RESEARCH PAPER

Towards Text-to-SQL over Aggregate Tables

Shuqin Li[1], Kaibin Zhou[2], Zeyang Zhuang[2], Haofen Wang[1†], Jun Ma[3*]

[1]College of Design and Innovation, Tongji University, Shanghai, 200092

[2]School of Software, Tongji University, Shanghai, 201804

[3]School of Automotive Studies, Tongji University, Shanghai, 201804

Keywords: Text-to-SQL; Question Answering; Business Intelligence; Deep Learning

Citation: Li, S.Q., Zhou K.B., Zhuang, Z.Y., et al.: Towards text-to-SQL over aggregate tables. Data Intelligence 5(2), 457-474 (2023). doi: 10.1162/dint_a_00194
Received: March 5, 2022; Revised: October 12, 2022; Accepted: November 12, 2022

ABSTRACT

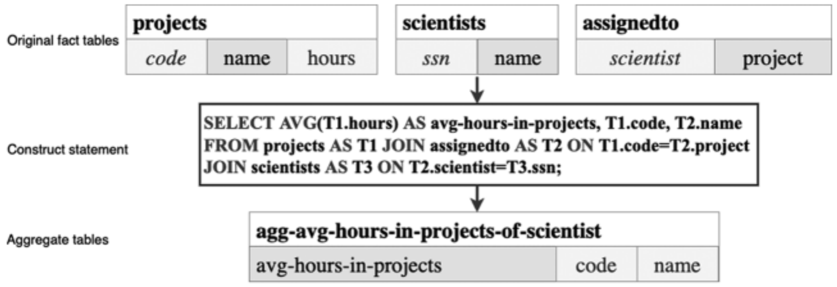
Text-to-SQL aims at translating textual questions into the corresponding SQL queries. Aggregate tables are widely created for high-frequent queries. Although text-to-SQL has emerged as an important task, recent studies paid little attention to the task over aggregate tables. The increased aggregate tables bring two challenges: (1) mapping of natural language questions and relational databases will suffer from more ambiguity, (2) modern models usually adopt self-attention mechanism to encode database schema and question. The mechanism is of quadratic time complexity, which will make inferring more time-consuming as input sequence length grows. In this paper, we introduce a novel approach named WAGG for text-to-SQL over aggregate tables. To effectively select among ambiguous items, we propose a relation selection mechanism for relation computing. To deal with high computation costs, we introduce a dynamical pruning strategy to discard unrelated items that are common for aggregate tables. We also construct a new large-scale dataset SpiderwAGG extended from Spider dataset for validation, where extensive experiments show the effectiveness and efficiency of our proposed method with 4% increase of accuracy and 15% decrease of inference time w.r.t a strong baseline RAT-SQL.

† Corresponding author: Haofen Wang (E-mail: carter.whfcarter@gmail.com; ORCID: 0000-0002-0672-8081).
* Corresponding author: Jun Ma (E-mail: majun tongji@foxmail.com).

1. INTRODUCTION

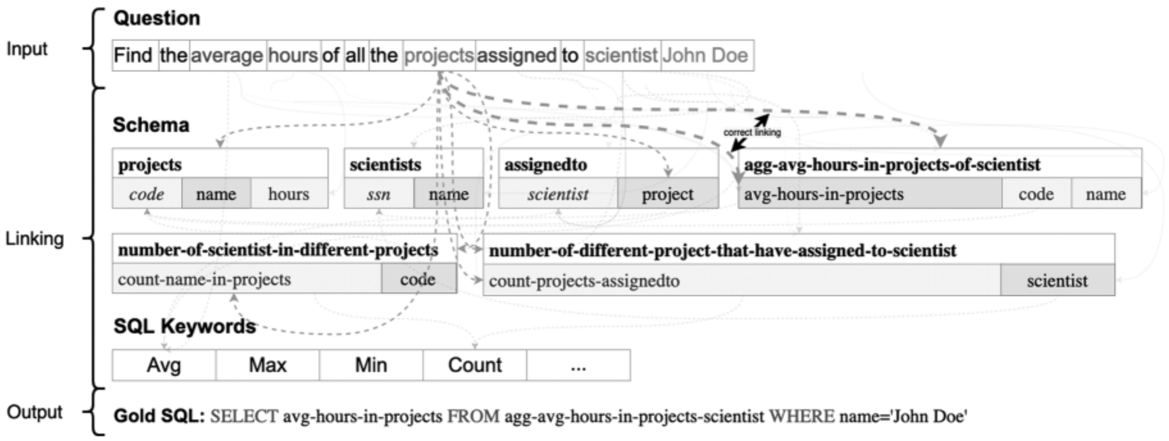
Text-to-SQL unlocks the ability to translate textual questions into the corresponding SQL queries. It has many applications such as Argumented Business Intelligence (ABI), question answering and fact searching. Recently, a series of datasets and models [1, 2, 3, 4, 5, 6, 7, 8] have been proposed.

In database querying, aggregate tables are widely used to move time-intensive calculations into pre-computed storage by changing the granularity on specific dimensions and aggregating data up along these dimensions based on existing tables [9]. Although text-to-SQL receives much attention, recent studies paid little attention to this scenario. Figure 1 shows an aggregate table and its construction statement. The database contains three fact tables: `Projects`, `Scientists` and `AssignedTo`. The aggregate table `agg avg hours of projects scientist` stores the average hours of all the projects assigned to each scientist and their names. When one wants to know a specific scientist's average hours of all assigned projects, we can run it against the appropriate aggregate table.



```
SELECT AVG(T1.hours) AS avg-hours-in-projects, T1.code, T2.name
FROM projects AS T1 JOIN assignedto AS T2 ON T1.code=T2.project
JOIN scientists AS T3 ON T2.scientist=T3.ssn;
```

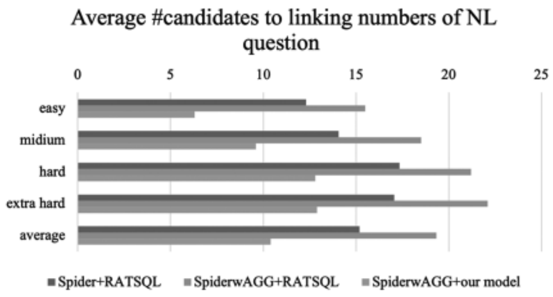
Figure 1. Aggregate table constructing example.


Input
Question
Find the average hours of all the projects assigned to scientist John Doe
Schema
Linking
correct linking
projects
code
name
hours
scientists
ssn
name
assignedto
scientist
project
agg-avg-hours-in-projects-of-scientist
avg-hours-in-projects
code
name
number-of-scientist-in-different-projects
count-name-in-projects
code
number-of-different-project-that-have-assigned-to-scientist
count-projects-assignedto
scientist
SQL Keywords
Avg
Max
Min
Count
...
Output

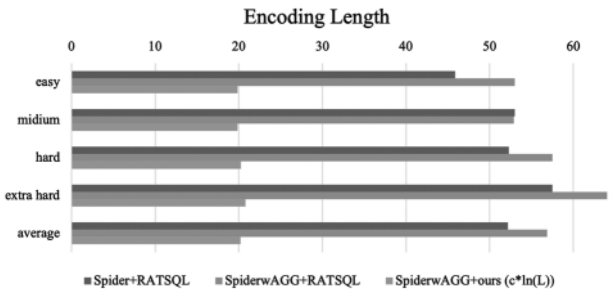
Gold SQL: SELECT avg-hours-in-projects FROM agg-avg-hours-in-projects-scientist WHERE name='John Doe'

Figure 2. Demonstrate textual linking of an example on the new dataset SpiderwAGG, number in the parentheses after the natural language question token represents the number of candidates to link. In this example, the average #candidates to link among question tokens is 4.17 (25/6 tokens), which makes it very ambiguous to distinct the correct candidates. It is due to the naming of aggregate tables relies on its original table.

The new scenario poses new challenges on the current complex text-to-SQL models in both effectiveness and efficiency ways. Firstly, a centric problem dealt with previous models [5, 10] is how to determine correct relations between questions and table schema. The researchers typically rely on textual matching to build the initial relations (shown in Figure 2, the token `projects` is matched with 6 table columns and table names, 2 with fact tables, 4 with aggregate tables, the thick link is correct). However, they lack a mechanism to select among multiple relations. The fact implies that those techniques exist a certain deficiency in the new scenario, i.e. determining correct relation is very ambiguous. As the aggregate table naming should manifest aggregate and original table information, the names of aggregate tables will refer to existing names, thus, a question token could match with multiple tables and columns of original tables. Figure 2 demonstrates the average relations for all tokens in the question sentence being 4.17. Further, Figure 3(a) compares the number of relations between Spider dataset and the dataset in the new scenario, relation selection becomes even harder to determine as SQL becomes harder. Secondly, it suffers from high computational costs. The self-attention mechanism has been widely used in many state-of-the-art text-to-SQL models in order to contextually encode question and DB schema. It has computational complexity of $\Omega(L^2)$, where L is the length of the question and table schema sequence. As L grows, computational cost grows rapidly. As in Figure 3(b), the introduction of aggregate tables brings longer sequences.



(a) Average #candidates to link between NL tokens and schema



(b) Average encoding length of different SQL hardness

Figure 3. (a) Average #candidates to link increase with different SQL difficulties. On our new dataset, RAT-SQL odel has more linking numbers (orange column) compared with the original dataset Spider (blue column). After applying our method (gray column), the linking number drops. (b) Average encoding length also increases with different SQL difficulties. More aggregate tables will increase the encoding length sharply. After applying our method, the situation is mitigated.

To this end, we propose a new model WAGG, which includes two innovative components. Firstly, we propose Relation Selection, which uses a fully-connected with residual network that identifies candidate to be matched items and discards the original relations incorrectly matched due to textual similarity; secondly, we propose Dynamic Pruning, which holds a simple intuition that many tables are not relevant for prediction, so it can be removed. The closest research is *ProbSparse* mechanism proposed in Informer [11] and it retains any elements that are K-most informative by measuring sparsity, reducing the time complexity and storage complexity to $\Omega(L\ log(L))$. Our method is different in two ways: first, we keep the pruned items in each head consistent, whereas Informer calculates the pruned items for each head; secondly, both the query and value length is reduced to $c\ ln(L)$, therefore the time complexity further reduced to $\Omega(log(L)^2)$.

Compared with the RAT-SQL [5] baseline (state-of-the-art on Spider Dataset), the overall efficiency and accuracy significantly increases.

To support training and evaluation of text-to-SQL over aggregate tables, we also construct a dataset called SpiderwAGG based on Spider. The dataset contains 406 aggregate tables and 9,074 NL-SQL data pairs on aggregate tables. We invent strategies to automatically generate the dataset and retain aggregate table diversity and coverage to the actual situation. Our text-to-SQL data pairs cover multiple domains. Considering that people often only perform simple queries on aggregate tables, we do not include nested SQL.

The contributions of this paper are three-fold:

- We present a new scenario for text-to-SQL the first time and identify two challenges: difficult keyword mapping and high computational cost;
- We propose a model WAGG, which contains two innovative components, which correspond to Dynamic Pruning and Relation Selection module. Therein, Dynamic Pruning im proves computation and storage efficiency and Relation Selection module helps to choose the correct one among multiple possible keywords;
- We construct a large-scale dataset containing 406 aggregate tables and more than 9,000 data pairs to support learning and evaluation.

The rest of the paper is organized as follows. Section 2 introduces the related work including dataset and models of recent progress of text-to-SQL. Section 3 introduces our model and methods of constructing the dataset. Section 4 introduces dataset analysis used for experiment and experimental results. Section 5 brings the conclusions.

2. RELATED WORK

2.1 Existing Text-to-SQL Models

Recent progress of text-to-SQL models mainly resorts from two aspects. Firstly, the rich expressioness of natural language requires not only to understand syntactic information, but also semantic information. Recent advance in pretraining models [12, 13] produces better word embedding that represents better semantic information, helping to map implicitly. The second aspect derives from the fact that SQL depends on not only the question, but also the schema of the database. Bogin et al. [14] first noted the importance of understanding schema. Researchers also took the database content into the encoding. RAT-SQL [5] proposes to incorporate contextual information from NL question, schema and encode them all together by the self-attention mechanism. On the aforementioned basis, Lin et al. [10] represented table schema, question and their textual matching relationships into a single tagged sequence and leave the transformer [15] model to calculate their relationship. However, the models are pruned to map tokens into SQL keywords and schema content separately, yet the field has ignored the importance of this result because existing datasets hardly contain operator-related schema names. The scenario that we propose not only emphasizes the challenge of understanding schema, and considers a more complex type of schema which brings multiple challenges.

2.2 Existing Text-to-SQL Datasets

There have been new datasets for Text-to-SQL proposed in recent years. Zhong et al. [1] constructed a cross-domain text-to-SQL dataset WikiSQL, presenting a new challenge requiring model's generalization in database split setting. They present a new challenge requiring model's generalization in database split setting. Based on it, Yu et al. [2] further proposed a dataset Spider, which contains more complex analytical calculations (e.g., GROUPBY, JOIN) and involves more tables in a nested SQL query. Many other datasets inquire the problem in different aspects. Gu et al. [16] built a complex and diverse dataset to deal with the occurrence of out-of-distribution questions, promoting the generalization of question answering models. There are also cases where the model is required to tackle information in diverse data forms. Wolfson et al. [17] introduced QDMR formalism to decompose complex questions and allow to query over multiple information sources, which effectively improves open-domain question answering efficiency. Baik et al. [18] proposed a dual-specification query synthesis system to incorporate NLQ and programming-by-example (PBE), enabling expressive SQL queries. To hybrid multiple sources, Chen et al. [19] presented a large-scale dataset with heterogeneous data forms and designed a baseline architecture to cope with hybrid information. Wenhu et al. [20] proposed the techniques of early fusion and cross-block reader for open-domain question answering, facilitating the retrieval of evidence distributed across tabular and textual data, etc. However, the scenario with the presence of aggregate tables has been relatively less explored.

3. METHOD

In this section, we introduce our model (Figure 4). Our model is based on RAT-SQL [5], so we first make a brief introduction to the model RAT-SQL and then discuss the innovative modules proposed in the encoder. Our decoder shares the same implementation with RAT-SQL.

3.1 Preliminary Introduction of RAT-SQL

To answer a question for unseen table schema, RAT-SQL [5] proposes relation-aware attention mechanism to contextually encode questions, table schema and relations between them.

The researchers first represent natural language question and its table schema with common NLP embedding methods such as GloVe and BERT, then adopt Bi-LSTM to process the embeddings. The final input thus is $X = \{x_i^{init}\}_i^n$, in which x^{init} denotes a question token or a table schema item's representation. Then one encodes them together with relation aware attention layers, for each element x^{init} in the total input, a relation-aware self-attention layer that contains H heads transforms it into y_i

$$e_{ij}^h = \frac{x_i W_Q^h (x_j W_K^h + r_{ij}^K)}{\sqrt{d/H}} \quad \alpha_{ij}^h = softmax_j\{e_{ij}^h\} \tag{1}$$

$$z_i^h = \sum_{j=1}^{n} \alpha_{ij}^h (x_j W_V^h + r_{ij}) \quad y_i = Concat(z_i^1, z_i^2, \ldots, z_i^H) \tag{2}$$

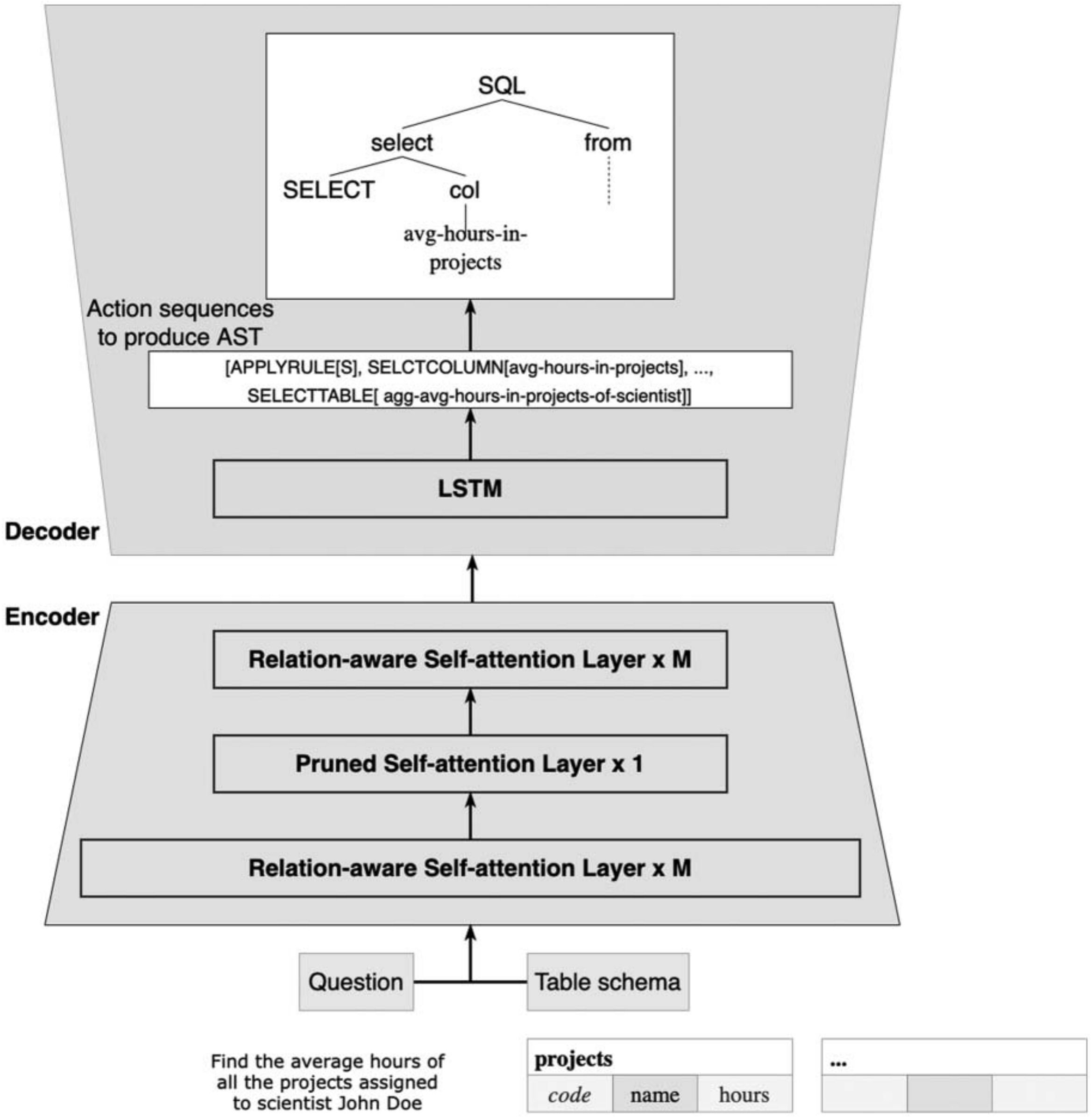

SQL
select
from
SELECT
col
avg-hours-in-projects
Action sequences to produce AST
[APPLYRULE[S], SELCTCOLUMN[avg-hours-in-projects], ...,
SELECTTABLE[agg-avg-hours-in-projects-of-scientist]]
LSTM
Decoder
Encoder
Relation-aware Self-attention Layer x M
Pruned Self-attention Layer x 1
Relation-aware Self-attention Layer x M
Question
Table schema
Find the average hours of all the projects assigned to scientist John Doe
projects
code
name
hours
...

Figure 4. Total Framework. In the encoder, besides the relation-aware self-attention layer, we add a pruned self-attention layer to abandon the irrelevant elements in the sequence. The relation selection mechanism is used in both type layers. For the decoder, we have the same implementation with RAT-SQL [5].

where r_{ij} represents the relation between element i and element j. Specifically, each relation is calculated via textual matching between database schema or value in the database and the question token. Each type of relation is assigned with a specific exlusive value. The types include QUESTION-COLUMN-M, QUESTION-TABLE-M, COLUMN-QUESTION-M or TABLE-QUESTION-M. Here M is one of EXACTMATCH, PARTIALMATCH, or NOMATCH.

To capture relations between the final encoder output and the schema, they also compute a memory-schema linking matrix for columns and tables. They compute matrix for tables similarly. The matrices for columns and tables are as follows:

$$\tilde{L}_{ij}^{col} = \frac{y_i W_Q^{col}(c_j^{col} + r_{ij}^K)}{\sqrt{d_x}}$$

$$L_{ij}^{col} = softmax\{\tilde{L}_{ij}^{col}\}$$

In the decoder, RAT-SQL employs the tree-structured architecture in [21]. The scholars adopted LSTM to generate a sequence of actions that eventually forms an abstract syntax tree(AST) of SQL. There are two types of actions, the first action is *ApplyRule*[*r*], which means the application of the grammar rule r on the last node; the second action is called SelectColumn[v] or *SelectTable*[*v*], it means selecting a column or a table v from the schema when completing a leaf node.

Therefore, the process of generating SQL can be represented formally as

$$p(SQL \mid Y) = \prod_d P(a_t \mid a_{<t}, Y)$$

$$m\_t,$$

$$m_t \;\; h_t = f_{LSTM}\left(\left[a_{t-1} \,\|\, z_t \,\|\, h_{p_t} \,\|\, a_{p_t} \,\|\, n_{f_t}\right] \;\; m_{t-1} \;\; h_{t-1}\right)$$

where Y is the representation given by the encoder, m_t is the LSTM cell state, h_t is the LSTM output at step t, a_{t-1} is the embedding of the previous action, p_t is the step corresponding to expanding the parent AST node of the current node, and n_{ft} is the embedding of the current node type. Finally, z_t is the context representation, computed using multi-head attention (with 8 heads) on h_{t1} over Y.

To determine actions, for *ApplyRule*[*r*], they compute $P(a_t = ApplyRule[r] | a < t) = softmax_r(g(h_t))$, where g is a 2-layer MLP with a tanh. For *SelectColumn*[*v*] or *SelectTable*[*v*],

$$\tilde{\lambda}_i = \frac{h_t W_Q (y_i W_K)}{\sqrt{d_x}} \quad \lambda_i = softmax\{\tilde{\lambda}_i\} \quad P(a_t | a_{<t}, y) = \sum_j^{|y|} \lambda_j L_{ji}^{col}$$

3.2 Relation Selection

Retaining all the relations brings both useful information and additional noise, it bears the main reason for wrong predictions. Noise is more common in this problem because a question token can be matched multiple times. Therefore, we design the relation selection module to help the model to determine the relationship between question tokens and tables' schema.

Specifically, we provide a way to learn to select relations that is needed for further prediction and abandoning the irrelevant relations. The information for distinguishing comes from an overall understanding about SQL, table schema and the question. However, it is learned through attention mechanism or other encoding mechanism. As shown in Figure 5(b), the token "project" has several relations computed by textual matching. In this way, multiple tokens "projects", "avg-hours-in-projects" and "number-of-scientist-in-different-project" are matched by adjusting the attention mechanism. We formalize a separate model that encodes both the target and the related tokens to learn to choose the correct relation.

$$e_{ij}^{h} = \frac{x_i W_Q^h (x_j W_K^h + f(r_{ij}^K, x_i, x_j)}{\sqrt{d/H}} \tag{3}$$

where *f* is a two-layer full-connected network and tanh nonlinearity, which helps to select proper relations. As shown in Figure 5(b), the embedding of token `projects` is originally matched with multiple words (in dark orange), and then the relation can be reduced into zero for the incorrect mapping with `number-of-scientists-in-different-projects`.

3.3 Dynamic Pruning

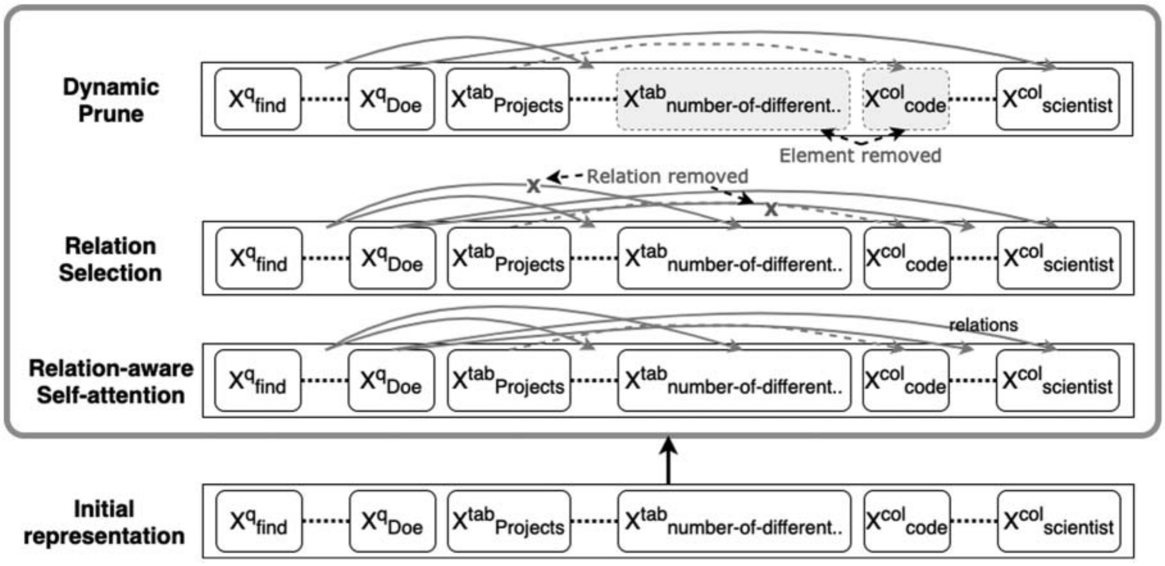
Dynamic pruning reduces the number of items to be encoded (Figure 5(c)). Since the new aggregate tables in a database have increased the number of tables' schema to be encoded. For a SQL, there are more irrelevant tables that are unnecessary to be encoded, thus could be removed. Informer proposes a method to measure sparsity between query and key and use it to find the most dominant ones. They compute a different dominant set for each head. In our case, it remains constant for a text token to match which table across all heads, so we only compute once for all heads. As defined in Informer [11], they first propose a sparsity measurement for each element in the sequence(Equation 4). Based on the measurement, they define *ProbSparse*

Self-attention by retaining top U most dominant queries q_j' to be attended on. Thus the queries are reduced to shape $U * d$. They choose U to be $c * log(L_K)$, where c is a constant scaling factor.

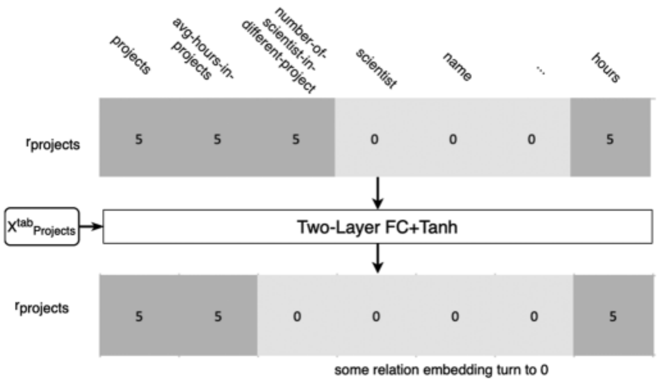
$$\bar{M}(q_i, K) = max_j \left\{ \frac{q_i k_j^T}{\sqrt{d}} \right\} - \frac{1}{L_K} \sum_{j=1}^{L_K} \frac{q_i k_j^T}{\sqrt{d}} \tag{4}$$

We further prune the relations by choosing from the index of top U queries to be r'. Also, since in the self-attention settings, the queries, values, keys are all the same, we pruned the values and keys the same way. Finally, we have a new pruned sequence X' of length $c * log(L)$, where L is the original length. Thus, the overall encoder equation is as follows, where x_i' are pruned queries

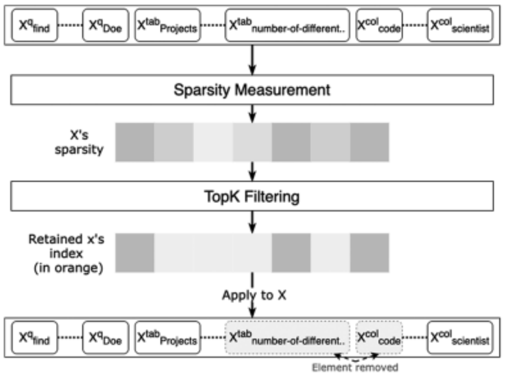
$$\begin{gathered} e_{ij} = \frac{x_i W_Q^h (x_j W_K^h + f(r_{ij}^K, x_i, x_j)}{\sqrt{d/H}} \\ \alpha_{ij} = softmax_j \{e_{ij}\} \\ Z_i = \sum_{j=1}^{n} \alpha_{ij} \left(x_j' W_V + r_{ij} \right) \end{gathered} \tag{5}$$



(a) WAGG self-attention layer



(b) Relation Selection



(c) Dynamic Prune

Figure 5. (a) The overall attention layer in our model. (b) Relation Selection Specification. (c) Dynamic Prune Specification.

3.4 Data Construction

3.4.1 Overview

The generation of the dataset mainly includes two steps: generating aggregate tables and generating text-SQL data pairs. Our method is based on previous work, which is worth a beforehand introduction.

Wang et al. [22] invented a phenomenal method to generate semantic parsing dataset overnight. They first generate logical forms and their pairing canonical text by predefined determined rules, then leverage crowd-sourcing to transform the canonical text into natural language. Traditionally, people first prepare the natural text and then transform them into logical forms. There the two steps require massive manual labeling. To solve this problem, this method wisely chooses a different direction for generating data pairs. It hugely decreases the burden of human resources and speeds up the whole process. More information can be found in the paper.

We adopt similar procedures but vary in ways of creating logic forms and natural utterances. Specifically, we adopt the creating process proposed in GAZP [16]. The method samples logical forms with respect to grammar (such as SQL) and then generates utterances according to logical forms.

3.4.2 Generate Aggregate Tables

To generate realistic aggregate tables, we make use of SQLs in the Spider dataset as a useful source because they are realistic queries that interest people.

We adopt two algorithms to create them. The first one is based on the SQL template. We transfer all SQLs in Spider dataset to the template, then fill them with proper contents from other databases into the new SQL and save the new SQL query results as a new aggregate table, finally, we give the table and columns names by rule-based and SQL-to-text model. The second method directly uses all SQLs from Spider dataset, and then filters out some detail condition that is unlikey to take place in the aggregate tables. Detailed description about the two algorithms can be seen at Algorithm 1 and 2.

3.4.3 Generate NL-SQL Pairs

To create SQLs on new aggregate tables, we follow two methods. Firstly, we make extensions on the SQL used to create aggregate tables and then transform them into SQLs on aggregate tables. Extension includes adding conditions, join//and/intersect/union with other tables. Secondly, we transform SQL into templates, then fill them with the contents of the aggregate tables.

To create text accordingly, we use the model of SQL2NL to transform SQLs back to text. We also transform it into SQL again to ensure cycle consistency.

4. EXPERIMENT

We next conduct experimental evaluation on SpiderWAGG. To facilitate simultaneous training of Spider and SpiderWAGG, we remain in our dataset the same train and validation splits of databases as Spider.

We build our novel model based on the GloVe version of the SOTA model RAT-SQL as our baseline, which is implemented by the PyTorch framework. Experiment settings are consistent with the original paper, readers who want to know more knowledge may refer to [5]. The pruning factor used by Dynamic Relation Pruning is 5, and the index of pruning layer in the second layer in the stack of attention layers.

Algorithm 1. Aggregate tables built with templates.

```
1: $d \leftarrow$ GetSample($AllDBs$)
2: $z \leftarrow \emptyset$
3: for $d' \in d$ do
4:     $t \leftarrow d'$.AlterColsToTemplates()
5:     if ConformToFilter($t$) then
6:         $z$.Add($t$.FilterWithRules())
7:     end if
8: end for
9: $z \leftarrow z$.TopKTemplatesByKmeans()
10: for $s \in z$ do
11:     $s \leftarrow s$.FillSQL()
12:     if ConformToRegularize($s$) then
13:         $s \leftarrow s$.RegularizeSQL()
14:     end if
15: end for
16: $r \leftarrow$
17: for $s \in z$ do
18:     $c \leftarrow z$.RandRuleChoice()
19:     $v \leftarrow z$.BuildAggTableName($s, c$)
20:     $r$.Add($z$.BuildAggTable($s, v$))
21: end for
22: return $r$
```

Algorithm 2. Aggregate tables built with original SQL.

```
1: $d \leftarrow$ GetSample($AllDBs$)
2: $z \leftarrow \emptyset$
3: for $d' \in d$ do
4:     if ContainManyQualifiers($d'$) then
5:         Continue
6:     end if
7:     $z$.Add($d'$)
8: end for
9: for $s \in z$ do
10:     if ConformToRegularize($s$) then
11:         $s \leftarrow s$.RegularizeSQL()
12:     end if
13: end for
14: $r \leftarrow$
15: for $s \in z$ do
16:     $c \leftarrow z$.RandRuleChoice()
17:     $v \leftarrow z$.BuildAggTableName($s, c$)
18:     $r$.Add($z$.BuildAggTable($s, v$))
19: end for
20: return $r$
```

4.1 Dataset Analysis

Our dataset SpiderwAGG has 406 aggregate tables and 9,752 NL-SQL pairs. We summarize the NL-SQL statistics of SpiderwAGG and the Spider dataset in Table 2. As demonstrated in the table, we have comparable distributions on all indicators, which means we share similar SQL complexities with Spider dataset, but with significantly less manpower. Table 1 shows some SQL query examples given the database name.

Table 1. SQL-NL example from SpiderwAGG.

DB ID consert singer
Question how many singer are there for country?
SQL select count (*) from count singer there for each country DB ID consert singer
Question how many singer are there for each age?
SQL select T1.age, count single from singer as T1 join count singer there as T2 DB ID employee hire evaluation
Question how many employee live from each city?
SQL select T1.City, count employee from employee as T1 joincount employee we have as T2

Table 2. Statistics and comparisons of our dataset SpiderwAGG and Spider.

Dataset	#Sample	#Q	#SQL	#DB	#Domain	#Table/DB	GROUP BY	AS	ON	JOIN	INTERSECT
Spider	8659	4714	4713	146	138	5.1	1110	2029	2044	2046	145
SpiderwAGG	8671	6248	6248	145	138	5.1	1056	2778	1010	2778	234

Table and column names are critical for correctly linking, as shown in Figure 3, we find that new scenario brings 30% increase of average number of candidates to link and 10% increase of average encoding length.

4.2 Metrics

We adopt the new metric proposed in [23], which utilizes a test suite for evaluating accuracies to approximately compute semantic accuracy for SQLs of the same semantics. They first distill a small test suite of databases and then execute SQLs on them. Then at evaluation time, they could compare the execute results of gold SQL and predicted SQL. The metric gives a tight upper accuracy.

4.3 SpiderWAGG Results

Table 3 shows the accuracy of our model WAGG and RAT-SQL baseline in each difficulty level data pair and total data pair. WAGG exceeds the baseline above by a large margin of 5% of exact match accuracy and 3.8% of execute accuracy. As the difficulty of SQL increases, the performance of our model will also decrease. A point worthy of mention is that WAGG does not surpass RAT-SQL for all hardness, a possible explanation is that relation pruning makes more mistakes as hardness increases. A more accurate pruning strategy should be invented in the future.

Table 3. Exact match (EM) accuracy and execute accuracy (EX) on SpiderWAGG.

	EM Accuracy	EM Accuracy	Easy EM	Medium EM	Hard EM	Extra hard EM
RAT-SQL(baseline)	42.7	54.9	64.7	44.3	60.7	53.3
WAGG	**47.7**	**58.7**	71.6	46.3	57.4	46.7

Table 4 shows the ablations using different modules in our model on SpiderWAGG. Dynamic pruning makes statistically significant improvements by 2.7 points. The relation selection module further improves by 1.1 points. To evaluate performance of the new model on the original dataset, we further do an experiment on Spider. Table 5 shows accuracy on the Spider and SpiderWAGG dataset. WAGG has a minor increase on the Spider dataset, which is not surprise because of less ambiguity.

Table 4. Accuracy of WAGG ablations on the SpiderWAGG dev set.

	EM Accuracy	EX Accuracy
WAGG	**47.7**	**58.7**
WAGG w/o relation selection	46.8	57.6
WAGG w/o dynamic prune	42.7	54.9

Table 5. EM accuracy on the Spider and SpiderWAGG dataset. The value of RAT-SQL is taken from the Spider leader-board.

on Spider	on Spider	WAGG
RAT-SQL	62.7	54.9
WAGG	**63.2**	**57.6**

4.4 Discussions

4.4.1 Dynamic pruning

WAGG applies dynamic pruning module to reduce inference time. Figure 6 demonstrates the comparison of inference time of 200 queries in different hardness between RAT-SQL and our model. From the figure, we can find out that our model speeds faster than RATSQL on all hardness. Since the lengths of input sequences are reduced to $c * Log(L)$, the inference time does not vary among hardnesses on our model. We also inspect how different index of inserting dynamic prune will infect the result (see Table 6), where the prune layer *index* = 0 means inserting dynamic pruning at the very first layer. The best index is 2. A possible explanation to this result is that values on the first layer are not yet contextualized with meaningful information, so pruning too early could cause a drop of performance.

Table 6. Accuracy of different dynamic prune layer index on the SpiderWAGG dev set.

	EM Accuracy	EX Accuracy
WAGG, prune layer=0	45.2	56.2
WAGG, prune layer=2	46.8	57.6
WAGG, prune layer=4	45.8	56.4

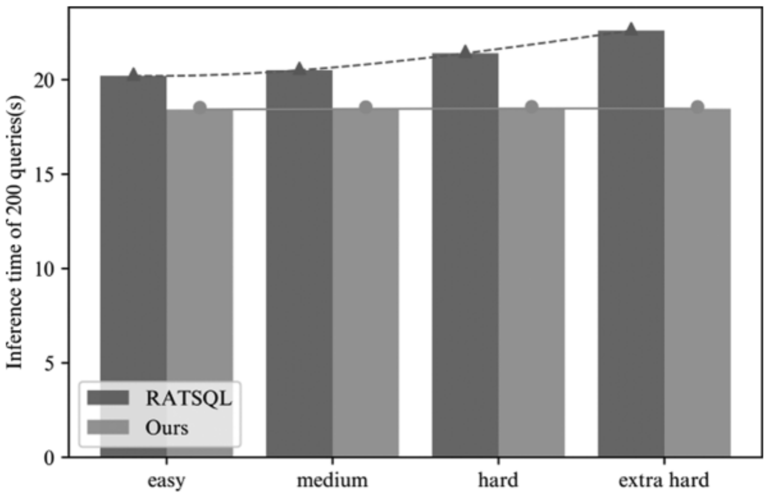


Figure 6. Our model speeds faster than RATSQL on all hardness. Since the lengths of input sequences are reduced to $c * log(L)$, the inference time does not vary among hardnesses on our model.

4.4.2 Error Analysis

We randomly select 50 samples of mis-predicted SQL and found that the errors can be classified into two kinds: 64% of incorrect column or table in SQL and 36% requiring extra knowledge. The first error results from wrong relation selections or exessive dynamic pruning. The main reason is that some questions can be expressed by multiple SQLs, so there should be different group of relations, meaning there is a correlation between the relation of two words and the other words. Regarding whether this model can produce group relations, we plan to do more research in the future.

For the second error type, it could be potentially solved by defining rule-based operator mappings or add pre-training models that has the knowledge.

5. CONCLUSION

In this paper, we discover a new scenario for text-to-SQL that is ignored by current text-to-SQL community that brings new challenges. To inspect on this, we construct a large scale dataset SpiderwAGG for the scenario based on Spider dataset. Methods of constructing rely lightly on human resource, largely exploit the capability of machine automation. To solve the challenges, we propose a new model for the new scenario, which has two innovative modules: relation selection and dynamic pruning. Experiments show our model has mitigated the challenges.

We also point out drawbacks that we plan to optimize in the future. Since the dataset is built fully by automation, there are some irrational items. For example, there is SQL doing aggregation on id. We plan to include human resource to check the meaningfulness.

AUTHOR CONTRIBUTIONS

S.Q. Li (1931957@tongji.edu.cn) discovers the problem, design and implement the total framework, and does experiments. K.B. Zhou (kb824999404@tongji.edu.cn) and Z.Y. Zhuang (flash@tongji.edu.cn) implement and run the code for constructing dataset, do analysis on it and do helped the overall experiment. H.F. Wang (carter.whfcarter@gmail.com) gives many solid advice for the whole paper from the begining to the end, and helps to proofread the paper. J. Ma (majun tongji@foxmail.com) gives advice.

ACKNOWLEDGEMENTS

Many thanks to my supervisor Mr.Wang for giving me much advice and opportunities. Also thanks to Huifang Du, Xing Huang, Siqi Wang in our laboratory for the open discussions.

REFERENCES

[1] Zhong, V., et al.: Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. In: arXiv preprint arXiv:1709.00103 (2017)
[2] Yu, T., et al.: 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, pp. 3911–3921 (2018)
[3] Wang, L., et al.: DuSQL: A Large-Scale and Pragmatic Chinese Text-to-SQL Dataset. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, pp. 6923–6935 (2020)
[4] Sun, N., et al.: TableQA: a Large-Scale Chinese Text-to-SQL Dataset for Table-Aware SQL Generation. In: arXiv preprint arXiv:2006.06434 (2020)
[5] Wang, B., et al.: 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, pp. 7567–7578 (2020)
[6] Hu, W., et al.: Service-oriented Text-to-SQL Parsing. In: Findings of the Association for Computational Linguistics: EMNLP 2020, pp. 2218–2222 (2020)
[7] Sen, J., et al.: ATHENA++: natural language querying for complex nested SQL queries. In: Proceedings of the VLDB Endowment, pp. 2747–2759 (2020)
[8] Wang, C., et al.: Robust Text-to-SQL Generation with Execution-Guided Decoding. In: arXiv preprint arXiv:1807.03100 (2018)
[9] Ralph Kimball: Margy Ross. The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling (Second ed.). Wiley Computer Publishing, p. 356. (2002)
[10] Lin, X., et al.: Bridging Textual and Tabular Data for Cross-Domain Text-to-SQL Semantic Parsing. In: Proceed ings of the 2020 Conference on Empirical Methods in Natural Language Processing, pp. 4870–4888 (2020)
[11] Zhou, H., et al.: Informer: Beyond efficient transformer for long sequence time-series forecasting. In: Proceedings of the AAAI Conference on Artificial Intelligence, 35(12), 11106–11115 (2020)
[12] Devlin, J., et al.: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: NAACL-HLT 2019, pp. 4171–4186 (2018)
[13] Yin, P., et al.: TABERT: Pretraining for Joint Understanding of Textual and Tabular Data. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp. 8413–8426 (2020)

[14] Bogin, B., et al.: Representing schema structure with graph neural networks for text-to-SQL parsing. In: 57th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, pp. 4560–4565 (2021)

[15] Vaswani, A., et al.: Attention is all you need. In: Advances in Neural Information Processing Systems, pp. 5999–6009 (2017)

[16] Gu, Y., et al.: Beyond I.I.D.: Three levels of generalization for question answering on knowledge bases. In: Proceedings of the 30th International Conference on World Wide Web, pp. 3477–3488 (2020)

[17] Wolfson, T., et al.: BREAK it down: A question understanding benchmark. In: Transactions of the Association for Computational Linguistics, 8, 183–198 (2020)

[18] Baik, C., et al.: Duoquest: A Dual-Specification System for Expressive SQL Queries. In: 2020 ACM SIG-MOD/PODS Conference, pp. 2319–2329 (2020)

[19] Chen, W., et al.: HybridQA: A dataset of multi-hop question answering over tabular and textual data. In: Findings ofthe Association for Computational Linguistics: EMNLP 2020, pp. 1026–1036 (2020)

[20] Wenhu, C., et al.: Open Question Answering over Tables and Text. In: International Conference on Learning Representations 2021, pp. 2319–2329 (2021)

[21] Yin, P., et al.: A syntactic neural model for general-purpose code generation. In: Annual Meeting of the Association for Computational Linguistics, pp. 440–450 (2017)

[22] Wang, Y., et al.: 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), pp. 1332–1342 (2015)

[23] Zhong, R., et al.: Semantic evaluation for Text-to-SQL with distilled test suites. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, pp. 396–411 (2020)

AUTHOR BIOGRAPHY

Shuqin Li is a graduate student at the College of Design and Innovation, Tongji University. She is interested in artificial intelligence, especially its applications in semantic parsing and conversational system.
ORCID: 0000-0001-9909-5449

Kaibin Zhou is a undergraduate student at the School of Software, Tongji University. He is interested in artificial intelligence, especially its applications in natural language processing and computer graphics.

Zeyang Zhuang is an undergraduate student at the School of Software Engineering, Tongji University. He is interested in natural language processing, especially its application of lexical analysis and text generation in complex scenes.

Haofen Wang has long served as Chief Technology Officer in a first-line artificial intelligence company and has rich experience in AI R&D management. He is one of the founders of OpenKG, the world's largest Chinese open knowledge graph alliance. He is responsible for participating in a number of provincial and ministerial AI-related projects, and has published more than 90 high-level papers in the AI field, which have been cited more than 2,000 times and the H-Index has reached 21. He built the world's first interactive virtual idol—"Amber·Xuyan"; the intelligent customer service robot he built has served more than 1 billion users. Currently, he is the Deputy Director of the Terminology Committee of China Computer Society, the Deputy Secretary General of the Language and Knowledge Computing Committee of Chinese Information Society and distinguished research fellow in College of Design Innovation, Tongji University.
ORCID: 0000-0002-0672-8081

Jun Ma Double Employed Professor of College of Design & Innovation, and School of Automotive Studies, of Tongji University. He is the head of the innovation design and entrepreneurship discipline of the School of Design and Innovation, and uses "design-driven innovation" and "design+" to help the transformation and upgrading of the industry. He also serves as the "Plug and Play" entrepreneurship mentor, the chairman of Rhine-Main Sino-German Innovation Center, the member of Tencent Think Tank, the head of Tongji-PSA Innovation Stellab. He adopts natural language applications on design & innovation.