Semi-Automatic Construction of Text-to-SQL Data for Domain Transfer

Tianyi Li[⋆], **Sujian Li**[†] and **Mark Steedman**[⋆]

[⋆]University of Edinburgh
[†]MOE Key Lab of Computational Linguistics, School of EECS, Peking University
tianyi.li@ed.ac.uk, lisujian@pku.edu.cn, steedman@inf.ed.ac.uk

Abstract

Strong and affordable in-domain data is a desirable asset when transferring trained semantic parsers to novel domains. As previous methods for semi-automatically constructing such data cannot handle the complexity of realistic SQL queries, we propose to construct SQL queries via context-dependent sampling, and introduce the concept of *topic*. Along with our SQL query construction method, we propose a novel pipeline of semi-automatic Text-to-SQL dataset construction that covers the broad space of SQL queries. We show that the created dataset is comparable with expert annotation along multiple dimensions, and is capable of improving domain transfer performance for SOTA semantic parsers.

1 Introduction

Due to the broad use of SQL in real-world databases, the task of mapping natural language questions to SQL queries (Text-to-SQL) has drawn considerable attention. Several large-scale cross-domain Text-to-SQL datasets have been manually constructed and advanced the development of Text-to-SQL semantic parsing (Zhong et al., 2017; Yu et al., 2018).

While these datasets are built for domain-general semantic parsing, current state-of-the-art (SOTA) semantic parsers still suffer sharp performance drop when generalising to unseen domains (Wang et al., 2020; Guo et al., 2019; Zhang et al., 2019).

This could be attributed to the observation that the mapping of Text-to-SQL vary vastly across different domains, particularly in terms of the expressions of predicates[1]. It is very difficult for models to generalize to those variations in a zero-shot fashion. Thus, additional in-domain data is desirable when applying semantic parsers to novel domains.

Unfortunately, the cost and scarcity of supervised data have been a major barrier for the wider application of the Text-to-SQL task, as creating pairs of natural language questions and SQL queries is a complex task demanding expertise in both SQL language and the specific domains. Take the SPIDER dataset (Yu et al., 2018) for example, 10,181 Text-SQL pairs in 200 databases (from 138 domains) required 11 computer science graduates to invest 1,000 human hours.

In semantic parsing, some semi-automatic dataset construction methods have been proposed. Wang et al. (2015) built logical forms compositionally, converted them to rigid pseudo natural language (Pseudo-NL) questions with rules, then crowd-sourced those Pseudo-NL questions into NL questions. Cheng et al. (2018) further broke down the pseudo-NL questions into question sequences to make them more digestible for crowd workers.

While these approaches shed light on the methodology of semi-automatic construction of semantic parsing datasets, applying them to collect broad-coverage Text-to-SQL data for domain transfer is not trivial. Firstly, SQL language has a much larger variety of realistic queries than Lambda-DCS logical forms (Liang, 2013), which were the focus of earlier work. Blind enumeration-up-to-a-certain-depth from a CFG is therefore intractable in size. Secondly, Herzig and Berant (2019) have discovered a mismatch between semi-automatically constructed queries and real-world queries, in terms of the distribution of logical form and the style of natural language expressions. As achieving accuracy gains in domain transfer demands high quality for the in-domain data, narrowing these mismatches is crucial.

In this paper, we propose a novel semi-automatic pipeline for robust construction of Text-SQL pairs as training data in novel domains, with broad semantic coverage, from databases only. Following

[1]An example illustrating such difference is presented in Appendix A

Proceedings of the 17th International Conference on Parsing Technologies (IWPT 2021), pages 38–49
Bangkok, Thailand (online), August 6, 2021.

Wang et al. (2015), our pipeline consists of three parts: automatic SQL query construction, SQL-to-PseudoNL conversion and PseudoNL-to-NL paraphrasing. In SQL query construction, we use a context-dependent probabilistic approach: first, we choose a topic of interest, a random n-tuple of tables in the novel domain, such as *Concerts and Stadiums*; then, we sample from a set of grammar rules, at each step pruning the generation space based on decision history. In SQL-to-PseudoNL conversion, we follow Cheng et al. (2018) in breaking down constructed SQL queries with templates, but also assign a "dominant concept" to topics to simplify Pseudo-NL questions. In PseudoNL-to-NL paraphrasing, we do crowd annotation, and provide crowd workers with various annotation scaffolds to collect quality NL questions. An example through our pipeline is shown in Figure 1.

We show that by using schema-inspired *topic* and context-dependent sampling instead of blind enumeration, SQL queries analogous to real-world queries can be constructed, and effective fine-tune datasets for domain transfer can be built. Our experiment shows that even a modest amount of our data facilitates domain transfer across a range of semantic parsers, raising accuracy in novel domains by up to 1%[2].

2 Related Work

Alternative Supervision To resolve the difficulty in gathering supervised data for semantic parsing, various methods have been proposed from different perspectives.

Numerous approaches have explored distant supervision to bypass the use of expensive annotated data. Kwiatkowski et al. (2013); Berant et al. (2013); Yao and Van Durme (2014); Berant and Liang (2014) used question-answer pairs as supervision instead of logical form annotations; Reddy et al. (2014) used web-scale corpora of descriptive sentences, formalizing semantic parsing as a graph matching problem. These methods perform well on factoid questions; however, for more complex questions, it is harder to infer the underlying queries from the answers alone.

Later on, semi-automatic data collection methods, which reduce the cost of annotation, have been given considerable attention. Wang et al. (2015) propose to compose logical forms by combining sub-components through a grammar, and translate the resulting trees into NL questions via rigid Pseudo-NL questions and crowd-paraphrasing. Cheng et al. (2018) further replace the Pseudo-NL questions with question sequences to simplify the annotation for individual sentences.

While those approaches pioneered semi-automatic data collection, the query construction method based on exhaustive enumeration has its weaknesses. Herzig and Berant (2019) showed that there exists a significant mismatch between the distributions of created logical forms and real query logical forms, and between the language style of paraphrases and real questions.

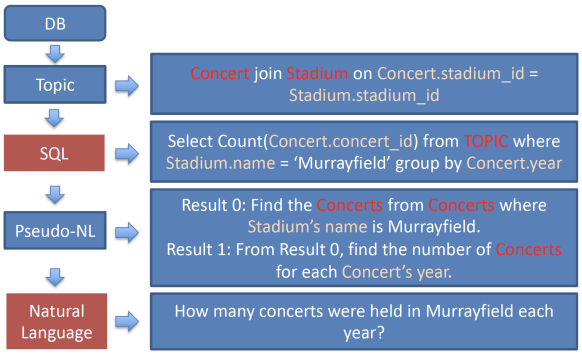

Figure 1: A example of the data construction pipeline, from the topic of "concerts and stadiums" to the final results: the SQL query and the paired natural language question.

Text-to-SQL Text-to-SQL as a semantic parsing task, has attracted increasing interest, where multiple large-scale datasets have been released. Zhong et al. (2017) created a large single-table Text-to-SQL dataset, WikiSQL, from Wikipedia entries, upon which many semantic parsers have been trained, achieving high accuracies surpassing 80% (Chang et al., 2020; Lyu et al., 2020; Wang et al., 2018; Hwang et al., 2019; He et al., 2019). Yu et al. (2018) proposed SPIDER, another large-scale text-to-SQL dataset with multi-table databases, much wider grammar coverage, and more complex queries. It involves features including GROUP-BY aggregation, ORDER-BY, nested queries. This has made SPIDER a more realistic, and also more challenging parsing task, with SOTA semantic parsers achieving accuracies of above 60%. (Guo et al., 2019; Wang et al., 2020).

Although SPIDER is considered a domain-general semantic parsing dataset, semantic parsers

[2]Our code and data will be released at https://github.com/Teddy-Li/SemiAuto_Data_Text_SQL

trained on it still suffer a sharp performance drop when generalizing to unseen domains[3].

Thus, additional resource for domain transfer is appealing. However, in this more complex multi-table Text-to-SQL task, previous semi-automatic dataset construction methods face an even greater challenge. With multi-table SQL queries with more complex clauses, exhaustive enumeration is intractable in size and prone to mismatches.

More recently, various methods of Text-to-SQL dataset construction have been proposed (Yu et al., 2020; Zhong et al., 2020; Zhang et al., 2021), further automating the SQL-to-NL step with neural question generation. However, for query construction, they either do vanilla grammar-based SQL query sampling (Zhang et al., 2021) or use template sketches from existing datasets (Zhong et al., 2020; Yu et al., 2020). On the other hand, we focus instead on the context-dependent construction of SQL queries that both generalize beyond existing datasets and remain realistic.

$Z ::= Q\ intersect\ Q \mid Q\ union\ Q \mid Q\ except\ Q \mid Q$
$Q ::= Topic\ Filter\ Group\ Select\ Sort$
$Topic ::= PREDEFINED\_TOPIC$
$Filter ::= and\ Filter\ Filter \mid or\ Filter\ Filter \mid$
$> A\ V \mid > A\ Q \mid < A\ V \mid < A\ Q \mid >= A\ V \mid$
$>= A\ Q \mid = A\ V \mid = A\ Q \mid like\ A\ V \mid$
$between\ A\ V \mid not\ like\ A\ V \mid in\ A\ Q \mid$
$not\ in\ A\ Q \mid \epsilon$
$Group ::= Col\_keys\ Filter \mid \epsilon$
$Select ::= A\ Select \mid A$ (with a limit of 5)
$Sort ::= Order\ Col\_keys \mid \epsilon$
$Col\_keys ::= C \mid C\ C$
$Order ::= most \mid least \mid asc\ Limit \mid desc\ Limit$
$A ::= C \mid max\ C \mid min\ C \mid sum\ C \mid avg\ C \mid count\ C \mid$
$distinct\_count\ C \mid count(*)$
$C ::= column$
$V ::= value$
$Limit ::= number \mid \epsilon$

Figure 2: The grammar for generating SQL queries, listed iteratively. *PREDEFINED_TOPIC* is the topic set with method in section 3.1; terminal nodes are in red, non-terminal nodes are in blue.

3 Method

Our pipeline takes database schema as input, and outputs a set of aligned pairs of NL questions and SQL queries. We start by selecting a topic of interest from the schema, followed by sampling production rules from a concise SQL query grammar as in figure 2, resulting in a created SQL query. We then convert the query into a sequence of pseudo-NL questions, and finally crowd-paraphrase the sequence into a fluent NL question.

Specifically, we highlight two key features in our SQL query construction algorithm, asserting control to the process of sampling:

- We set the topic, namely the attended subset of tables in database with an algorithm based on Pagerank and Prim's MST algorithm;
- At each step of sampling, we prune the space of candidates by conditioning the distribution of production rules heuristically on the decision history, namely ancestor nodes, left siblings and left siblings' descendants.

3.1 Setting the Topic

For each valid NL question, there is one topic referring to a concrete concept; similarly, for each real SQL query, however complex, there is one topic, the set of entities it attends to, typically specified in the 'FROM' clause, that should reflect some concrete concepts. For queries involving one table, the topic is simply the table itself; for queries involving multiple tables, which is an iconic feature of SPIDER, it is crucial to identify which tables should be bound together and how.

Our approach here, which is a novel contribution in this paper, takes inspiration from observations in existing datasets. In SPIDER, 93.9% of 'join-on' clauses are between columns with foreign-key relations, an additional 4.2% share the same name[4]. This is consistent with our intuition about SQL language, where join-on clauses are most closely related to foreign-key relations. The popularity of columns sharing the same name apart from foreign-keys is partly a result of missing foreign-key relations and partly a reflection of the fact that columns with the same names are likely relevant.

We therefore set up a table relation graph for each database to model the probabilities that a topic defined by join-on clauses is meaningful. When two columns have a foreign key relation or share the same name, we add an edge between their corresponding tables (foreign-key relations are given higher weights for frequency). Multiple edges between pairs of tables are reduced by sum. To maintain the completeness of grammar space, we assign a small 'background radiation' weight be-

[3] https://yale-lily.github.io/spider

[4] for examples of join-on relations with foreign-key and same-name, please refer to Table 5 in Appendix

tween each pair of tables.

The topic for each SQL query can then be modelled as a sub-graph of this table relation graph, and the transition distribution given previously chosen tables can be modelled with a stochastic version of Prim's MST algorithm (Prim, 1957), formalized as:

$$p(t_k|\Phi) = \frac{\Sigma_{e \in edge(t_k,\Phi)} e.w * e.To.w}{\Sigma_{e \in edge(\Phi^c,\Phi)} e.w * e.To.w} \quad (1)$$

where $\Phi = t_1, ..., t_{k-1}$ is the set of previously chosen tables, $e.To$ is the candidate table, and $e.w$ are edges' weight and $e.To.w$ are Pagerank weights of candidate tables.

With these transition probabilities, the problem has been reduced to choosing the first table. To do this, we need a prior distribution among all tables. Again, we use Pagerank for this purpose: first, each table is assigned an initial importance according to their columns, then we do Pagerank on the table relation graph with random-jump probability of 0.2 to get a context-aware importance among tables, which is then normalized to a distribution. Note that we turn edges to the reverse direction so that weights would accumulate from the primary side to the foreign side of foreign-key relations and the foreign sides would be more likely chosen as the first table, as we would hope.

In sum, as the first step of constructing an SQL query from scratch, we settle its topic. We first sample an 'initial table' from a prior distribution of tables, which is 'Concert' in the case of our example in Figure 1; then we iteratively expand to other tables until halting after a random number of steps[5], which in the case of our example, results in the value of *Topic* row in Figure 1.

3.2 Context-dependent Sampling

After sampling a topic from the table relation graph, we move on to sampling the whole SQL query from a concise SQL query grammar as in Figure 2. We start from a root node Z and recursively dive down until all branches have hit terminal nodes (colored red in Figure 2). An example is shown in Figure 3.

We follow depth-first traversal order, and, to create SQL query sets analogous to the queries in real world, we use decision history as condition of candidate distributions at each step. Namely, we

[5]in practice the maximum number of tables is limited to 4 following statistical observations.

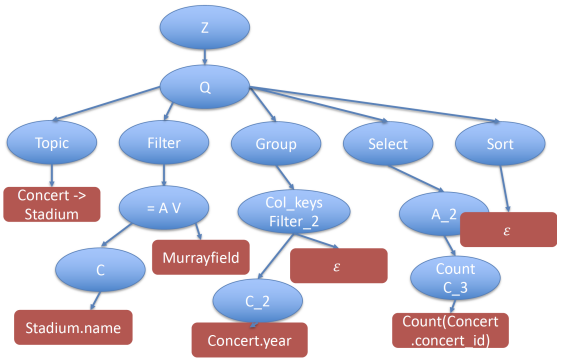

Figure 3: An Example of Created SQL query "Select Count(Concert.concert_id) from Concert join Stadium on Concert.stadium_id = Stadium.id where Stadium.name = 'Murrayfield' group by Concert.year" with tree structure.

assign a larger probability mass to relevant candidates, avoid contradictory or redundant candidates, thereby asserting control to clause structures.

On one hand, we want the resulting SQL queries to make sense in the real world; on the other hand, we don't want their distribution to over-fit to existing domains. Thus, in practice we employ a conservative heuristic approach, set up rules by collecting patterns of 'bad' queries and other domain-agnostic patterns from trials, and prune the space by tuning distributions toward 'good' combinations and against 'bad' ones.

For example, the following rule "A column is more likely chosen to 'where' clause if it has been chosen in the last 'where' clause", tunes probabilities against queries like *select editors' names from journal committees and their corresponding journals and editors, whose age is smaller than 50 and journal's sales is larger than 1600*, in favour of those like *select editors' names whose age is larger than 40 and smaller than 50*.

Additionally, to reduce redundancy, we validate all candidate clauses at each step by executing them against the databases and collecting responses. We compare query response before and after adding a candidate clause, and screen out clauses that either make no difference or result in empty responses. We present the full set of rules in Appendix B.

3.3 From SQL to Pseudo-NL

Following previous work, to translate SQL queries to NL questions, we first use a template-based approach to convert them to pseudo-NL questions. Similarly to Cheng et al. (2018), we deterministically convert complex SQL queries into sequences

of pseudo-NL questions to make annotation easier. In practice, with the more complex SQL clause structures, we find it not ideal to split questions into sequences as granular as Cheng et al. (2018), because annotators again get lost in the labyrinth of coreferences between questions in the same sequence. Thus we re-balance the trade-off between the number of sentences and individual complexity towards longer but fewer sentences, so it's not too hard for crowd workers to follow[6].

Notably, while generally speaking SQL language looks similar to natural language, its FROM clauses with table-joining are very unnatural, and when involving many tables, can make their literal translations impenetrable. Unlike in NL questions where there is an integrated topic, in SQL language the topic defined by 'FROM' clause could be long and confusing. Take the example in Figure 1, its topic *'Concerts and Stadiums'* in the form of SQL query becomes *'Concert join Stadium on Concert.stadium_id = Stadium.id'*. Worse still, multiple tables also make the meaning of wildcard column '*' confusing in clauses such as *'select count(*)'*.

Luckily, we have observed that for a pair of tables joined by foreign key relations, we can always consider the foreign side as the 'main' table of the pair, since it is the one from which the primary side is extended. Therefore, we define this directionality for a topic sub-graph of the table relation graph: primary $\rightarrow$ foreign as root-wise and foreign $\rightarrow$ primary as leaf-wise; for table pairs linked by same-name relation, an edge is kept on both directions.

Then, for multi-table queries, we assume that the table(s) at the root-most position is the "dominant concept" of the topic. Since sub-graphs are predominantly trees, mostly there is one dominant concept for each query. Whenever possible, we replace all pseudo-NL phrases for the table joining, such as the above, with expressions like *'Concerts and their corresponding Stadiums'*, and replace all phrases for wildcard column *'count(*)'* with expressions like *'the number of Concerts'*. This way pseudo-NL questions are simplified, and the annotation burden is eased.

3.4 From Pseudo-NL to NL

The last part of our pipeline involves crowd-paraphrasing these pseudo-NL question sequences into fluent NL questions. We recruit workers on the Amazon Mechanical Turk (AMT) platform, present tasks to AMT workers randomly and pay $0.25 for each task completed.

In each task, we present the workers with a pseudo-NL question sequence paired with examples from DB response. In pilot trials, we found that annotators tend to keep fragments from the pseudo-NL questions even when they're clearly rigid. We hypothesize that this is an exposure effect, that annotators' exposure to the pseudo-NL questions influenced their own style of expression.

As another of our novel contributions, we pose a countermeasure to this exposure effect. First, we engage annotators in the context of helping their foreign friends sound local. Further, we present a personalized example specific to each generated SQL query. These personalized examples are taken from expert annotated datasets in other domains, but can give crowd workers a general idea of what level of naturalness is expected and in which way.

Each personalized example involves a pseudo-NL question sequence, an expert-annotated NL question and an example DB response. To provide the most relevant hint, we retrieve from existing data the most similar entries to each created SQL query, where similarity is measured as the cosine similarity regarding an engineered feature vector[7]. We retrieve the top 10 example queries with smallest distances in the above terms. We then randomly pick one as the personalized example to display.

We employed only the English speaking AMT workers with 95%+ acceptance rate and 50+ acceptance history to restrict this paraphrasing task to a set of competent workers. However, empirically we still found a considerable number of workers submitting nonsensical paraphrases, apparently not understanding the task or giving up on the complex input. Thus, we restricted the access to only the trusted workers who had previously performed well in our task.

4 Our Experimental Corpus

4.1 Corpus Construction Details

We experiment on the basis of the SPIDER dataset, with data split details described in Table 1. Because the databases and Text-SQL pairs for SPIDER test set domains are not released, we re-split the 20 SPIDER development (dev) set domains equally

[6]For a flowchart with details please see Figure 4 in Appendix.

[7]For details of feature vector please refer to Appendix C.

Data Split	Domains	SQL Construction	SQL-to-NL	Data Size
seen-domain-train-set	166 train domains	Gold	Gold	7000
novel-domain-test-set	10 novel dev domains	Gold	Gold	440
seen-domain-dev-set	10 seen dev domains	Gold	Gold	594
novel-domain-ours-set	**10 novel dev domains**	**Ours**	**Ours**	**543**
novel-domain-oracle-set	**10 novel dev domains**	**Gold**	**Ours**	**466**
seen-domain-small-set	166 train domains	Gold	Gold	543
Zhang et al. (2021)	20 dev domains	-	-	58691

Table 1: Data splits involved in our experiment. **Ours** SQL construction refers to method in section 3.1, 3.2, while **Ours** SQL-to-NL refers to 3.3, 3.4. Zhang et al. (2021) is a SOTA data augmentation system described in 5.1.

into **seen** domains and **novel** domains[8]. Accordingly, we define the seen-domain-dev-set and novel-domain-test-set from the SPIDER dev set, along with the SPIDER train set renamed seen-domain-train-set.

Additionally, we collect two data sets in the novel domains, novel-domain-ours-set and novel-domain-oracle-set. Novel-domain-ours-set is our target dataset with entries constructed from scratch with only the databases and our full pipeline. Novel-domain-oracle-set entries start from the novel domain gold SQL queries, annotated with our SQL-to-NL method. Moreover, we create a new data split called seen-domain-small-set, which has the same size as novel-domain-ours-set, but is randomly sampled from the expert-annotated seen-domain-train-set.

We use a linear regression to derive the number of SQL queries to construct for each database, w.r.t the number of TABLES, COLUMNS and FOREIGN-KEY relations. Each created SQL query is paraphrased into natural language by 2 annotators as in SPIDER, paraphrases too short or too long are filtered out. The resulting data sizes are illustrated in Table 1.

In total, the paraphrasing and human evaluation (to be elaborated below) cost us $349.34.

4.2 Human Evaluation

To test the effectiveness of our SQL query construction and SQL-to-NL conversion method respectively, we conduct two experiments of human evaluation with participants recruited on AMT.

To evaluate the constructed SQL queries, we use the corresponding computer-generated pseudo-NL as a proxy for SQL queries to involve crowd-sourcing. We present each participant with a Pseudo-NL question, ask them to indicate whether the Pseudo-NL is succinct, sensible, relevant and complex. Each question is randomly chosen either from our created novel-domain-ours-set or from expert-annotated SQL queries in novel-domain-oracle-set, where the choice is hidden from participant workers. We evaluate on all entries in the 10 novel domains, with results presented in Table 2.

Queries considered having following properties (in %)	ours	expert
Succinct	75.51	74.46
Sensible	95.95	89.49
Relevant	83.06	76.82
Complex	55.80	37.98

Table 2: Human evaluation results between ours and expert SQL queries.

As shown, compared to expert-annotated ones, participants considered a larger proportion of our queries complex, but also a larger proportion concise, sensible and relevant. Although human evaluation scores are subjective and could fluctuate across individuals, this at least shows that from the annotators' point of view, our created SQL queries are comparable to expert annotated ones on some dimensions.

To evaluate our SQL-to-NL conversion method, we compare our crowd-sourced questions from novel-domain-oracle-set with expert annotated NL questions from novel-domain-test-set. Since both are aligned to the same set of gold SQL queries, we show participants pairs of NL questions referring to the same SQL query. We ask the participants how similar the question pairs are, and which of them is smoother in language.

With crowd workers as judges, we cannot di-

[8]The 10 selected novel domains are: *orchestra, singer, real_estate_properties, tvshow, battle_death, voter_1, student_transcripts_tracking, concert_singer, world_1, and course_teach.*

rectly measure how rigorous our NL questions are in preserving the semantics of SQL queries. However, by taking expert-annotated questions as the gold standard of query semantics, and inspecting the similarity between ours and expert-annotated questions for the same gold-standard SQL queries, we can indirectly measure the level of rigorousness that our conversion method is capable of.

Results show that the average similarity between our questions and gold questions is scored 3.78 out of 1 to 5, indicating a good alignment between the meanings of the question pairs. As for preference, 45% of times our question is preferred, while 39% of times gold question is preferred and 16% of times the two are considered equally smooth in expression. This result verifies the validity of our SQL-to-NL conversion method.

5 Evaluation by Domain Transfer

5.1 Evaluation Setting

To further evaluate our pipeline's effectiveness, we do an extrinsic evaluation in the context of domain transfer. Namely, we test the capability of our created data to help semantic parsers generalize to novel domains. Below we first define 4 dataset settings and 2 training scenarios:

Dataset Settings

- PRETRAIN Trains on seen-domain-train-set[9], validates on seen-domain-dev-set, tests on novel-domain-test-set. This reflects the process of training a model on seen domains then applying the trained model to novel domains;
- OURS Trains on novel-domain-ours-set, tests on novel-domain-test-set. This is our target setting, which reflects training with our semi-automatically constructed in-domain data then test on real-world queries in the same domains;
- GOLD Trains on half (220 entries) of the novel-domain-test-set, tests on the other half. This setting approximates a theoretical upper bound, reflecting how much accuracy gain can be achieved with gold in-domain data;
- TRAIN(SMALL) Trains on novel-domain-small-set, tests on novel-domain-test-set. This setting is an out-of-domain expert-annotated baseline for OURS setting in RANDOM-INIT scenario, and answers the question "how powerful is our created, in-domain data compared to the expert-annotated but out-of-domain data, in terms of training models from random initialization".

Training Scenarios

- RANDOM-INIT Start training from randomly initialized model parameters. For models with BERT, initialize BERT parameters with pretrained checkpoints[10];
- FINETUNE Start training from model checkpoints acquired from RANDOM-INIT training on PRETRAIN data.

We conducted experiments with 3 popular recent Text-to-SQL semantic parsers: lang2logic (Dong and Lapata, 2016), IRNet-BERT (Guo et al., 2019), and RAT-SQL-BERT (Wang et al., 2020). lang2logic is the first semantic parser to employ Seq2Seq paradigm, IRNet-BERT is the first practically effective semantic parser on SPIDER challenge, and RAT-SQL-BERT is the latest reproducible SOTA when using BERT-Large encoder. The lang2logic models were originally written in Lua and are re-implemented with PyTorch; to serve as a vanilla baseline, seq2seq setting is used instead of the more complex seq2tree setting. For RAT-SQL-BERT models, due to memory limits of our 1080 TI GPUs and for a fair comparison with IRNet-BERT, we use the bert-base-uncased version as in IRNet-BERT, instead of the bert-large-whole-word-masking version in the original implementation.

For each parser, we first train them under the RANDOM-INIT scenario with PRETRAIN data; then FINETUNE the pretrained model saperately, with OURS and GOLD data. Additionally, we train each parser under RANDOM-INIT scenario with OURS and TRAIN(SMALL) data respectively, to evaluate our data outside the scope of domain transfer.

For all RANDOM-INIT models we use the same set of hyper-parameters as in their original settings; for FINETUNE models, following the intuition that fine-tuning should have learning rates no larger than pretrain, we do log-uniform sampling through 1/3, 1/10, 1/30 and 1/100 of the original learning rates as well as the original learning rates themselves; for models with BERT encoders, we further grid-search both having the BERT learning rates fixed and aligned with other parameters.

We have attempted to compare with previous work on semi-automatic dataset construction. How-

[9] For definitions of data splits, please refer to Table 1

[10] https://huggingface.co/transformers/pretrained_models.html

Accuracy (%)	lang2logic	IRNet-BERT	RAT-SQL-BERT
pretrain	3.87	65.60	59.77
finetune-ours	**5.00***	**66.74**†	60.00$^\diamond$
finetune-gold	8.06	71.07	61.13
random-init-ours	1.14	21.79	19.19
random-init-train(small)	1.82	42.66	14.55

Table 3: Exact Match accuracy in percentage points. $\star$ means at learning rate of 1e-4, † means at 3e-4, $\diamond$ means at 1e-5. To faithfully reproduce a domain transfer setting, the accuracies of PRETRAIN models are reported on novel-domain-test-set. Therefore, reported accuracies may vary from the ones reported on the development set of SPIDER.

ever, the method of Wang et al. (2015) is restricted to LambdaDCS logical forms and not applicable to our setting of multi-table SQL queries; the data construction method of Cheng et al. (2018) is not contained in their open-source codebase, and the first author was unfortunately not reachable for that implementation.

Nonetheless, we discuss Zhang et al. (2021) for comparison, a recent work on large-scale Text-to-SQL data augmentation, with context-free sampling of SQL queries and hierarchical neural automatic NL generation without human intervention.

5.2 Results and Discussions

Evaluation results are shown in Table 3. Accuracies are evaluated on the novel-domain-test-set of SPIDER. As shown, extra in-domain data collected with our pipeline improves novel domain accuracy by more than 1% for both lang2logic and IRNet-BERT models.

We also tried applying the same fine-tuning to the IRNet and RAT-SQL parsers without BERT. However, that did not increase the accuracy. We attribute this difference to the fact that while our additional data provides information on the novel domain, its language style is not the same as SPIDER train set. Our crowd-paraphrased questions are bound to be different in style to expert-annotated questions. The conventional recurrent encoders, trained only on the SPIDER train set, fail to capture the meaning of questions in our fine-tune dataset. On the other hand, with BERT contextualizers, which had been trained on texts at the magnitude of billions, the language of our questions looks more familiar to models, and they can more successfully absorb the domain-related information encoded in our data. As BERT-Large models are bigger and more powerful, we would expect the accuracy gain to be larger with RAT-SQL-BERT-Large, and expect the same trend for other semantic parsers in general.

The finetune-gold result provides an approximate upper bound, from expert-annotated data pairs at the same magnitude of our size. It is encouraging that we are able to correct roughly 20% of the correctable errors by this standard. Further gains could reasonably be expected from increasing scale.

Comparisons between models trained from RANDOM-INIT scenario also show interesting findings. Since with both *random-init-ours* and *random-init-train(small)*, the data sizes are a magnitude smaller than the SPIDER training set used for PRETRAIN, parsers trained under these two settings perform less competitively than their FINETUNE counterparts. But among themselves, training from RANDOM-INIT with OURS data is generally comparable to that of expert-annotated TRAIN(SMALL) setting of the same size, and in the case of RAT-SQL-BERT it even exceeds that of TRAIN(SMALL).

The results in Table 3 are for the best fine-tuned model checkpoint on our test data, the novel-domain-test-set. This choice reflects the scenario of our chosen task: we start with novel domains, having only the databases available for use. We semi-automatically create an in-domain fine-tune dataset and train semantic parsers with it. We envisage deploying the fine-tuned models to arrive at the best model checkpoint by the use of feedback from users, for which optimizing on the novel-domain-test-set is used as a proxy.

We are nevertheless interested in knowing if we can choose the best model checkpoint independently of our test. In terms of our hypothetical scenario, it reflects whether we can achieve good performance with real queries, by assuming access to a fixed development set, and choosing the best model checkpoint according to performance on that fixed set.

The standard practice here is to validate on a

set-aside development set, then report the accuracy on the test set. However, the 440 entries of novel-domain-test-set is all the expert-annotated data that we have for our novel domains. In response to this dilemma, we split the novel-domain-test-set in half, into two subsets A and B. We then do cross-validation between them: we use one subset for picking the model checkpoint with highest accuracy, assessing its accuracy on the other (and vice versa). Under this setting, we still achieve an accuracy gain of 0.74% with IRNet-BERT and 1.25% with lang2logic.

Contemporary with our work, Zhang et al. (2021) generated a much larger set of 50,000+ synthetic training data (which they were able to do by not involving human judges). Under the same evaluation strategy as us in Table 3, and starting from a different IRNet-BERT pretrained baseline of 59.5%[11], they report an augmented accuracy of 61.7% on the SPIDER development set, obtaining an increase of 2.2%. However, we achieve half of that increase with only around 1% their amount of data. In the absence of access to their code, we have been unable to determine the performance that Zhang et al. (2021) would obtain from a comparably small dataset, but we feel confident that gains comparable to their full dataset could be obtained from our method with more modest increases in scale.

6 Conclusion

We have presented a novel approach to construct in-domain Text-to-SQL data, following the three-step paradigm of Wang et al. (2015). We randomly select a topic of interest from a table relation graph prior to building the actual query, and sample query clauses in a context-dependent manner. We identify "dominant concept" of the topics to simplify the converted Pseudo-NL, and retrieve personalised examples as annotation scaffold for crowd-paraphrasing. Our experiments show that our in-domain data is comparable with expert-annotated data, and capable of increasing the accuracy of SOTA IRNet-BERT semantic parser by up to 1%.

For future work, we plan to explore more sophisticated probabilistic models to control SQL query construction, and pair our query construction method with recent work on SQL-to-NL translation, so as to bring our method to larger scale.

[11] which is mainly due to their use of SPIDER development set and our use of novel-domain-test-set.

Acknowledgements

We thank Yantao Jia and Jeff Pan for helpful comments and feedbacks, we'd also like to thank the three anonymous reviewers for their valuable comments. This work was supported partly by a Mozilla PhD scholarship at Informatics Graduate School, by the University of Edinburgh Huawei Laboratory, and by National Natural Science Foundation of China (No. 61876009).

References

Jonathan Berant, A. Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *EMNLP*.

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

Shuaichen Chang, Pengfei Liu, Yun Tang, Jing Huang, Xiaodong He, and Bowen Zhou. 2020. Zero-shot Text-to-SQL learning with auxiliary task. In *AAAI*.

Jianpeng Cheng, Siva Reddy, and Mirella Lapata. 2018. Building a neural semantic parser from a domain ontology. *CoRR*, abs/1812.10037.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33–43, Berlin, Germany. Association for Computational Linguistics.

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

Pengcheng He, Yi Mao, Kaushik Chakrabarti, and Weizhu Chen. 2019. X-SQL: reinforce schema representation with context. *arXiv preprint arXiv:1908.08113*.

Jonathan Herzig and Jonathan Berant. 2019. Don't paraphrase, detect! rapid and effective data collection for semantic parsing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3810–3820, Hong Kong, China. Association for Computational Linguistics.

Wonseok Hwang, Jinyeong Yim, Seunghyun Park, and Minjoon Seo. 2019. A comprehensive exploration on WikiSQL with table-aware word contextualization. *arXiv preprint arXiv:1902.01069*.

Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke Zettlemoyer. 2013. Scaling semantic parsers with on-the-fly ontology matching. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1545–1556, Seattle, Washington, USA. Association for Computational Linguistics.

Percy Liang. 2013. Lambda dependency-based compositional semantics. *CoRR*, abs/1309.4408.

Qin Lyu, Kaushik Chakrabarti, Shobhit Hathi, Souvik Kundu, Jianwen Zhang, and Zheng Chen. 2020. Hybrid ranking network for Text-to-SQL. Technical Report MSR-TR-2020-7, Microsoft Dynamics 365 AI.

Robert Clay Prim. 1957. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401.

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

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, Online. Association for Computational Linguistics.

Chenglong Wang, Kedar Tatwawadi, Marc Brockschmidt, Po-Sen Huang, Yi Mao, Oleksandr Polozov, and Rishabh Singh. 2018. Robust text-to-sql generation with execution-guided decoding. *arXiv preprint arXiv:1807.03100*.

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

Xuchen Yao and Benjamin Van Durme. 2014. Information extraction over structured data: Question answering with Freebase. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 956–966, Baltimore, Maryland. Association for Computational Linguistics.

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, Richard Socher, and Caiming Xiong. 2020. Grappa: Grammar-augmented pre-training for table semantic parsing.

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

Ao Zhang, Kun Wu, Lijie Wang, Zhenghua Li, Xinyan Xiao, Hua Wu, Min Zhang, and Haifeng Wang. 2021. Data augmentation with hierarchical SQL-to-question generation for cross-domain Text-to-SQL parsing.

Rui Zhang, Tao Yu, Heyang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019. Editing-based SQL query generation for cross-domain context-dependent questions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5338–5349, Hong Kong, China. Association for Computational Linguistics.

Victor Zhong, Mike Lewis, Sida I. Wang, and Luke Zettlemoyer. 2020. Grounded adaptation for zero-shot executable semantic parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6869–6882, Online. Association for Computational Linguistics.

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

A Cross-Domain Differences

The expression for query semantics can vary significantly across domains, SQL queries of similar structure can be mapped to fundamentally different questions.

For example, the two SQL queries in Table 4 look similar in their sketch, differing only in their schema tokens. However, their corresponding questions, when expressed in natural English, are very different even from the structures. For instance, the similar 'where' clause is a modifier of the subject 'brand' in one domain, and an adjunct of the predicate 'held' in the other.

B Set of Major Heuristic Rules

- Every table in topic is involved in at least one clause

Domain:	Car Makers
SQL:	Select Brand.car_maker, Count(Brands.brand_id) from Brands JOIN Country ON Brands.country_id = Country.country_id where Country. name = 'Germany' group by Brand.car_maker
Question:	How many German brands does each car maker own?

Domain:	Singers and Concerts
SQL:	Select Concert.year, Count(Concert.concert_id) from Concert JOIN Stadium ON Concert.stadium_id = Stadium.id where Stadium.name = 'Murrayfield' group by Concert.year
Question:	How many concerts were held in Murrayfield each year?

Table 4: An example pair of SQL queries with similar structures in different domains, the corresponding natural language questions that map to them have very different structures.

- A column is more likely chosen to 'WHERE' clause if it has been chosen in the previous 'WHERE' clause
- A sub-query nested in 'WHERE' clause likely returns the same column as the subject column in that 'WHERE' clause or is related to it via foreign-key relation
- Columns present in equality conditions are likely not chosen in 'GROUP BY' clauses.
- Subject columns in 'GROUP BY' clauses are likely to be selected.
- 'GROUP BY' clauses always take effect either by aggregating 'SELECT' columns, 'HAVING' conditions or 'ORDER BY' clauses.
- Columns present in 'SELECT' are likely also present in 'ORDER BY'
- When two queries are linked together via an 'UNION', 'EXCEPT' or 'INTERSECT', it is likely that the two queries share similar structure, only with one or two different structures such as 'WHERE' conditions.

Example: Foreign key
Select avg(T1.killed) from perpetrator as T1 join people as T2 on T1.people_id == T2.id where T2.height $<$ 1.8m
How many people were killed by perpetrators shorter than 1.8m on average?
Example: Same Name
SELECT T1.id FROM trip AS T1 JOIN weather AS T2 ON T1.zip_code = T2.zip_code GROUP BY T2.zip_code HAVING avg(T2.mean_temperature_f) $>$ 60
Give me ids for all the trip that took place in a zip code area with average mean temperature above 60.

Table 5: Examples of Foreign-Key and Same-Name conditions, with SQL queries in the first line and paired questions in the second.

C Feature Vector for Personalized Examples

In retrieving the personalized examples, the feature vectors for measuring similarity between SQL queries involve the following feature values. Features related to "SELECT", "FROM", "WHERE", "GROUP BY", "ORDER BY" and Set Operation clauses are listed in Table 6, 7, 8, 9, 10, 11 respectively.

Feature about "SELECT"	Weight
Number of "SELECT" clauses	1.0
Wildcard "*" in column	4.0
MAX in column	2.0
MIN in column	2.0
COUNT in column	2.0
SUM in column	2.0
AVG in column	2.0

Table 6: Feature vector values regarding the column clauses, with weights specified to the right of each feature.

Feature about "FROM"	Weight
Number of tables in "FROM"	1.0
All tables connected by "JOIN ON"	2.0

Table 7: Feature vector values regarding the "FROM" clauses, with weights specified to the right of each feature.

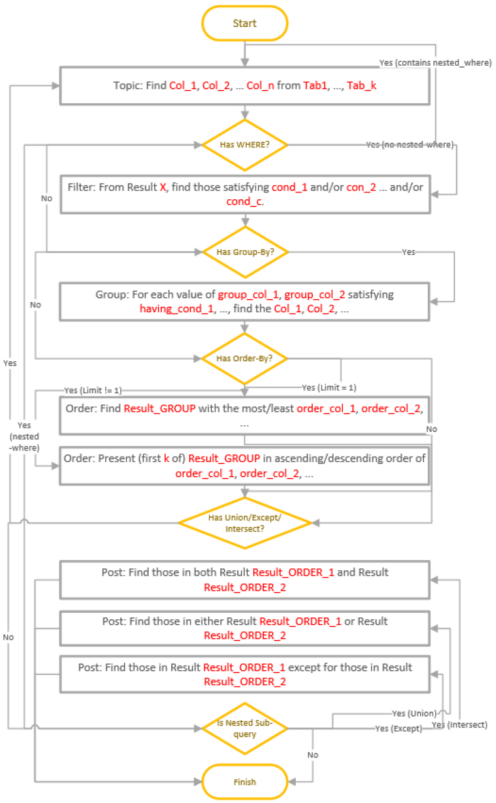

Figure 4: Flowchart illustration of how SQL queries are split into sequences of computer-generated questions. Triangle boxes and connection lines indicate the sequential relationship between these templates, in rectangle boxes are individual templates. In these templates, colored red are the slots to fill in. To build a sequence of Pseudo-NL questions, a program walks through the chart from start to finish, and lists the resulting instantiated templates iteratively as output.

Feature about "WHERE"	Weight
Empty "WHERE" clause	2.0
Number of "WHERE" clauses	1.0
"*" in column	4.0
MAX in column	1.0
MIN in column	1.0
COUNT in column	1.0
SUM in column	1.0
AVG in column	1.0
Sub-query in clauses	4.0
Column-valued clauses	4.0
'between' as operator	1.0
'equal' as operator	1.0
'larger than' as operator	1.0
'smaller than' as operator	1.0
'not larger than' as operator	1.0
'not smaller than' as operator	1.0
'not equal' as operator	1.0
'in' as operator	1.0
'like' as operator	1.0

Table 8: Feature vector values regarding the "WHERE" clauses, with weights specified to the right of each feature.

Feature about "GROUP BY"	Weight
Number of "GROUP BY" clauses	1.0
Involves "HAVING" clause	2.0
Involves "HAVING" with Sub-query	2.0

Table 9: Feature vector values regarding the "GROUP BY" clauses, with weights specified to the right of each feature.

Feature about "ORDER BY"	Weight
Number of "ORDER BY" clauses	1.0
Ascending / Descending order	1.0
Involves "LIMIT" clause	1.0
Involves "LIMIT 1"	2.0

Table 10: Feature vector values regarding the "ORDER BY" clauses, with weights specified to the right of each feature.

Feature about Set Operations	Weight
Involves "UNION" clause	4.0
Involves "EXCEPT" clause	4.0
Involves "INTERSECT" clause	4.0

Table 11: Feature vector values regarding the Set Operation clauses, with weights specified to the right of each feature.