Importance of Synthesizing High-quality Data for Text-to-SQL Parsing

Yiqun Hu, Yiyun Zhao*, Jiarong Jiang, Wuwei Lan, Henry Zhu, Anuj Chauhan, Alexander Li, Lin Pan, Jun Wang, Chung-Wei Hang, Sheng Zhang, Jiang Guo, Marvin Dong, Joe Lilien, Patrick Ng, Zhiguo Wang, Vittorio Castelli, Bing Xiang
AWS AI Labs
*yiyunzhao@arizona.edu
{yiqunhu, jiarongj, lanwuwei, henghui, chaanj, hanboli, linpan, juwanga, cwhang, zshe, gujiang, mingwd, lilienj, patricng, zhiguow, vittorca, bxiang}@amazon.com

Abstract

There has been increasing interest in synthesizing data to improve downstream text-to-SQL tasks. In this paper, we examined the existing synthesized datasets and discovered that state-of-the-art text-to-SQL algorithms did not further improve on popular benchmarks when trained with augmented synthetic data. We observed three shortcomings: illogical synthetic SQL queries from independent column sampling, arbitrary table joins, and language gaps between the synthesized SQL and natural language question (NLQ) pair. To address these issues, we propose a novel synthesis framework that imposes strong typing constraints, incorporates key relationships from schema, and conducts schema-distance-weighted column sampling. We also adopt an intermediate representation (IR) for the SQL-to-text task to further improve the quality of the generated NLQ. When existing powerful text-to-SQL parsers are pretrained on our high-quality synthesized data, these models have significant accuracy boosts and achieve new state-of-the-art performance on Spider. We also demonstrate the effectiveness of our techniques with ablation studies.

1 Introduction

Text-to-SQL parsing refers to the semantic parsing task that translates a natural language question (NLQ) to a corresponding SQL query. In recent decades, many industries have adopted high-level digitalization in their workflow and possessed large-scale datasets—many of which are stored as relational databases. Extracting insights from these relational databases to further drive business decisions is an important task. However, due to the complexity of these relational databases, query language experts are often needed to extract valuable insights. Thus a high-performing text-to-SQL system with a natural language interface would greatly lower the barrier for business users to query their databases.

In order to obtain high-quality training data for the text-to-SQL parser, human annotators with SQL expertise are needed to construct NLQ-SQL parallel data, which are difficult and expensive to scale. Thus data scarcity is a well-known bottleneck in the text-to-SQL task (Yu et al., 2018b). To address the data scarcity issue, there is an increasing interest in leveraging synthetic data to improve the downstream performance. Yu et al. (2021) handcrafted high-quality rules to synthesize SQL and NLQ simultaneously, but these grammar rules need to be carefully designed through expensive manual work. To automate the synthesis process, recent attempts (Wang et al., 2021; Wu et al., 2021; Shi et al., 2021; Zhong et al., 2020) utilize a two-stage approach that synthesizes SQL first and then composes NLQ with a SQL-to-text generator. Alternatively, Yang et al. (2021) proposed a reversed pipeline that uses an entity-to-question model to generate natural language queries and then a text-to-SQL parser to generate SQL queries.

In this paper, we delve into the two-stage synthesizing method that first synthesizes SQL queries and then generates NLQs. We first provide a comprehensive literature review and evaluate the contribution of existing synthetic datasets for the text-to-SQL task on popular benchmarks, such as Spider (Yu et al., 2018b). Results show that they only have negligible impact on boosting recent parsing models accuracy (e.g. T5 + PICARD proposed by Scholak et al. (2021)) when being augmented to the original training set. We analyze the quality of these synthetic datasets and find they suffer from three major issues, including illogical synthetic SQL queries from independent column sampling, arbitrary table joins, and language gaps between the synthesized SQL and NLQ pairs. We then propose a novel framework[1] aiming to reduce

*Work done during an internship at AWS AI Labs.

[1]Source code will be made publicly available.

Findings of the Association for Computational Linguistics: ACL 2023, pages 1327–1343

these problems present in existing methods. During the stage of SQL synthesis, we employ template synthesis with strong typing constraints, template key relationship preservation, and schema-distance-weighted column sampling. As for the SQL-to-text generation step, we adopt an intermediate representation (IR) to reduce the gap between SQL and target NLQ. We show that the top-performing text-to-SQL parsers can have significant accuracy improvements when being pretrained using our high-quality synthesized data and achieve new state-of-the-art performance on Spider.

In summary, our main contributions are:

- We systematically compare the existing text-to-SQL synthesis methods and identify three shortcomings that lead to the low quality;
- we propose three novel techniques for generating synthetic data and demonstrate its augmentation benefits for various text-to-SQL parers, underscoring the importance of high synthesis quality;
- we adopt an intermediate representation (IR) for the SQL-to-text task, which further improves the quality of the NLQ generation.

2 Existing Synthesis Methods and Limitations

We first conduct a detailed investigation towards the existing text-to-SQL synthesis frameworks to understand each of their advantages and shortcomings, the details of which can be found in Appendix A. In particular, Figure 5 summarizes and compares the key characteristics from different dimensions.

In this section, we experiment with two recent synthetic datasets (Wang et al., 2021) and (Wu et al., 2021), and leverage the latest state-of-the-art text-to-SQL model T5 + PICARD (Scholak et al., 2021) to assess their effectiveness. We find that they only bring negligible impact on the performance when being used to train the parsing model. We then discuss three main shortcomings in these synthetic datasets based our manual inspection and analysis.

2.1 Synthetic Data Effectiveness Assessment

As a pilot study, we use T5-Large + PICARD as the text-to-SQL parser to examine the synthetic data quality. The baseline model is trained on Spider training set only. To add synthetic data during training, we setup a two-stage process. In *Stage 1*, the model is (pre-)trained using only the synthetic data. In *Stage 2*, we finetune the model with the original Spider training set (also see *Training Configurations* in Section 4.1).

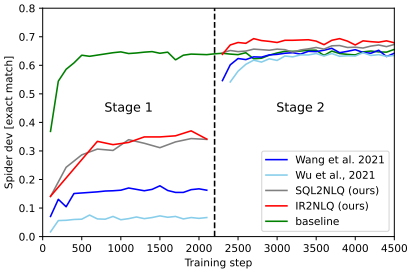

Figure 1: Comparison of different synthetic data on Spider dev set performance. The baseline model is trained using Spider data only. Previous synthetic datasets (Wu et al., 2021; Wang et al., 2021) fail to bring additional performance boost to the model. Our synthetic data (SQL2NLQ and IR2NLQ) yield better results on both stages. (Stage 1 is training with synthetic data only and Stage 2 is finetuning with real Spider training set.)

As shown in Figure 1, the exact match (EM) accuracy of the models trained on both synthetic datasets (blue and light blue curves) are less than 20% during Stage 1, in contrast to 60% (green curve) when trained with real Spider data. This gap indicates the limited transferability from existing synthetic data to real data. Further finetuning on Spider training data in Stage 2 does not outperform the baseline model which is only trained using Spider data, indicating the ineffectiveness of these synthetic datasets. However, our synthetic data (red and gray curves for IR2NLQ and SQL2NLQ[2]) can still yield better performance for both two stages. We will discuss in detail our methodologies in Section 3.

2.2 Synthetic Data Quality Analysis

We now provide an analysis towards the previous synthesis methodologies and identify three probable causes for obsolescence.

2.2.1 Illogical Synthetic SQLs from Invalid Grammars or Templates

Both Wang et al. (2021) and Wu et al. (2021) adopted context-free grammars (CFG) to generate SQL queries. The CFG designed by Wu et al. (2021) is constrained and they limited SQL generation to one table. While Wang et al. (2021) designed flexible grammars, they neglected

[2]SQL2NLQ and IR2NLQ referring to two different SQL-to-text models we use during NLQ synthesis, more discussion in Section 4.1 and 4.2

the constraints between operators and column types. This negligence leads to mistakes such as `SUM(student.name)`, where an aggregation operator is applied to a text column.

Furthermore, CFG or probabilistic CFG (PCFG) generated SQL queries often fail to capture foreign-key and key relations between columns. This leads to invalid SQLs such as `SELECT name, age FROM student INTERSECT SELECT address FROM teacher`, where it intersects two sub-queries with different number of columns. In fact, designing a grammar to produce high coverage and logical SQLs is a difficult task due to the implicit dependencies of SQL elements.

Alternatively, SQL templates extracted from training data better preserves column typing information (Zhong et al., 2020). This approach drastically reduces the invalid SQLs caused by a misalignment between operators and column types. However, existing work still misses the critical key relations in the templates.

2.2.2 Over-Complex SQLs from Arbitrary Multi-table Joins

When SQLs are materialized, the column/table selection from existing work is independent and results in SQL queries with unnecessary complexity. Those queries often have unclear intents and thus are difficult to be correctly translated to natural language questions. An example is presented in Table 2, where a simple template that requires only two columns can be turned into a complicated and nonsensical SQL query with three table joins.

2.2.3 Language Gap between SQL and NLQ

Recent work typically trains a sequence-to-sequence model to obtain corresponding NLQs from synthetic SQLs (Wang et al., 2021; Shi et al., 2021). The gap between SQL-NLQ pairs are well recognized in text-to-SQL task and intermediate representation (IR) is commonly used to reduce such mismatch (Gan et al., 2021b; Guo et al., 2019a; Yu et al., 2018a; Shi et al., 2021). However, the reverse of the source and target in SQL-to-text brings in its own challenge, such as incorrect references for `SELECT *`, missing conditions within long and complex SQL queries, and misinterpretation of ORDER phrases.

3 Proposed Method

Our synthesis framework builds on top of the template-based SQL synthesis approach similar to

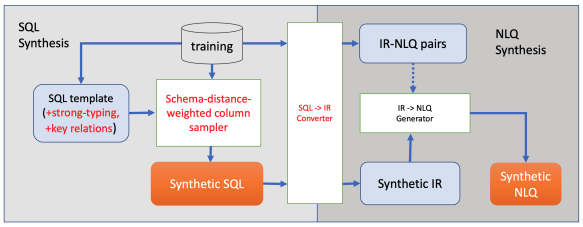

Figure 2: Our NLQ-SQL synthesis framework. Novel components include strong-typing, key relations, schema-distance-weighted column sampler, and SQL → IR converter.

Zhong et al. (2020); Zhang et al. (2019) and generates SQL-NLQ pairs with a sequence-to-sequence model. Figure 2 provides a general overview of our pipeline. We develop the following techniques to address the issues in existing synthesis methods discussed in the previous sections:

- for SQL synthesis:
  - introducing strong typing for columns *w.r.t.* semantic types and key properties,
  - encoding the key relation with the extracted templates for more logical SQLs,
  - proposing a schema-distance-weighted column sampling strategy to avoid over-complex joins,
- for NLQ synthesis:
  - designing an improved IR to bridge the gap between SQL and NLQ specifically for the SQL-to-text task.

3.1 SQL Synthesis

To create new SQLs on training data schemas, we utilize a template-based approach following Zhong et al. (2020): First, a pool of SQL templates are created by normalizing the schema-related mentions (column and value) and removing JOIN phrases. During SQL generation, a template is sampled based on the training distribution, and columns are sampled with constraints to fill in the normalized slots of the template. We highlight several improvements made to the existing approaches.

3.1.1 Strong Typing

When normalizing columns, we enforce strong typing of a template by enriching and preserving the data type (e.g., text, number, date, etc) as well as key identity (key or not) for each column. For example, in Table 1, we use `textkey` instead of `key` to normalize `artist_name` because operators such as `MAX` can be applied to number key but usually not to other text key.

SQL	`SELECT artist_name FROM song INTERSECT SELECT artist_name FROM artist`
Previous	`SELECT col1_key INTERSECT col2_key`
Ours	`SELECT col1_textkey INTERSECT col2_textkey_fk1`

Table 1: Our modifications for template extraction: strong typing is highlighted in blue and key relation preservation is highlighted in pink.

3.1.2 Template Key Relationship Preservation

A foreign key is a column in a table referring to the primary key (unique identifier) of another table. In multiple table join scenarios, key and foreign key are the most common columns to be joined on. Restricting a column to be a foreign key to another key column is critical for a SQL to be valid especially in the following two cases: 1) queries including `INTERSECT`, `EXCEPT`, `UNION`, and 2) queries containing nested queries in `WHERE` conditions. For instance, the query in Table 1 implies the constraint that `song.artist_name` should be a subset of `artist.artist_name`. In our template, `FK1` captures the constraint of the key relationship between the two `artist_name` columns, which prevents the template from generating nonsensical queries such as `SELECT` **`gender`** `FROM artist INTERSECT SELECT` **`country`** `FROM artist`.

3.1.3 Schema-distance-weighted Column Sampling

To mitigate the issue of arbitrary multi-table joins, we implement a weighted sampling function biased toward columns that are close, in terms of table distance (defined below), to the columns already selected in a SQL template.

For a given database d, we first establish an undirected graph for all the tables in d. Each table represents a node in the graph. The distance between any two tables, $e(\cdot, \cdot)$, is the least number of joins necessary to join the two tables (i.e. shortest path distance) under the restriction that table join can only take place with qualified primary key and foreign key pairs. See Appendix B for a detailed example demonstrating how table distances are computed.

Define a template t as $(q, \mathbf{c}, \mathbf{v})$, where q is the flat template string, $\mathbf{c} = [c_1, \ldots, c_m]$ is the set of column placeholders, and $\mathbf{v} = [v_1, \ldots, v_n]$ is the set of value placeholders in q. Denote T_c to represent the table that contains column c and $S_d(\tau)$ as the set of columns in d with the *strong type* τ. Given a template t and a qualified database d, the fundamental algorithm of SQL synthesis is described in Algorithm 1.

Algorithm 1: Single SQL Synthesis with Schema-Weighted Column Sampling

Input : template $t = (q, \mathbf{c}, \mathbf{v})$, database d, decay rate γ

Output: SQL query y

1 Let $y = q$

2 Random sample z_1 from $S_d(\tau_{c_1})$ and replace c_1 with z_1 in y

3 Compute sampling weights

$$w(z) = \begin{cases} 1, & \text{if } T_z = T_{c_1} \\ \frac{1}{\gamma^{\delta_{c_1}(z)}}, & \text{o.w.} \end{cases}, \quad \forall z$$

where $\delta_c(z) = e(T_c, T_z)$

4 **for** $c \leftarrow c_2 : c_m$ **do**

5 Compute sampling distribution

$$p(z) = \begin{cases} \frac{w(z)}{\sum_{z' : \tau_{z'} = \tau_c} w(z')}, & \text{if } \tau_z = \tau_c \\ 0, & \text{o.w.} \end{cases}$$

6 Sample z from $S_d(\tau_c)$ with p

7 Replace c with z in y

8 Update sampling weights, $\forall z$,

$$w(z) \leftarrow w(z) + \begin{cases} 1, & \text{if } T_z = T_c \\ \frac{1}{\gamma^{\delta_c(z)}}, & \text{o.w.} \end{cases},$$

9 **end**

10 **for** $v \leftarrow v_1 : v_n$ **do**

11 Identify relevant columns w.r.t. v and retrieve a set of possible values for v from the d

12 Random sample one value from the set and replace v with the value in y

13 **end**

The intuition behind the schema-weighted column sampling algorithm is as follows: after we select the first column for the given template (Line 2), we want to choose other columns in the database that are more relevant to the first column, so as to boost the chance of synthesizing more realistic SQL queries. We do so by sampling columns, for the remaining column placeholders in the template, according to a particular sampling probability (Line 5-6), which is a monotonically decreas-

Template	`SELECT col1_numberkey WHERE col2_name = VALUE`
Random	`SELECT T1.Club_ID FROM club AS T1 JOIN coach as T2 ON T1.Club_ID = T2.Club_ID JOIN player_coach AS T3 ON T2.Coach_ID = T3.Coach_ID JOIN player AS T4 on T3.Player_ID = T4.Player_ID where T4.Rank = "3rd"`
Ours	`SELECT Club_ID FROM club WHERE Club_Name="AIK"`

Table 2: Random sampling vs our schema-distance-weighted column sampling for a given template. The former produced a query with three joins while ours have both columns from the same table.

ing function of the edge value in the table graph for type-qualified *column candidates* (Line 3, 8), and 0 for non-qualified *column candidate*. Such implementation is motivated from the observation that over-lengthy SQLs resulted from multiple tables joins are rare in real world scenarios under the only-join-on-primary-key-foreign-key assumption. Table 2 shows an example of how adopting the schema-weighted sampling can help reduce the unrealistic SQLs in the random case.

Value of γ in Algorithm 1. γ is a hyperparameter that controls the decay rate in the sampling probability for columns that are farther away from the columns that have already been selected. Under the restricted join condition, we look at the number of tables in a query as a proxy to the table distance. To determine the value of γ, we randomly sample 7000 synthetic SQL queries with replacement and calculate the average number of tables from the samples. We repeat this process for 1000 times and plot the distribution. Then we perform the same steps for the real Spider training data. We choose γ so that the distribution of the average number of tables in the synthetic data is close to the real data. This helps prevent generating over-simplified or over-complicated SQL queries.

Based on this experiment, we chose γ to be 5 for the Spider benchmark. Figure 3 displays the distribution for three types of datasets: Spider training, synthetic dataset with schema-distance-weighted column sampling, and synthetic dataset with random column sampling. The figure demonstrates that the weighted sampling process, which provides an interface to tune the value of γ, can generate synthetic SQL queries that better match the real training data.

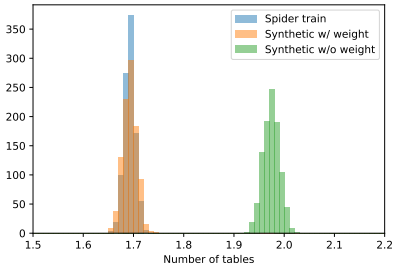

Figure 3: Histogram of the average table count (i.e. umber of joins) for three types of datasets with $\gamma = 5$. Our schema-distance-weighted column sampling reduces the table number of synthetic SQLs and better matches the training distribution.

3.2 NLQ Synthesis

Intermediate representation (IR) has been employed to simplify the SQL query with minimum information loss (Gan et al., 2021a; Guo et al., 2019b; Gan et al., 2021b; Guo et al., 2019a; Yu et al., 2018a; Shi et al., 2021). Common operations include removing `FROM/JOIN` clauses and `GROUP BY` clauses, and merging `WHERE` clauses and `HAVING` clauses. Previous work found the use of IR often improves text-to-SQL performance.

In this section, we explore whether the SQL-to-text generation could also benefit from an IR. According to a prior research by Wu et al. (2021), altering the query's linearization order could already affect the synthetic text quality. The objective of an IR here is to convert SQL to a representation that more closely resembles the NLQ. This conversion involves both simplifications (such as removal of redundant information) and specification (such as introducing information using heuristics).

We outline the main new rules to transform SQLs into IRs and explain the rationale (examples in Table 3):

EX1	**SQL**	`SELECT T1.name FROM student AS T1 JOIN has_pet AS T2 ON T1.student_id = T2.has_pet.student_id`
	IR	`SELECT name of student FROM has_pet`
	NLQ	Find the name of students who have pets.
EX2	**SQL**	`SELECT T2.name, count(*) FROM concert AS T1 JOIN stadium AS T2 ON T1.stadium_id = T2.stadium_id GROUP BY T1.stadium_id`
	IR	`SELECT name of stadium, Count ( record of concert ) GROUP BY ( stadium_id of concert )`
	NLQ	Show the stadium name and the number of concerts in each stadium.
EX3	**SQL**	`SELECT T1.neighbourhood_name neighbourhood AS T1 JOIN business AS T2 ON T1.business_id = T2.business_id WHERE T2.city = "Madison" GROUP BY T1.neighbourhood_name ORDER BY COUNT ( DISTINCE T2.name ) DESC LIMIT 1`
	IR	`SELECT neighbourhood_name of neighbourhood WITH most Count ( DISTINCT name of business ) WHERE city of business = "Madison"`
	NLQ	Which neighbourhood has the most number of businesses in Madison?
EX6	**SQL**	`SELECT T2.name FROM USER AS T2 JOIN review AS T1 ON T2.user_id = T1.user_id GROUP BY T2.name HAVING AVG (T1.rating) < 3`
	IR	`SELECT EACH ( name of user ) WITH Avg ( rating of review ) < 3`
	NLQ	Find users whose average review rating is below 3.

Table 3: IR examples that illustrate the examples of removing tables, enriching * columns, specifying most/least intent, removing redundant `GROUP BY`. Unwanted intents are in grey, redundant intents are in green. Texts related to IR operations are highlighted with yellow.

- Only drop tables in the FROM/JOIN phrase if they appear in other SQL elements (**EX2-EX4**). Removal of tables can simplify queries but tables in JOIN can also behave as filters and need to be preserved to avoid information loss (**EX1**).
- Replace * in count(*) with the table whose columns in JOIN act as foreign key to provide explicit context for counting. This is because, in multi-table join queries, foreign key represents the many of the one-to-many relations and thus the rows from the table is more meaningful to be aggregated (see **EX2** replaces * with concert rather than stadium).
- When SQL contains ORDER BY COUNT (...) LIMIT ... , rewrite the query to explicitly express the most or least intent for better intent alignment (**EX3**).
- Drop GROUP BY phrase if the column grouped by appears in SELECT and attach EACH to the specific column if the query does not express the most/least intent (see GROUP dropped in **EX3 - EX4** but not **EX2**). This aims to distinguish SQLs with GROUP BY and SELECT on the same column from those without SELECT.

Similar to previous IR designs, we also removed repeated text in EXCEPT/INTERSECT/UNION queries and made lexical adjustments.

4 Experiments

We conduct experiments on the Spider benchmark and demonstrate the effectiveness of our synthesis framework for the text-to-SQL task.

4.1 Experiment Setup

Spider Benchmark Spider (Yu et al., 2018b) is a large-scale text-to-SQL dataset. It has 10,181 annotated questions, 5693 unique complex SQLs, and 200 databases with multiple tables. The **train/train-others/dev/test** sets contain 7000/1659/1034/2147 examples and 140/6/20/40 databases, respectively. Spider has a challenging and realistic evaluation setting, where SQL queries and databases do not appear across different splits, posing a generalization challenge for text-to-SQL semantic parser. Since Spider test set is not publicly available, we use dev set for evaluation and train-others for checkpoint selection.

SQL-to-text Generator We finetune T5-large models on Spider training set for NLQ synthesis using both SQL and IR as input, named SQL2NLQ and IR2NLQ respectively. The best checkpoints are selected with the highest BLEU score on **train-others**.

Text-to-SQL Parser We test our synthetic data with several top-performing text-to-SQL parsers, including T5-3B, RASAT, and T5-3B + PICARD. We use T5-3B (Raffel et al., 2020) as our base parser, since previous work (Shaw et al., 2021) has shown that T5-3B can achieve competitive performance for Text-to-SQL semantic parsing. RASAT leverages the advantage of T5 and integrates relational structures into the pretrained model. Recently, PICARD (Scholak et al., 2021) demonstrates that constraint decoding on top of T5-3B can produce state-of-the-art performance on Spider.

Training Configurations We adopt a two-stage text-to-SQL training mechanism (Wang et al., 2021) in our experiment. In *Stage 1*, we use synthetic data only for model pre-finetuning. In *Stage 2*, we initialize the model weights with the first stage checkpoint, and then finetune it on the real data only. Both stages share the same hyperparameters. We train T5 with Adafactor and learning rate of 1e-4, and use gradient accumulation batch size 2050 for T5-3B model.

4.2 Spider Results and Analysis

The overall results[3] are shown in Table 4. We can see our synthetic data can further improve all three text-to-SQL parsing model, and achieve the best results with T5-3B + PICARD on Spider development set. Specifically, we have 4.4 points of EM score improvement on top of T5-3B model, while previous work (Wu et al., 2021; Wang et al., 2021) has marginal gain or even hurts the performance, demonstrating the effectiveness of our proposed method. More importantly, T5-3B has shown SOTA or near SOTA performance on 21 knowledge grounding tasks (Xie et al., 2022), our success of improving T5-3B with synthetic data for text-to-SQL can potentially generalize to other semantic parsing tasks with different logical forms. PICARD is an incremental parsing method for constraint decoding, which can reduce the syntax errors of language models for SQL generation. From Table 4,

[3]Some models do not predict cell values or access to database content, we leave '-' for EX.

Model	EM	EX
DT-Fixup SQL-SP (Xu et al., 2021)	75.0	-
LGESQL + ELECTRA (Cao et al., 2021)	75.1	-
S2SQL + ELECTRA (Hui et al., 2022)	76.4	-
DT-Fixup + Syn (Yang et al., 2021)	76.4	-
T5-3B (Shaw et al., 2021)	70.0	-
T5-3B + Syn data (Wu et al., 2021)	69.1	-
T5-3B + Syn data (Wang et al., 2021)	70.3	-
T5-3B + Syn data (ours)	74.4	-
T5-3B + PICARD (Scholak et al., 2021)	74.1	-
T5-3B + PICARD + Syn data (ours)	**76.9**	-
SmBoP + GraPPa (Rubin and Berant, 2021)	69.5	71.1
GAP + NatSQL (Gan et al., 2021a)	73.7	75.0
RASAT (Qi et al., 2022)	72.6	76.6
RASAT + Syn data (ours)	74.4	78.8
T5-3B† (Scholak et al., 2021)	71.5	74.4
T5-3B† + Syn data (ours)	74.5	78.6
T5-3B† + PICARD (Scholak et al., 2021)	75.5	79.3
T5-3B† + PICARD + Syn data (ours)	**76.1**	**81.4**

Table 4: Comparison of the top-performing text-to-SQL models in Spider leaderboard, as well as models trained with synthetic data (where synthetic are generated by training schema only). We report exact set match (EM) and execution accuracy (EX) for Spider dev set. † means T5-3B is trained with database content. For each of the parsing model (RASAT, T5-3B, T5-3B + PICARD), we observe performance improvements when trained with our synthetic data.

we see that T5-3B combined with PICARD and our synthetic data performs the best, implying the orthogonality of synthetic data augmentation and constraint decoding.

We also submitted our model to the official Spider website for evaluation on the hidden test set and received 76.6 for EX and 73.1 for EM. Without developing new model architectures for the text-to-SQL task, we achieved best performance by only augmenting synthetic data on top of the training set, compared to all other non-anonymous evaluation submission. Since our approach is generic for the text-to-SQL task and model-independent, we can easily apply our framework to other model submissions for additional improvements.

In Figure 1, we plot the training curves with different synthetic datasets. Compared with previous work (Wu et al., 2021; Wang et al., 2021), our synthetic data demonstrates significant improvement in Stage 1, with NLQ synthesized from both SQL2NLQ and IR2NLQ generator models, proving the high-quality of our synthesized SQLs. We also compare the generated NLQs with different automatic measurements in Table 5, where we can see IR benefits the NLQ generation process and produces the text closer to groundtruth NLQs.

Settings	BLEU	R-1	R-2	P-BERT	R-BERT
SQL→ NLQ	27.7	59.6	35.3	93.6	93.2
SQL→ IR→ NLQ	29.3	60.5	36.8	93.9	93.3

Table 5: Generated NLQ quality evaluations on the Spider dev set between SQL→ NLQ and SQL→ IR→ NLQ. The BLEU (Papineni et al., 2002), ROUGE (Lin, 2004), and P-BERT/R-BERT (Zhang et al., 2020) scores show that IR helps generate NLQs that are closer to the groundtruth.

4.3 Ablation Study

We conduct ablation study to understand the efficacy from each of the techniques we propose. In particular, we remove each technique and regenerate the synthetic data during Stage 1 training (using T5-3B) and evaluate on the Spider dev set. Results are summarized in Table 6.

Setting	EM	EX
All techniques included	**74.5**	**78.6**
w/o strong typing	72.3 (-2.2)	76.7 (-1.9)
w/o template key relation preserv	71.5 (-3.0)	77.2 (-1.4)
w/o schema-dist-weighted col sample	73.7 (-0.8)	76.6 (-2.0)

Table 6: Ablation study on the three proposed synthesis techniques. We generate the same amount of synthetic data with each of the proposed technique removed and repeat the same training and evaluation process. Both EX and EM are worse on the dev set.

We can see that removing any technique will result in a decrease in the final evaluation for the Spider dev set, indicating that each of the proposed three synthesis strategies contributes to a good quality synthetic data. In addition, Figure 1 shows that IR generated NLQ in general yields better results compared to SQL generated NLQ, which demonstrate the effectiveness of the proposed IR technique during NLQ synthesis.

4.4 Discussions

Few-shot setting: How much real data do we need before achieving acceptable performance? Annotating text-to-SQL dataset takes extremely high human effort. In practice, it is hard to create a large-scale corpus with a limited annotation budget. Table 7 presents the text-to-SQL semantic parsing results with a limited number of training examples. We also generate our synthetic data on top of the corresponding subset. Interestingly, as training size decreases from 7K to 128, our synthetic data becomes more essential, and the performance gain increases from 4.4 points to 27.2 points. Even with only 512 training examples, our synthetic data can assist the T5-3B model to achieve ∼60% accuracy

Model	f-shot:	128	256	512	1024	full (7k)
	# tmpl	68	116	205	318	746
	# syn	7839	10775	14457	17002	21851
T5-3B	real only	19.1	32.3	43.6	53.2	70.0
	real + syn	46.3	54.4	59.9	62.2	74.4

Table 7: Text-to-SQL experiment with the few-shot setting, where we sampled a subset from the original Spider training set with size varying from 128 to 1024, then created synthetic data with templates only from the subset. **# tmpl** and **# syn** represent the number of templates and synthesized NLQ-SQL pairs for the corresponding training subset. We report exact set match on the Spider dev set.

level. These few-shot setting results are encouraging, as we can annotate a small-scale training set and still achieve acceptable performance with the help of synthetic data.

Generator size: How big of the generator model do we need to produce high-quality NLQs? Since our proposed IR is to reduce the gap between NLQs and SQLs, we hypothesize that the NLQ generation process should have less reliance on model size. As shown in Figure 4, our synthetic data (with IR2NLQ) still presents comparable performance even with a small size T5-base generator, implying the effectiveness and robustness of our proposed IR. As comparison, SQL2NLQ has larger divergence between T5-Large and T5-Base, indicating some difficulty of translating SQL to NLQ.

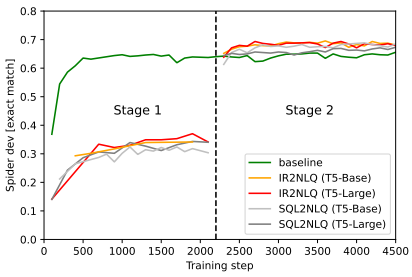

Figure 4: Comparison of different T5 model sizes for NLQ generations. On top of T5-Base (220M parameters) and T5-Large (770M parameters), we finetune generators for both SQL-to-text and IR-to-text, then evaluate the effectiveness with text-to-SQL semantic parsing in Spider.

Seen schema: How good of the synthetic data if we consider a broader coverage of database schema? Since the cross-database evaluation setting presents generalization challenge for text-to-SQL parsers, our synthetic framework can potentially overcome this by utilizing more public database schemas, or even ones that can implicitly cover the evaluation set. In addition to using schema from training set, we can take advantage of more public schemas for data synthesis, for example, WikiTables (Bhagavatula et al., 2015), GitTables (Hulsebos et al., 2021), WikiSQL (Zhong et al., 2017) and SQL tutorial websites. Some of them are even schema source for Spider benchmark. We simply added 20 databases from dev set into our synthetic data generation, then trained text-to-SQL parser on top of T5-Large. With this setting, we observed ∼2 points of performance improvement compared to that with training schema only. This pilot study implies the potential helpfulness of synthesizing data with targeting database schemas to further improve the downstream performance. While this setting breaks the cross-schema setting, we believe it still has practical values.

Single-table: How effective is our method on the single-table text-to-SQL parsing? Although our SQL synthesis is mainly designed for multi-table operations, it should also be compatible with the single table, but with foreign key preservation ineffective. WikiSQL (Zhong et al., 2017) and SQUALL (Shi et al., 2020) are two popular datasets for single-table text-to-SQL parsing. Compared to multi-table case, the single-table is much easier, for example, most text-to-SQL parsers are above 90% accuracy level in WikiSQL[4]. We took a relatively challenging SQUALL dataset for experiment. From 9K training examples, we created 30K synthetic NLQ-SQL pairs. We observe a smaller performance gain and hypothesize several reasons: 1) foreign key relationship is not applicable in single table, but critical to our data synthesis framework (as shown in the ablation study); 2) 9k examples are sufficient for model training, especially for SQLs without JOIN clause. Therefore, the effect of synthetic data is further diluted.

5 Conclusion

In this work, we proposed a data synthesis framework for text-to-SQL semantic parsing. After incorporating key relationships from schema, imposing strong typing, conducting schema-distance-weighted column sampling, and bridging SQL → NLQ generation with an intermediate representation, we synthesized high-quality dataset that can further improve the state-of-the-art parser on Spider benchmark. We also revealed the efficiency of the synthetic data and pointed out the potential usefulness of reducing human annotations for text-to-SQL parsing.

[4] https://github.com/salesforce/WikiSQL

Limitations

The proposed synthesis framework has been targeted at text-to-SQL task, which may not generalize to other tasks that require large amount of synthetic data without major modification. For instance, other popular tasks involving converting natural language questions to some sort of logic forms are in natural very similar to text-to-SQL, yet all techniques relying on the "key" property in the database might no longer be applicable. On the other hand, the template based synthesis method currently relies on templates extracted from the real data. By incorporating some carefully designed grammar (e.g. PCFG), we may be able to further enrich the template set.

Ethics Statement

The training and evaluation of our experiments rely on many compute resources, which may not be environment-friendly. For example, each parsing model requires training using NVIDIA A100-SXM4-40GB GPUs for many hours, which can inevitably cause more CO2 emission.

References

Chandra Bhagavatula, Thanapon Noraset, and Doug Downey. 2015. Tabel: Entity linking in web tables. In *SEMWEB*.

Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. 2021. LGESQL: Line graph enhanced text-to-SQL model with mixed local and non-local relations. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2541–2555, Online. Association for Computational Linguistics.

Yujian Gan, Xinyun Chen, Jinxia Xie, Matthew Purver, John R. Woodward, John Drake, and Qiaofu Zhang. 2021a. Natural SQL: Making SQL easier to infer from natural language specifications. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2030–2042, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Yujian Gan, Xinyun Chen, Jinxia Xie, Matthew Purver, John R. Woodward, John H. Drake, and Qiaofu Zhang. 2021b. Natural SQL: making SQL easier to infer from natural language specifications. *CoRR*, abs/2109.05153.

Daya Guo, Yibo Sun, Duyu Tang, Nan Duan, Jian Yin, Hong Chi, James Cao, Peng Chen, and Ming Zhou. 2018. Question generation from SQL queries improves neural semantic parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1597–1607, Brussels, Belgium. Association for Computational Linguistics.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019a. Towards complex text-to-sql in cross-domain database with intermediate representation. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4524–4535. Association for Computational Linguistics.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019b. 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.

Binyuan Hui, Ruiying Geng, Lihan Wang, Bowen Qin, Yanyang Li, Bowen Li, Jian Sun, and Yongbin Li. 2022. S^2SQL: Injecting syntax to question-schema interaction graph encoder for text-to-SQL parsers. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1254–1262, Dublin, Ireland. Association for Computational Linguistics.

Madelon Hulsebos, Çağatay Demiralp, and Paul Groth. 2021. Gittables: A large-scale corpus of relational tables. *arXiv preprint arXiv:2106.07258*.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12–22, Berlin, Germany. Association for Computational Linguistics.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Jiexing Qi, Jingyao Tang, Ziwei He, Xiangpeng Wan, Chenghu Zhou, Xinbing Wang, Quanshi Zhang, and Zhouhan Lin. 2022. Rasat: Integrating relational structures into pretrained seq2seq model for text-to-sql.

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

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

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2021. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 922–938, Online. Association for Computational Linguistics.

Peng Shi, Patrick Ng, Zhiguo Wang, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Cicero Nogueira dos Santos, and Bing Xiang. 2021. Learning contextual representations for semantic parsing with generation-augmented pre-training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 13806–13814.

Tianze Shi, Chen Zhao, Jordan Boyd-Graber, Hal Daumé III, and Lillian Lee. 2020. On the potential of lexico-logical alignments for semantic parsing to SQL queries. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1849–1864, Online. Association for Computational Linguistics.

Bailin Wang, Wenpeng Yin, Xi Victoria Lin, and Caiming Xiong. 2021. Learning to synthesize data for semantic parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2760–2766, Online. Association for Computational Linguistics.

Kun Wu, Lijie Wang, Zhenghua Li, Ao Zhang, 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. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8974–8983, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I. Wang, Victor Zhong, Bailin Wang, Chengzu Li, Connor Boyle, Ansong Ni, Ziyu Yao, Dragomir Radev, Caiming Xiong, Lingpeng Kong, Rui Zhang, Noah A. Smith, Luke Zettlemoyer, and Tao Yu. 2022. Unifiedskg: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. *arXiv preprint arXiv:2201.05966*.

Peng Xu, Dhruv Kumar, Wei Yang, Wenjie Zi, Keyi Tang, Chenyang Huang, Jackie Chi Kit Cheung, Simon J.D. Prince, and Yanshuai Cao. 2021. Optimizing deeper transformers on small datasets. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2089–2102, Online. Association for Computational Linguistics.

Wei Yang, Peng Xu, and Yanshuai Cao. 2021. Hierarchical neural data synthesis for semantic parsing. *arXiv preprint arXiv:2112.02212*.

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, Richard Socher, and Caiming Xiong. 2021. Grappa: Grammar-augmented pre-training for table semantic parsing. In *International Conference on Learning Representations*.

Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir R. Radev. 2018a. Syntaxsqlnet: Syntax tree networks for complex and cross-domain text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 1653–1663. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018b. 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.

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.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. Bertscore: Evaluating text generation with bert. In *International Conference on Learning Representations*.

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 Existing Synthesis Frameworks

Inspired by prior work by Jia and Liang (2016) in semantic parsing, Yu et al. (2021) extended a synchronous context-free grammar (SCFG) approach to the text-to-SQL task where they manually crafted about 90 high-quality SQL-NLQ aligned patterns to generate new SQL-NLQ pairs. They found pretraining on the synthetic dataset leads to a significant improvement even tested with a very strong text-to-SQL parser RAT-SQL on the Spider benchmark.

While SCFG usually creates high-quality data because patterns are carefully designed and aligned, the coverage of the patterns is limited, and expert knowledge is required to design such patterns. Thus, more efforts are devoted to automating the procedure. Guo et al. (2018) utilized a two-stage approach by first sampling SQL queries from a simple pattern and then generating questions using a copy-based RNN encoder-decoder structure find the synthetic data that can improve the existing state-of-the-art model on the WikiSQL benchmark. Zhong et al. (2020) followed the same two-stage approach but used templates extracted from training to generate SQL and augmented the NLQ generator with pretrained transformer BERT and iteratively updated the parser and generator. Only the synthetic dataset that was created using target schemas filtered with cycle consistency can facilitate the downstream performance.

Along the same approach, Wang et al. (2021) identified problems with fixed SQL synthesis rules and employed a full-fledged probabilistic context-free grammar (PCFG) that enabled generating SQLs with varying structures. They synthesized natural language queries with a BART SQL-NLQ generator. Their synthesis method has been shown to boost the RAT-SQL parser performance on the Spider benchmark, though the improvement is not as significant as pretraining using SCFG generated synthetic data (Yu et al., 2021). The gap might be due to the quality of the synthetic dataset as the independent selection of generation step in PCFG introduces substantial noise such as illogical SQL queries.

To improve the quality of synthetic data, Wu et al. (2021) introduced a clause-level synthesis framework: first decomposing a query into sub-clauses and translating sub-SQL clauses into sub-questions, and finally assembling sub-questions into a whole question. They found clause-based synthesis method is better than flat synthesis.

Alternatively, Yang et al. (2021) proposed to improve the quality of synthetic data by incorporating domain information in question generation. Specifically, they learned an entity sampler and synthesized questions using an entity-to-question generator with entities sampled from the sampler, followed by generating pairing SQL queries through a baseline parser. For this approach, they also attractively updated the parser and generator, in a similar fashion as in Zhong et al. (2020). Their synthetic dataset can significantly improve a DT-Fixup parser on the Spider benchmark.

This work seeks to investigate value of synthetic dataset with current state-of-the-art PICARD model and refine a synthetic method in an automate and non-iterative manner. Thus, we examine two synthetic datasets from recent work (Wang et al., 2021; Wu et al., 2021) that demonstrate improvement of downstream performance with previous state-of-the-art text-to-SQL parser (RAT-SQL) over Spider benchmark without iterative training.

Paper	Method	SQL Synthesis		NLQ Synthesis		SQL-NLQ Bridging	Manual Effort
		Abstraction	Limitation	Procedure	Generator		
Guo et al (2018)	Two stage	Template	Single template, no `JOIN`	SQL → NLQ	copy-based RNN	-	minimal
GAZP (Zhong et al 2020)	Iterative two stage	Template	Violating foreign key relations, limit to training templates	SQL → NLQ	BERT + point decoder	-	minimal
Wang et al (2021)	Two stage	PCFG	OP/COL incompatibility, invalid SQL structure	SQL → NLQ	BART	-	minimal
Wu et al (2021)	Two stage	CFG	No support for IEU, no `JOIN`	SQL → Sub-SQL → NLQ fragment →NLQ	copy-based RNN	SQL clause / NLQ fragment	combination rules
Yang et al (2021)	Iterative reversed two stage	-	Dependent on base parser	schema → entity → NLQ	T5	-	minimal
Grappa (Yu et al 2021)	Synchronous	Template	Limit to training templates	simultaneous instantiation of SQL-NLQ template		Aligned SQL-NLQ template	alignment
Ours	Two stage	Template	Limit to training templates	SQL→ IR→ NLQ	T5	IR	minimal

Figure 5: Comparison of different data synthesis methods for text-to-SQL task. *Synchronous* refers to generating SQL and NLQ together, *Two-stage* first synthesizes SQL then generates NLQ, *reversed two-stage* first generates NLQ then synthesizes SQL. **SQL-NLQ Bridging** refers to intermediate operations or representations for matching SQL and NLQ.

B Details on Schema-distance-weighted Column Sampling

B.1 Table Distance.

For a given database d, we first establish an undirected graph for all the tables in d. We can then compute the distance between any two tables, $e(\cdot, \cdot)$, defined as the least number of joins necessary to join the two tables under the restriction that table join can only take place with qualified primary key and foreign key information. In other words, we disable arbitrary join of two tables if they lack key and foreign key relationship.

We give some examples using one of the databases (id: `college_1`) in the Spider benchmark, as shown in Table 8.

- $e(\text{T1}, \text{T2}) = 1$ because the column `class code` in table *class* (T1) is a foreign key in table *course* (T2). We can also observe from the table graph in Figure 6: there is a direct path between table node *class* and table node *course*.
- $e(\text{T2}, \text{T7}) = 2$ since we first need to join table *course* (T2) with table *department* (T3), followed by joining table *department* with table *student* (T7). Note that even though we can also join using the path $T2 \rightarrow T1 \rightarrow T5 \rightarrow T7$, this is not the ***least*** number of joins between the two tables.

Table 8: Example database (id: `college_1`)

Alias	Table Name	Primary Key	Foreign Key	
			Table	Column
T1	class	`class code`	enroll	`class code`
T2	course	`course code`	class	`class code`
T3	department	`department code`	course	`department code`
			professor	`department code`
			student	`department code`
T4	employee	`employee number`	class	`professor employee number`
			department	`employee number`
			professor	`employee number`
T5	enroll	-	-	-
T6	professor	-	-	-
T7	student	`student num`	enroll	`student number`

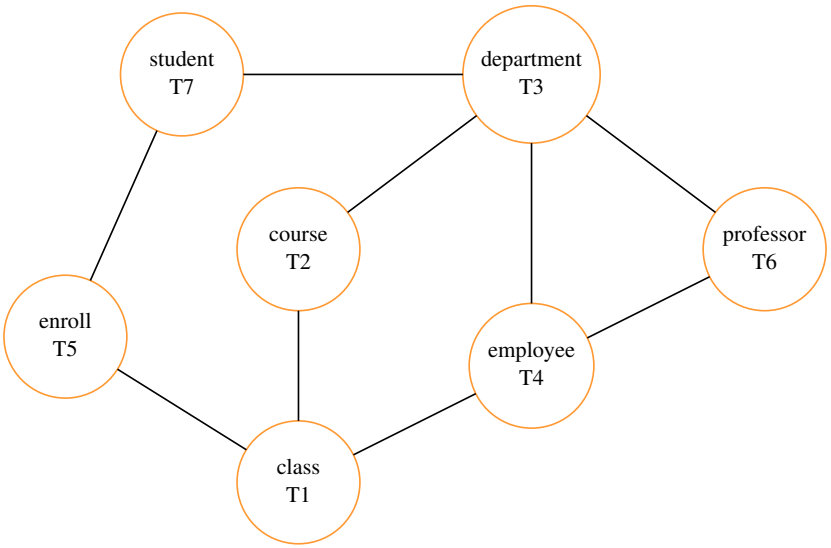

Figure 6: Example table graph (id: `college_1`)

The reason we introduce the concept of *table distance* is that we want to leverage this value to promote table joins with appropriate relationships while discouraging illogical joins when two tables are irrelevant. During the process of column sampling, we will choose columns that have smaller table distance with the other columns that have already been selected with the objective to create more realistic synthetic SQL queries. In the example above, assume we have first sampled a column from the table *student* (T7). For the next column placeholder, we are more likely to sample a column from table *enroll* (T5) than table *professor* (T6) — it is more natural to ask questions like "how many students enrolled in class X" compared to asking "how many students enrolled in classes taught by professors who were employed before year YYYY".

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
See the Limitations section.

☑ A2. Did you discuss any potential risks of your work?
See the Ethics Statement section.

☑ A3. Do the abstract and introduction summarize the paper's main claims?
See Abstract and Section 1 Introduction.

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☑ Did you use or create scientific artifacts?

We used the Spider data (Section 2 and Section 4)

☑ B1. Did you cite the creators of artifacts you used?
Section 2, 4 and References.

☒ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
Spider is one of the top used datasets for text-to-sql tasks and have been accepted by the community for scientific research.

☐ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
Not applicable. Left blank.

☐ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
Not applicable. Left blank.

☐ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
Not applicable. Left blank.

☑ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
See Section 4.1.

C ☑ Did you run computational experiments?

Section 4.

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
Ethics Statement and Section 4.

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Section 4.

☐ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
Not applicable. Left blank.

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
Section 4.

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
Not applicable. Left blank.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
Not applicable. Left blank.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
Not applicable. Left blank.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
Not applicable. Left blank.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
Not applicable. Left blank.