficult to re-use existing training data [53] (or existing rules, in rule-based systems). Whenever one wishes to deploy an NLIDB in a given domain, new NL-SQL examples (or handcrafted mapping rules) are needed, requiring yet another costly interaction with experts.

In this paper we propose a weakly supervised approach for training machine learning-based NLIDBs, that does not require expert annotators. Namely, we avoid the use of labeled NL-SQL pairs, and rely exclusively on data procured from non-expert users. Figure 1 presents a high-level view of the input data (in yellow) that we use to synthesize SQL queries (in green) which, in turn, are used to train an NL-to-SQL model. The supervision signals consist of the question’s answer and uniquely, a structured representation of the question decomposition, called QDMR, whose annotation can be effectively crowdsourced by non-experts [63]. Question decompositions provide an effective intermediate representation in SQL synthesis as they are expressed using NL, enabling annotation by laypeople, while being sufficiently structured to be mapped to SQL.

In a nutshell, QDMR is a series of computational steps, expressed by semi-structured utterances, that together match the semantics of the original question. An example appears in the top right of Figure 1. The computational process that answers “Which authors have more than 10 papers in the PVLDB journal?” is broken into five steps (where each step may refer to previous ones): #1. focus on papers; #2. further focus on those published in PVLDB; #3. find the authors of each paper in #2; #4. find the number of #2 (i.e., of papers in PVLDB) for each author retrieved in #3; and finally #5. return the #3 (i.e., authors) where #4 (i.e., number of papers in PVLDB) exceeds 10. As QDMR is derived entirely from the original question, it is agnostic to the underlying domain, schema or even the form of knowledge representation. It has been applied to questions on text and images and relational databases [16, 46, 52]. Here, we utilize QDMR structure and show that it can successfully be mapped to SQL. Furthermore, we are the first to train models using predicted QDMRs, generated by a parameterized model, instead of being manually annotated. Our results show that even when automatically generated by a model, QDMRs provide an effective supervision signal compared to manually labeled SQL.
Figure 1: Using data collected by non-experts as weak supervision for automatically synthesizing NL-SQL pairs.

Our models, data and entire codebase are publicly available.¹

2 BACKGROUND

Weakly Supervised Machine Learning. The performance of supervised machine learning models hinges on the availability of labeled data in sufficient quantity and quality. In practice, obtaining large-scale labeled data for new tasks is often cost-prohibitive and therefore poses a significant challenge for supervised learning. Weak supervision (also referred to as distant supervision) is a broad class of methods aimed at reducing the need for humans to manually label large training sets for supervised ML models [8, 21, 24, 69, 71]. Previous works have utilized weak or noisy sources of supervision, such as regular expression patterns [2], indicative keywords for classification [26], alignment rules over existing knowledge bases [35, 43] or heuristic data labeling functions [3, 41, 56]. These different sources can all be used as weak rules for heuristically annotating large amounts of unlabeled data. Weak supervision has been applied to various data types such as MRI sequences and unstructured text, and in domains such as healthcare and e-commerce [4, 15, 42].

In the context of NLIDBs, an influential line of work has been dedicated to supervision in the form of question-answer pairs, often termed as learning from denotations [20, 36, 37]. A key issue in learning to map NL-to-SQL from denotations is the vast search space of potential candidate queries. To address this challenge, previous works focused on a constrained query search space, which limited their application either to simpler factoid questions [7] or single table databases [59]. Contrastly, we use QDMR decompositions in order to find potential candidate SQL queries, resulting in a general solution for training NL-to-SQL models using weak supervision.

Question Decomposition. A question decomposition meaning representation (QDMR) expresses the meaning of a question by breaking it down into simpler sub-questions. Given a question \( x \), its decomposition \( s \) is a sequence of reasoning steps \( s^1, \ldots, s^{|s|} \) 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. Table 1 provides example QDMR operations; for a full description we refer to [63].

Compared to expert annotated SQL, QDMR annotation is often cheaper and does not require that annotators be familiar with either

¹https://github.com/tomerwol/github/qdmr2sql
SQL or the precise structure of the underlying database. Recent work has shown how crowdsourcing can be used to label tens of thousands of NL questions with QDMR [63]. For a comparison with SQL annotations, the popular SPIDER dataset [66] contains 5,693 SQL queries and 10,181 questions that were all annotated by experts. The reported SQL annotation time stood at 500 hours (650 hours including manual review), resulting in an average annotation of 11.38 queries per hour (8.76 including review). In contrast, the reported QDMR annotation time was 2 minutes, i.e., 30 queries per hour, nearly three times faster than that of SQL annotation.

3 SYSTEM OVERVIEW

This section presents our solution for generating NL-SQL training data using weak supervision. We begin by describing the high-level pipeline, while in §4 we review the different modules of our system.

Our system pipeline is presented in Figure 2. It receives as input a set of weakly supervised training examples \((x_i, a_i, s_i, D_i)\). Each example consists of an NL question \(x_i\) over a target database \(D_i\), its answer \(a_i\) and \(s_i\), the QDMR decomposition of \(x_i\). We assume \(x_i, a_i, s_i\) are manually annotated.\(^2\) Given an input example, our goal is to synthesize a SQL query \(Q_i\) that matches the intention of \(x_i\) and, in particular, executes to \(a_i\), i.e., \(Q_i(D_i) = a_i\). All examples for which a query \(Q_i(D_i) = a\) has been synthesized, are then used as supervised NL-SQL training data \((x_i, Q_i)\). In our experiments (§5) we use the synthesized data to fine-tune a pre-trained language model [40]. However, our data generation process is completely independent of the supervised machine learning model of choice.

We note that our system’s input is comprised entirely of weakly supervised data that can be procured without the use of expert annotators. As described in §2, question-answer annotations can be provided by non-experts, unfamiliar with SQL [6, 36, 61, 68]. As for QDMR instances, they can also be crowdsourced to non-experts [63] or automatically generated using a trained ML model [22, 49].

4 WEAKLY SUPERVISED SQL SYNTHESIS

As described in §3, our system is provided with examples \((x_i, a_i, s_i, D_i)\) of questions, answers, QDMR decompositions and databases. Using this data, it automatically generates a SQL query \(\hat{Q_i}\) which executes to the correct answer \(a_i\). Algorithm 1 describes our SQL synthesis procedure. Given a QDMR \(s_i\) and a database \(D_i\), it incrementally builds a candidate SQL \(\hat{Q}_i\) by iterating over the QDMR steps \(s_i^k\), linking phrases to columns in \(D_i\) and inferring relevant join paths. Next, we detail the various modules used in synthesizing query \(\hat{Q}_i\).

4.1 Phrase-to-DB Schema Linking

The goal of the Phrase-DB Linking component is to link phrases in \(s\) to their corresponding database columns and values in \(D\). Step 2 in Figure 2 depicts the mapping of phrases in \(s\), copied from \(x\) ("papers", "PVLDB", "authors"), to concrete database values. For example, the phrase "papers" is linked to column \texttt{publication.title}. For literal values, our system does not handle named entity recognition and therefore assumes such values in \(D\), e.g., strings or dates, appear verbatim in the database as they do in the original question. We analyze the effect of this assumption in our error analysis in §5.2.2. Hence, the phrase "PVLDB" is automatically linked to the value "PVLDB" stored in the column \texttt{journal.name} of \(D\). Next, we describe the implementation details of linking phrases to columns and values.

Value linking: First, the linker identifies all literal values in \(D\) that are explicitly mentioned in \(s\), using exact string match. For each value mention, it returns both the value and its relevant column in \(D\). When the same value appears in multiple columns, all relevant columns can be returned, which will result in multiple candidate assignments. E.g., the phrase "Smith" may be linked to both the column \texttt{student.lastname} as well as \texttt{faculty.lastname} if both a student and faculty with this last name appear in \(D\).

Column linking: Given an input phrase, our goal is to align it with a corresponding column in \(D\). We enumerate all columns in \(D\) and rank them based on their similarity to the input phrase, with the top ranked column returned as output. The phrase-column ranking is implemented by first lemmatizing the individual tokens of both the
phrase and candidate column, using the Wordnet lemmatizer. Next, we search for columns with tokens overlapping with those of the lemmatized phrase. Last, we compute the similarity scores between the phrase and each column using their GloVe word embeddings similarity [38]. Alternative alignment models may easily be integrated into our pipeline. However, as database linking models are orthogonal to our work, we leave their study for future work.

During the DB linking phase, each phrase may be potentially linked to multiple columns or values in D. In Figure 2, the phrase “authors” is at first incorrectly linked to the column author .aid. To disqualify incorrect linking candidates we use the original answer a. In §4.4 we describe the Execution-guided SQL Search used to search for a correct candidate \( \hat{Q} \) based on its execution results \( \hat{Q}(D) \).

### 4.2 Join Path Inference

Following the alignment of phrases in \( s \) to corresponding columns in \( D \), our next step is to infer the relevant join paths between these columns. Specifically, given two sets of columns, the join path inference module returns the shortest join path connecting both groups. Algorithm 1 uses the procedure ShortestJoinPath in line 9 to join all columns mentioned in QDMR step \( s^k \) with those mentioned in previous steps \( s^j \), that are referenced by \( s^k \) (line 5). We use the schema structure of \( D \) in order to compute join paths. First, \( D \) is converted to a graph, where nodes correspond to tables and edges correspond to foreign-key constraints between tables. Then, given two column sets, ShortestJoinPath computes the shortest join path connecting any two tables \( t_i, t_j \) such that \( t_i \in \text{tables}(\text{cols}) \) and \( t_j \in \text{tables}(\text{other}_\text{cols}) \). If multiple shortest paths exist, it selects the first path which contains either a column \( c_i \in \text{cols} \) as its start node or \( c_j \in \text{other}_\text{cols} \) as its end node.

Using the shortest join path heuristic has been shown to work very well in practice [19, 53], which is reaffirmed in our experimental findings (see §5). An alternative approach may automatically learn relevant join paths using SQL query logs [5] and we leave such extensions for future work. Figure 2 displays the join path inference performed on the tables corresponding to the columns returned by the phrase-column linker (publication, writes, author, journal). The inferred join paths (underlined) are later incorporated when synthesizing \( \hat{Q} \) using the SQL Mapper.

#### 4.3 QDMR to SQL Mapper

The next step is the mapping of the QDMR steps, linked columns and the inferred join paths to executable SQL queries by the MAP-SQL procedure (line 11 of Algorithm 1). The candidate SQL \( \hat{Q} \) is synthesized incrementally based on the QDMR steps of \( s \). As described in §2, given a question \( x \), its QDMR \( s \) is a list of simple questions \( s^1, ..., s^n \) that, when answered in sequence, return the question’s answer \( a \). Each step represents a specific query operation which either selects a set of entities, retrieves information about their attributes or aggregates information over entities. Column linking and join path inference are performed for each \( s^k \), using the previously described modules (lines 4-9). Then, the SQL Mapper maps the linked QDMR to SQL as follows. First, the query operation of step \( s^k \) is automatically inferred, based on its utterance template (§2). Then, in line 11, MapSQL is provided with the operation type \( o_p \), the linked columns \( \text{cols} \), inferred join paths \( \text{join} \) and references to previous steps \( s^j, j < k \) and their mappings to SQL \( \text{mapping} = \{ \hat{Q}\hat{I} | j < k \} \). Based on this data, we were able to define deterministic mapping rules from each QDMR step type to a corresponding SQL query. Table 1 provides example SQL mapping rules for several QDMR operations, as space does not permit to include all of them. In Figure 2, the “SQL Mapper” box displays the resulting mapped SQL \( \hat{Q} \). Note that the linked columns and inferred join paths are both underlined in \( \hat{Q} \). Additionally, Figure 3 describes the incremental SQL mapping of the QDMR of question “What is the ship name that caused most total injuries?”. Each row represents a separate QDMR step along with its operation, NL utterance

| Table 1: Examples of six of the QDMR operations and their mapped SQL templates. |
|-----------------------------------------------------------------------------|
| **Op** | Example Step | QDMR-SQL Template | Synthesized SQL |
|-------|--------------|-------------------|----------------|
| SELECT | “governments” | SELECT [“governments”].column FROM [“governments”].table | SELECT country.governmentform FROM country |
| FILTER | “in Africa” | (Q) WHERE [“in Africa”].condition | SELECT country.governmentform FROM country WHERE country.continent = “Africa” |
| PROJECT | types of #2 | SELECT [“types”].column FROM [“types”].table WHERE [“types”].column IN (#2) | SELECT country.governmentform FROM country WHERE country.governmentform IN (#2) |
| AGGREGATE | number of #3 | SELECT COUNT([#3].column) FROM [#3].table WHERE [#3].column IN (#3) | SELECT COUNT(country.governmentform) FROM country WHERE country.governmentform IN (#3) |
| GROUP | sum of #1 for each #2 | SELECT SUM([#1].column) FROM [#1].table WHERE [#1].column AND [#2].conditions GROUP BY [#1].column | SELECT SUMcars_data.cylinders FROM car_names, cars_data WHERE cars_data.id = car_names.makeid GROUP BY car_names.makeid |
| AGGREGATE | #1 where #2 is highest | SELECT [#1].column FROM [#1].table WHERE [#1].column AND [#2].conditions ORDER BY [#2].column DESC LIMIT 1 | SELECT car_names.makeid FROM car_names, cars_data WHERE car_names.makeid = cars_data.id AND car_names.makeid IN (#1) ORDER BY cars_data.horsepower DESC LIMIT 1 |

Algorithm 1 Weakly Supervised SQL Synthesis

1: **procedure** SQLSynth(s: QDMR, D: database schema)
2:  **mapping** ← []
3:  for \( s^k \in s = (s^1, ..., s^n) \) do
4:      cols ← PhraseColumnsLink(D, \( s^k \))
5:      refs ← ReferencesSteps(\( s^k \))
6:      join ← []
7:      for \( s^j \in refs \) do
8:          other_cols ← mapping[\( j \)].cols
9:          join ← join + ShortestJoinPath(D, cols, other_cols)
10:     op ← OrTypes(\( s^k \))
11:     \( \hat{Q} \) ← MapSQL(op, cols, join, refs, mapping)
12:    mapping[\( k \)] ← (\( s^k \).cols, \( \hat{Q} \))
13: return mapping[n], \( \hat{Q} \)

---

https://www.nltk.org/
Table 1: Mapping to SQL the QDMR of the question “What is the ship name that caused most total injuries?” Each step $s^k$ is mapped to query $Q^k$, with the last step mapped to the output $Q$. We highlight the added SQL clauses in each step.

| No. | Op. | Step | Col. Linking | SQL | Aug. Utterance |
|-----|-----|------|--------------|-----|----------------|
| #1 | SELECT | the Mississippi river | SELECT river.river_name FROM river WHERE river.river_name = "Mississippi" | | What is the name of ships whose populations run through states? |
| #2 | PROJECT | state run through | SELECT state.state_name FROM state, river WHERE river.traverse = state.state_name AND river.river_name IN (#1) | | Which states? |
| #3 | PROJECT | the populations of | SELECT state.population FROM state, river WHERE river.traverse = state.state_name AND state.state_name IN (#2) | | Which population? |

Figure 3: Mapping to SQL the QDMR of the question “What is the ship name that caused most total injuries?” Each step $s^k$ is mapped to query $Q^k$, with the last step mapped to the output $Q$. We highlight the added SQL clauses in each step.

Figure 4: QDMR to SQL mapping of the question “What are the populations of states through which the Mississippi river runs?”. Previously mapped steps are used as nested queries.

and linked columns. Last, we present two examples of MapSQL involving nested queries and self-joins.

Nest SQL Example: During the synthesis of $Q$, the mapper leverages the references to previous QDMR steps by incorporating their SQL mappings as nested queries. Figure 4 displays the mapping of three QDMR steps to SQL. The PROJECT steps #2 and #3 refer to steps #1 and #2 respectively. Hence, the previously mapped SQL of the referenced steps is used as a nested query in the condition clause, denoted in Figure 4 by its reference for brevity. Similarly, other QDMR operations also copy specific clauses or sub-queries from their referenced steps, as shown in Table 1.

Self Join Example: The SQL mapper is able to automatically handle queries whose synthesis requires self-joins. In Figure 5, steps #2 and #3 enforce the conditions $author.name = 'H. V. Jagadish'$ and $author.name = 'Yunyao Li'$. These contradictory assignments to $author.name$ are automatically identified as a self-join and resolved by using a nested query in step #3 of the example.

4.4 Execution-guided SQL Candidate Search

During SQL synthesis, the phrase linking component ($\mathcal{P}_L$) results in multiple candidate queries $Q$ being generated for each potential phrase-to-column alignment. Step 2 in Figure 2 presents a potential column linking where the phrases “papers” and “authors” are linked to publication.title and author.aid respectively. As the original question refers to the authors names, the resulting $\hat{Q}$ is incorrect due to linking “authors” to author.aid instead of author.name.

An incorrect $\hat{Q}$ may also be synthesized due to a mismatch between the QDMR and database structure. As it is agnostic to the underlying database, $s$ may fail to correctly capture database-specific language. E.g., in the question “How many student enrolled during the semester?”, $D$ may have a numeric column course.num_enrolled requiring to SUM all those enrolled for each course. Alternatively, $D$ might include a column course.student_id for each student enrolled, which will instead require to COUNT those students.

To filter out incorrect candidates $\hat{Q}$ we utilize an Execution-guided Candidate Search, based on the original question results $a$. The search terminates once it finds a candidate $Q$ which successfully executes to $a$, i.e., $\hat{Q}(D) = a$. We implement four search heuristics in order to avoid an exhaustive search over all potential candidates, which is exponential in the columns of $D$. These heuristics are domain-agnostic and are not tailored to a specific database.

1. Phrase linking search: We avoid iterating over each phrase-column assignment by ranking them according to their phrase-column similarity, as described in §4.1. The query $\hat{Q}(D)$ is induced from the top ranked assignment, where each phrase in $s$ is assigned its most similar column. If $\hat{Q}(D) \neq a$ we continue the candidate search using heuristics 2-4. Assuming that the additional search heuristics failed to find a candidate $\hat{Q}$ which successfully executes to $a$, i.e., $\hat{Q}(D) = a$, we return to the phrase linking component and resume the process using the candidate SQL induced from the following assignment $\hat{Q}(D)$, and so forth. In practice, we limit the number of assignments and review only the top-k most similar columns for each phrase in $s$, where $k = 20$. Our subsequent error analysis (§5.2) reveals that only a small fraction of failures are due to limiting $k$. Step 3 in Figure 2 represents the iterative process, where $\hat{Q}(D)$ executes to an incorrect result while the following candidate $\hat{Q}(D)$ correctly links “authors” to author.name and executes to $a$, ending the search.
2. **Distinct modification**: Given a candidate SQL \( \hat{Q} \) s.t. \( \hat{Q}(D) \neq a \), we add `DISTINCT` to its SELECT clause. In Table 2 the modified SQL executes to the correct result and is therefore returned.

3. **Superlative modification**: This heuristic automatically corrects semantic mismatches between annotated QDMR structures and the underlying \( D \). Concretely, steps in \( s \) representing `PROJECT` and `FILTER` operations may entail an implicit `ARGMAX/ARGMIN` operation. E.g., for the question “What is the size of the largest state in the USA?” in Table 2, step \#3 of its QDMR represents a `PROJECT` operation, “state with the largest \#2” which entails an `ARGMAX`. Using the NLTK part-of-speech tagger, the modification procedure automatically identifies any superlative tokens in `PROJECT` and `FILTER` steps of \( s \). These steps are then replaced with the appropriate `ARGMAX/ARGMIN`. In Table 2 the original step \#3 is modified to the `ARGMAX step “#1 where #2 is highest”`.

4. **Aggregate modification**: This heuristics replaces instances of `COUNT` in QDMR steps with `SUM` operations, and vice-versa, giving the database structure. In Table 2, 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 \( \hat{Q} \) correctly executes to \( a \) and is thereby returned.

5. **EXPERIMENTS**

We evaluate our data generation pipeline in two main settings: First in §5.2, we examine the coverage of our data generation approach, i.e., the percentage of questions for which we are able to generate accurate SQL queries. Second, we test the efficacy of our synthesized training data and compare it to that of gold SQL, annotated by experts (§5.3). Accordingly, we fine-tune a powerful pre-trained language model [40] and compare its performance when trained on our synthesized data to that when trained on gold SQL.

5.1 **Setting**

We experiment on five benchmark datasets for mapping NL-to-SQL. The first four datasets contain questions posed over a single database, each in a unique domain: `ACADEMIC` [31] contains questions over the Microsoft Academic Search database; `Geo880` [67] concerns US geography; `IMDB` and `Yelp` [64] contain complex questions on a film and restaurant database, respectively. Last, the `Spider` dataset [66] is unique as it measures domain generalization in mapping NL-to-SQL, containing questions over 160 databases.

As our question decompositions, we use the annotated QDMRs of NL-to-SQL datasets found in the `Break` dataset [63]. The only exceptions are the 259 questions of `IMDB` and `Yelp` which we manually label with corresponding QDMR annotations, as they are not in `Break`. Table 3 displays the number of QDMR-annotated examples used for each dataset.

Our NL-to-SQL models are pre-trained language models based on the Transformer architecture [57]. Specifically, we use the `T5-large` model [40] as it has been previously shown to perform competitively with state-of-the-art methods [11, 48]. We use the model implementation by HuggingFace [62] and train `T5-large` using the Adam optimizer [28]. Following fine-tuning on the development set, we adjust the batch size to 128 and its learning rate to 1e-4 (after experimenting with 1e-5, 1e-4 and 1e-3). All of our models were trained on a single NVIDIA GeForce RTX 3090 GPU.

5.2 **QDMR to SQL Data Generation**

Our first challenge is to evaluate our data generation pipeline in producing accurate SQL queries, given weak supervision. As our evaluation score we use SQL synthesis coverage, i.e., the percentage of examples where our procedure successfully produces a query which executes to the correct answer.

5.2.1 **Human-labeled Decompositions.** The upper rows in Table 3 present the SQL synthesis coverage when using manually annotated QDMRs from `Break`. We note that the synthesis coverage for single-schema datasets tends to be slightly higher than for the multi-schema `Spider` dataset, which we attribute to its larger size and diversity. As `Spider` contains thousands of questions over 160 databases, some QDMRs may not be correctly aligned with parts of their database structure (§4.4). We manually validate a random subset of 100 synthesized SQL from all datasets, revealing that 95% of them are indeed correct interpretations of the original question. Additionally, we conduct an error analysis on a random set of 100 failed examples, presented in Table 4. Synthesis errors are mostly due to annotation errors and implicit database-specific conditions. E.g., in `Geo880` the phrase “major river” should implicitly map to the condition `river.length > 750`. As our SQL synthesis is database-agnostic, it does not memorize domain-specific jargon.

5.2.2 **Automatically Predicted Decompositions.** While QDMR annotations can be crowdsourced at scale [63], moving to a new domain may potentially require a new set of `NL-QDMR` pairs, thereby incurring additional annotation costs. Instead, we utilize existing `NL-QDMR` corpora in order to train a parameterized model \( f \) which given a question \( x \) automatically parses it to its decomposition, i.e., \( f(x) = \tilde{s} \). As our QDMR parser, we train a sequence-to-sequence `T5-large` model on the `Break` dataset. The model is trained for 10 epochs and fine-tuned based on the exact string match (EM) on development set examples. Evaluating our parser on the `Break` test set reveals an EM score of 0.23, on par with the current state-of-the-art model.\(^5\) Using our trained parser, we generate decompositions \( \tilde{s} \) for questions in the development and test sets of `Spider` and `Geo880`. We avoid generating \( \tilde{s} \) for training set examples, as their questions were used for training our QDMR parser. As the `Spider` test set is hidden we are unable to include its examples. Then, the predicted \( \tilde{s} \), are incorporated in examples \( \langle x, a_i, \tilde{s}_i, D_i \rangle \) provided to our SQL synthesis procedure. The bottom lines of Table 3 list the SQL synthesis coverage of the predicted QDMRs. Somewhat surprisingly, the coverage when using predicted \( \tilde{s} \) exceeds that of manually annotated QDMRs: 77.6% compared to 77.2% on `Spider` dev; 86.1% to 85.5% on `Geo880` dev & test. We attribute the superior performance to the existence of annotations in `Break` that are either erroneous or include a step which does not conform to the QDMR step templates. Overall, our SQL synthesis is applicable to a broad range of domains. It is capable of translating a diverse set of questions, over 164 databases, which express a wide variety of user intents. Last, the strong performance of SQL synthesis from predicted QDMRs

---

\(^4\)These annotations are released along with our entire codebase.

\(^5\)https://leaderboard.allenai.org/break
5.3 Mapping NL to SQL

Next, we examine the efficacy of our synthesized data in training NL-to-SQL models, compared to expert SQL annotations. Given examples \((x_i, D_i)\) of questions over databases, we experiment with two different training sets with the following supervision signals: (1) A strongly supervised training set, where queries \(\hat{Q}_i\) are annotated by experts for each \(x_i\) and the model is trained on examples \(\{(x_i, \hat{Q}_i, D_i)\}_{i=1}^n\). (2) A weakly supervised training set where, instead of expert annotated SQL, we are provided with the answer \(a_i\) and QDMR \(s_i\) (either annotated or predicted, see §5.3.5). Using SQL synthesis on the input \(\{(x_i, a_i, s_i, D_i)\}_{i=1}^m\) we obtain queries \(\hat{Q}_i\). As SQL synthesis coverage is not 100% the process returns a subset \(\{(x_i, \hat{Q}_i, D_i)\}_{i=1}^m\) of \(m < n\) correctly synthesized examples. The NL-to-SQL model is then trained on these synthesized examples.\(^6\)

\(^6\)In practice, we do not train directly on \(\hat{Q}_i\) but on a representation of \(s_i\) and its phrase-column linking which is automatically mapped to SQL for execution.

### Table 2: Example of candidate search heuristics during SQL synthesis.

| 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 products.product_type_code FROM products`                                       |
| Superlative modification| What is the size of the largest state in the USA?                       | \[[1] \text{states in the USA}; \ 2 \text{size of } Q_1; \ 3 \text{state with the largest } Q_2; \ 4 \text{size of } Q_3\] | \[[1] \text{states in the USA}; \ 2 \text{size of } Q_1; \ 3 \text{state with the largest } Q_2; \ 4 \text{size of } Q_3\] |
| 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,COUNT(university.enrollment) FROM university GROUP BY university.affiliation` |

5.3.1 Training Data. We experiment with two large NL-to-SQL datasets: Spider [66] and Geo880 [67]. Spider is a widely used dataset for training cross-database NL-to-SQL models. Namely, as there is no overlap between the databases found in the train, dev and test sets of Spider, models are required to generalize to previously unseen domains. For the weakly supervised training set we use all 6,955 training set examples \((x_i, a_i, s_i, D_i)\) that have QDMR annotations in Break. Following SQL synthesis, we successfully generate 5,349 NL-SQL examples \((x_i, \hat{Q}_i, D_i)\). The second dataset we train on is the single-schema Geo880 dataset. We use all 547 of its training examples, with QDMR annotations in Break. SQL synthesis on Geo880 generates 454 NL-SQL training examples. As previously noted, our synthesized training data is smaller than the seed data due to SQL synthesis coverage (Table 3).

5.3.2 QDMR-induced Data Augmentation. In settings where training data is scarce, we employ a data augmentation technique based on QDMR structure. We utilize the incremental structure of QDMR in order to induce synthetic NL-SQL pairs \((\hat{x}_i, \hat{Q}_i)\) directly from its individual steps. As each QDMR step \(s^k\) is expressed in NL, we can derive a synthetic utterance \(\hat{s}^k\) describing it. Synthetic utterance examples are provided in the last column of Figure 3. Utterance \(\hat{s}^k\) is derived directly from \(s^k\), where references to previous steps \(s^j\), \(j < k\), are substituted with their previously induced utterances \(\hat{s}^j\). Last, we prepend a wh-word (“which”, “what”) to the resulting utterance. The corresponding SQL to \(\hat{s}^k\) is the query \(\hat{Q}^k\), generated from step \(s^k\) during the mapping of \(s^k\) to \(\hat{Q}^k\) (§4.3). We therefore generate synthetic NL-SQL pairs for each step \(s^k\) of our seed QDMRs \(s_i\). In total, the augmented data has 24,510 examples induced from the 5,349 QDMRs in Spider and 1,405 examples of the 454 in Geo880. However, the synthetic utterances result in somewhat unnatural language, e.g., “What ships where the number of injuries for each ships is the highest?” in step 4 of Figure 3. To prevent overfitting to the synthetic data we follow [60, 65] by first pre-training the NL-to-SQL model on the augmented \((\hat{x}^k, \hat{Q}^k, D_i)\) data, then fine-tuning it on the original training data.

### Table 3: SQL synthesis coverage scores.

| 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 %     |

### Table 4: SQL synthesis error analysis.

| Error               | Description                                                                 | %    |
|---------------------|------------------------------------------------------------------------------|------|
| Nonstandard QDMR    | An annotated QDMR containing a step that does not conform to its operation template | 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 (§4.4) | 11   |
| Gold SQL error      | Incorrectly annotated \(Q_i\) resulting in execution-guided search for an incorrect \(a = Q(D)\) | 6    |

(of previously unseen questions) eliminates the burden of manually annotating new decompositions. By automatically predicting \(\hat{s}^k = f(x_i)\), we are able to synthesize SQL using exclusively \((x_i, a_i)\) as our supervision.
- **T5-LARGE\textsubscript{Aug}** pre-trained on the augmented synthetic data, then fine-tuned on the same examples as T5-LARGE\textsubscript{QDMR}

For T5-LARGE\textsubscript{Gold} we experiment with two variants, one trained on the entire gold training set \( \left\{ (\mathbf{x}_i, Q_i, D_i) \right\}_{i \geq 1} \) and the other trained on the same examples as T5-LARGE\textsubscript{QDMR} but with the gold SQL. This setting helps measure the degree to which the lower coverage of our generated training data may affect the model performance.

5.3.4 Results. We evaluate our models using the execution accuracy of their predicted queries. Following Suhr et al. [53], the execution accuracy is defined as the percentage of predicted SQL which, when executed against the database, result in tuples containing the same set of rows as \( a \). Tables 5–6 present the execution accuracy results. For each model, we train three separate instances using different random seeds and list their average accuracy and standard deviation. Results of the models trained on Spider are listed in Table 5. All models were trained for 150 epochs and evaluated on the public development set of 1,034 examples. The weakly supervised T5-LARGE\textsubscript{QDMR} achieved on average 65.6% execution accuracy compared to 68.0% when training on annotated SQL. Therefore, T5-LARGE\textsubscript{QDMR} achieves 96.5% of T5-LARGE\textsubscript{Gold} performance, while being trained on 76.4% of its examples (5,349 examples compared to the 7,000). Interestingly, the T5-LARGE\textsubscript{Gold} model trained on the same 5,349 examples as T5-LARGE\textsubscript{QDMR}, but with gold SQL, achieves a slightly higher execution accuracy (by 0.8 points). We attribute this gap to the differences in the query representations \( \hat{Q} \) and \( \hat{Q} \) that each model has used for training. Specifically, our synthesized \( \hat{Q} \) are longer on average than the more concise expert annotated \( \hat{Q} \), averaging 34.2 tokens per query compared to 17.4 on Spider examples (24.7 to 16.0 on Geo880). Sequence-to-sequence models, such as T5-LARGE, are known to depend on the target query language (SQL, \( \lambda \)-calculus, Lisp, etc.), with shorter queries often leading to better performance [18, 23]. However, we leave the more concise encoding of synthesized \( \hat{Q} \) as future work.

As the Spider dataset targets a cross-database setting, we further evaluate its models on out-of-database (OOD) examples. As the OOD examples we evaluated on the datasets ACADEMIC (195), Geo880 (880), IMDB (131) and Yelp (128). Results in Table 5 show that all models struggle to generalize to OOD examples, similar to past findings [53]. However, the T5-LARGE\textsubscript{Gold} model performance is better on OOD examples than T5-LARGE\textsubscript{QDMR}. This result further indicates that training on synthesized SQL from QDMR and answer annotations is competitive compared to training on gold SQL.

The results of models trained on Geo880 are listed in Table 6. All models were trained for 300 epochs and fine-tuned based on development set performance. We evaluate the fine-tuned models execution accuracy on the 280 test examples. On Geo880, T5-LARGE\textsubscript{QDMR} achieves 90.7% of the performance of T5-LARGE\textsubscript{Gold} while trained on 83.0% of its examples. We observe a larger performance gap than in Spider, which we attribute to Geo880 having significantly less training examples, 547 compared to 7,000 (with 454 synthesized SQL compared to 5,349). This may result in each synthesis error affecting the overall performance to a much higher degree than in Spider. The T5-LARGE\textsubscript{Aug} model improves the performance on to 93.2% of that of T5-LARGE\textsubscript{Gold} while being trained exclusively on data synthesized from the 454 weakly supervised examples. The improved performance of T5-LARGE\textsubscript{Aug} on Geo880, in contrast to its performance on Spider (63.9%), is consistent with our intuition for data augmentation. Namely, that T5-LARGE\textsubscript{Aug} does improve performance where training data is more scarce. Due to the smaller training set of Geo880, pre-training on the 1,405 synthetic examples may have had a stronger effect.

5.3.5 Training on Predicted Decompositions. We performed additional experiments to test the efficacy of our approach without using annotated QDMRs in training. Similar to §5.2.2 we train a QDMR parser on Break to automatically map question \( x \) to its decomposition \( f(x) = s \). Using the QDMR parser enables us to synthesize SQL using only \( (x_i, a_i, D_i) \) examples. With the input to the SQL synthesizer being \( (x_i, a_i, s_i, D_i) \) where \( s_i = f(x_i) \). We make sure that there is no overlap between the questions \( x_i \) in the NL-SQL training data and those used to train the QDMR parser \( f \). To achieve this, we randomly select 30–40 databases from Spider train and their corresponding questions. These questions are then discarded from the Break training set and are used exclusively to train our NL-to-SQL models. We experiment with 3 different random samples of Spider train, listed in the upper, middle and lower sections of Table 7. Training the QDMR parser on the remaining Break examples reaches 0.24 EM on its development set (averaged on all 3 samples). Table 7 presents model performance on Spider when training on predicted QDMRs. The results correspond to our 3 random samples of Spider train that number 1,548, 2,028 and 2,076 examples each. The T5-LARGE\textsubscript{QDMR-P} model was trained only on a subset of these examples for which the SQL synthesis was successful. On average, T5-LARGE\textsubscript{QDMR-P} achieves 95.5% of the performance of T5-LARGE\textsubscript{Gold}. This result shows that even when queries are induced from \( (x_i, a_i) \) annotations and a QDMR parser, we are still able to train models that are competitive with those trained on gold SQL.

Overall, our NL-to-SQL experiments demonstrate the quality of our synthesized training data. We experiment both with annotated QDMRs as well as with automatically predicted decompositions. Experiments on the Spider and Geo880 datasets show weakly supervised models though trained on 17%-24% fewer examples, achieve 93.2%-96.5% of the performance of their fully supervised counterparts. Using predicted QDMRs instead of human annotations remains competitive with gold SQL supervision. With models trained on predicted QDMRs reaching, on average, 95.5% of the fully supervised model performance on Spider.

6 RELATED WORK

The relation between our approach and weakly supervised machine learning has been discussed in §2. For a thorough review of natural language interfaces to databases we refer to [1, 27, 53].

Research on NLIDBS can be broadly grouped into two main categories: Rule-based NLIDBS utilize human written rules to map NL utterances to executable SQL over a target database. These systems generally apply domain-specific rules, based both on database contents and input phrases, in order to extract query fragments which are then composed into full SQL [31, 39, 45]. To compensate for rigid mapping rules, rule-based NLIDBS may rely on interaction, by displaying multiple SQL translations for the user to choose from [31, 39]. However, rule-based systems tend to require manually-crafted rules to handle new databases or question types [47].
second category is of ML-based NLIDBs, where a parameterized function is trained to map NL-to-SQL. Research on ML-based NLIDBs has gained significant traction in recent years, spurred on by the introduction of large-scale supervised datasets for training models and evaluating their performance [66, 70]. Recent approaches have heavily relied on specialized architectures [19, 48, 58] combined with pre-trained language models [14, 33, 40]. Our solution targets ML-based NLIDBs by synthesizing NL-SQL pairs from weak supervision and can thereby be integrated in training supervised models, whilst mitigating their dependence on manual SQL annotations.

Intermediate meaning representations (MRs) have been previously used for mapping NL-to-SQL. Contrast to QDMR, these MRs necessitate expert annotations [25, 64], or were directly induced from such annotations as a means of enhancing model performance by training directly on the MRs instead of SQL [19, 23, 44, 53].

Table 6: Model results on Geo880 test set of 280 examples.

| Model                | Supervision | Train. set | Execution % |
|----------------------|-------------|------------|-------------|
| T5-LARGE_GOLD        | (${x}_i, Q_i, D_i$) | 1,348 | 30 | 48.4 |
| T5-LARGE_GOLD        | (${x}_i, Q_i, D_i$) | 1,129 | 30 | 47.4 |
| T5-LARGE_GOLDDB      | (${x}_i, a_i, D_i$) | 1,129 | 30 | 46.2 |
| T5-LARGE_GOLD      | (${x}_i, Q_i, D_i$) | 1,440 | 40 | 54.2 |
| T5-LARGE_GOLD_P     | (${x}_i, a_i, D_i$) | 1,440 | 40 | 54.2 |
| T5-LARGE_GOLD      | (${x}_i, Q_i, D_i$) | 2,028 | 40 | 57.1 |
| T5-LARGE_GOLD_P     | (${x}_i, a_i, D_i$) | 1,552 | 40 | 53.7 |
| T5-LARGE_GOLD      | (${x}_i, Q_i, D_i$) | 1,552 | 40 | 53.8 |

Table 7: 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.

| Model                | Supervision | Train. set | Execution % |
|----------------------|-------------|------------|-------------|
| T5-LARGE_GOLD        | (${x}_i, Q_i, D_i$) | 7,000 | 68.0 ± 0.3 | 7.9 ± 1.3 |
| T5-LARGE_GOLD        | (${x}_i, Q_i, D_i$) | 5,349 | 66.4 ± 0.8 | 4.9 ± 1.7 |
| T5-LARGE_GOLDDB      | (${x}_i, a_i, s_i, D_i$) | 5,349 | 65.6 ± 0.3 | 11.2 ± 1.0 |

as natural language is inherently ambiguous [12, 64]. Similar to our approach, DUOQUEST [6] combines both NL questions and answer tuples for SQL synthesis. However, it crucially depends on expert users both at test time, for user interaction, and in training their ML-based NLIDB on expert annotated NL-SQL examples [66].

Most similar to our work is that of [46] which generates executable SPARQL queries using QDMR annotations on Spider. Contrastly, our approach does not rely on the additional supervision of manually annotated linking of question tokens to DB elements, released by [30]. In addition, we extend our mapping of QDMR to SQL to four additional datasets beyond Spider [31, 64, 67]. Last, we provide additional experiments using predicted QDMRs instead of human annotations. These demonstrate the applicability of our approach when using question-answer pairs as supervision.

7 CONCLUSIONS

Supervised NL-to-SQL models are a cornerstone in building better NLIDBs. However, current state-of-the-art models depend on experts to either code mapping heuristics, in rule-based NLIDBs, or to annotate training sets of NL-SQL examples. This work has presented a weakly supervised approach for generating NL-SQL training data which relies exclusively on data procured by non-experts. We implemented an automatic SQL synthesis procedure, capable of generating thousands of training examples over 164 databases. Experiments on two NL-to-SQL benchmarks demonstrate the effectiveness of our synthesized data. Namely, our weakly-supervised models achieve 93.2%–96.5% of the expert annotated model performance, while training on fewer examples (76%–83%). Overall, our weakly supervised solution enables us to train quality NL-to-SQL models while mitigating expert involvement in NLIDBs.

ACKNOWLEDGMENTS

This research was partially supported by The Israel Science Foundation (grant 978/17), and the Yandex Initiative for Machine Learning and the European Research Council (ERC) under the European Union Horizons 2020 research and innovation programme (grant ERC DELPHI 802800).

REFERENCES

[1] Katrin Aboflor, Kurt Stockinger, and A. Bernstein. 2019. A comparative survey of recent natural language interfaces for databases. The VLDB Journal 28 (2019), 793 – 819.
[2] Isabelle Augustin, Tim Rocktaschel, Andreas Vlachos, and Kalina Bontcheva. 2016. Stance Detection with Bidirectional Conditional Encoding. In EMNLP.
[3] Abhijeet Awasthi, Sabysachi Ghosh, Ranza Goyal, and Sunita Sarawagi. 2020. Learning from Rules Generalizing Labeled Exemplars. In 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020.
[4] Stephen H. Bach, Daniel Rodriguez, Yintao Liu, Chong Luo, Haidong Shao, Candandra Xia, Souvik Sen, Alexander J. Ratner, Braden Hancock, H. Alborzi, Rahul Kuchhal, C. Rê, and Rob Malkin. 2019. Snorkel Drybell: A Case Study in Deploying Weak Supervision at Industrial Scale. Proceedings of the 2019 International Conference on Management of Data (2019).
from Compositional Neural Networks. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics, Online, 5594–5608. https://doi.org/10.18653/v1/2020.acl-main.495

[53] Alane Suhr, Ming-Wei Chang, Peter Shaw, and Kenton Lee. 2020. Exploring Unexplored Generalization Challenges for Cross-Database Semantic Parsing. In ACL.

[54] 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 (2021), 1033–1039.

[55] Quoc Trung Tran, C. Chan, and S. Parthasarathy. 2009. Query by output. Proceedings of the 2009 ACM SIGMOD International Conference on Management of data (2009).

[56] 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 (2021), 1033–1039.

[57] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Advances in Neural Information Processing Systems. 5998–6008.

[58] P. Varma and C. Ré. 2018. Snuba: Automating Weak Supervision to Label Training Data. Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases 12 3 (2018), 223–236.

[59] 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 (2020).

[60] Bailin Wang, Wenpeng Yin, Xi Victoria Lin, and Caimei Xiong. 2021. Learning Semantic Parsers from Denotations with Latent Structured Alignments and Abstract Programs. In EMNLP.

[61] Bailin Wang, Ivan Titov, and Mirella Lapata. 2019. Learning Semantic Parsers from Denotations with Latent Structured Alignments and Abstract Programs. In EMNLP.

[62] Bailin Wang, Wenpeng Yin, Xi Victoria Lin, and Caimei Xiong. 2021. Learning to Synthesize Data for Semantic Parsing. In NAACL.

[63] C. Wang, A. Cheung, and Rastislav Bodík. 2017. Synthesizing highly expressive SQL queries from input-output examples. Proceedings of the 58th ACM SIGPLAN Conference on Programming Language Design and Implementation (2017).

[64] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi 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 M. 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. Association for Computational Linguistics, Online, 38–45. https://www.aclweb.org/anthology/2020.emnlp-demos.6

[65] Tomer Wolfson, Mor Geva, Ankur Gupta, Matt Gardner, Y. Goldberg, D. Deutch, and Jonathan Berant. 2020. Break B Down: A Question Understanding Benchmark. Transactions of the Association for Computational Linguistics 8 (2020), 183–198.

[66] Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. SQLizer: query synthesis from natural language. Proceedings of the ACM on Programming Languages 1 (2017), 1 – 26.

[67] Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinxi Yang, Dragomir R. Radev, Richard Socher, and Caimei 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. https://openreview.net/forum?id=kydLEyHzZ.

[68] 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 (2019).

[69] Z. Zhou. 2018. A brief introduction to weakly supervised learning. National Science Review 5 (2018), 44–53.