NL2SQL:BERT-BASED MODEL FOR SQL GENERATION

Tinglong Liao (tl2564), Xue Bai (xb347), Alison Yao (yy2564)
CSCI-SHU 376 Natural Language Processing
NYU Shanghai

ABSTRACT

To empower non-programmers to interact with databases easily, our project aims to convert Chinese questions in natural language into corresponding SQL commands accurately. In this report, we document the design and performance of two BERT-based models and their variants to tackle the NL2SQL problem. We conclude that our second model, the Double Tower model, performs the best so far and further suggestions for improvement are given.

1 INTRODUCTION

Among programmers, computers are notoriously infamous for being fastidious about command integrity. That is why natural language is insufficient as computer commands to extract data from databases and why Computer Science and Data Science students at NYU Shanghai are encouraged to take the course Databases so that we can learn the correct SQL syntax to perform such tasks. In hope of empowering more non-programmers to interact with SQL databases and more efficient use of stored data, this project is designed to convert natural language into SQL commands (NL2SQL) accurately and efficiently.
With the rise of Conversational User Interfaces (CUI) such as voice assistants and chatbots, Semantic Parsing became a hot research topic in the field of Natural Language Processing, in which extrapolating formal meaning representation from natural language has attracted increasing attention. Among various natural languages, we chose Chinese as our context language; among tasks such as NL2BASH, NL2Python, NL2Java and so on, we chose SQL as the target. We would like everyone to be able to talk to their voice assistant, ask a question, then get answers from the databases right away without having to write any SQL queries themselves. Therefore, questions described in natural language and corresponding SQL statements are used to train, validate and test NL2SQL models. We compare 2 BERT-based models in this project based on accuracy.

2 DATA DESCRIPTION

We obtained the dataset, TableQA, from GitHub provided by Zhuiyi Technology (Sun et al., 2020). The original dataset consists of train, val, test and final test datasets because there were two rounds of competition. We merge the val and test datasets to become our validation data and use the final test dataset as our test data. The ratio of train-val-test split is about 10:2:1. Each dataset consists of Chinese question and SQL pairs and the corresponding tables. Table 1 shows the number of questions and SQL answer pairs and the number of tables in each dataset.

Table 1: Dataset Statistics

	Q&A	**Tables**
Train	41,522	5,013
Val	8,482	2,299
Test	4,055	978

All three datasets include two files. The first file contains the id and name of the tables and the name, data type, and values of the columns in each table. The second file contains the questions in plain

Chinese, the id of the corresponding table, and the SQL statements. Each SQL statement has four labels: the id of the selected columns, the aggregate functions associated with the selected columns (null, avg, max, min, count, sum), the relationship of conditions in the where clause (null, and, or), and a list of conditions. The list of conditions contains several sublists. Each sublist represents a condition, which is described by three labels: the id of the conditional column, the operation associated with the conditional column (>, <, ==, !=) and the value associated with the conditional column. An example is shown in Fig 1:

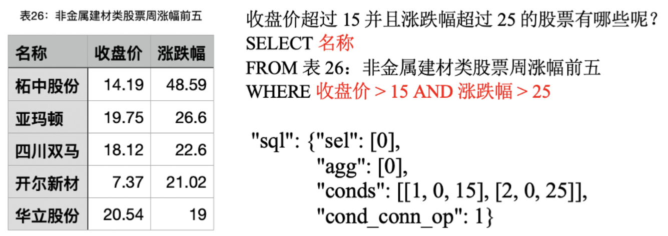

表26：非金属建材类股票周涨幅前五

名称	收盘价	涨跌幅
柘中股份	14.19	48.59
亚玛顿	19.75	26.6
四川双马	18.12	22.6
开尔新材	7.37	21.02
华立股份	20.54	19

```
收盘价超过 15 并且涨跌幅超过 25 的股票有哪些呢？
SELECT 名称
FROM 表 26：非金属建材类股票周涨幅前五
WHERE 收盘价 > 15 AND 涨跌幅 > 25

 "sql": {"sel": [0],
         "agg": [0],
         "conds": [[1, 0, 15], [2, 0, 25]],
         "cond_conn_op": 1}
```

Figure 1: Data Example

Fig 2 is the dictionaries for the aggregate functions, the operators and the connecting operation.

```
agg = {0:"", 1:"AVG", 2:"MAX", 3:"MIN", 4:"COUNT", 5:"SUM"}
op = {0:">", 1:"<", 2:"==", 3:"!="}
conn = {0:"", 1:"and",  2:"or"}
```

Figure 2: Operation Dictionary

This example is a relatively simple one, the tables and the SQL queries can be much more complicated. The table characteristics are are shown in Fig 3 and Fig 4.

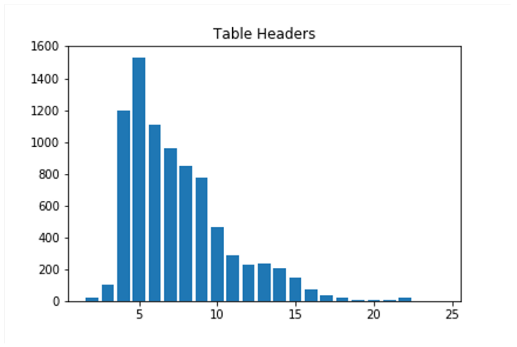

Figure 3: Table Header Frequency

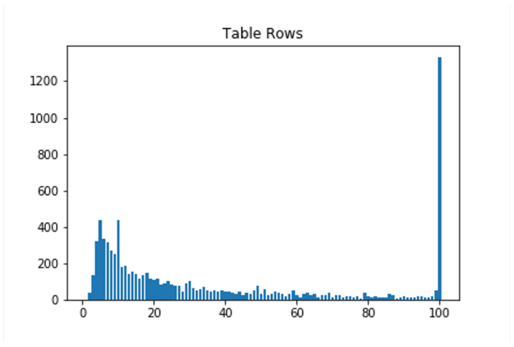

Figure 4: Table Row Frequency

The number of headers ranges from 3 to 23 and the number of rows can be as much as 100. The query characteristics are shown in Fig 5 and Fig 6.

The number of columns in the SELECT clause ranges from 1 to 3 and the number of conditions in the WHERE clause ranges from 1 to 4. However, the imbalance of the classes poses plenty of challenges to this project and hurts the classification performance of the underrepresented class.

3 Related Work

Many researchers applied different models on this dataset, among which M-SQL (Zhang et al., 2020) performs the best according to their evaluation results. M-SQL concatenates the question and the headers of the column together to form a long input. A BERT encoder is used to generate the hidden state of the input. The model formulates the problem not as a translation problem, but as a slot-filling classification problem. The task is decomposed into 8 subtasks: SELECT header

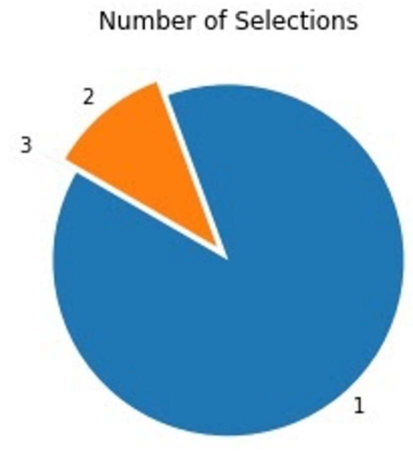

Figure 5: Selection Ratio

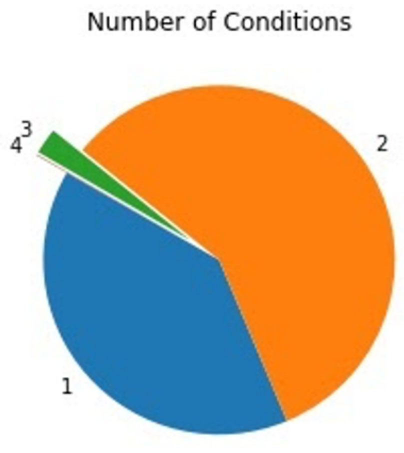

Figure 6: Condition Ratio

number, SELECT headers, SELECT header aggregate functions, WHERE column number, WHERE columns, WHERE column operators, WHERE condition connection and WHERE condition value. The information of each column is extracted using the hidden states of every token of that column's header.
While using every token is intuitive, it also requires much computation. Since the BERT encoder uses the Attention mechanism, we believe that the hidden state of the start token of the column will be sufficient to capture the relevance between the question and the column. Moreover, a more intuitive way is to encode the question and the headers of the columns separately and connect them together using the dot product. Although it is even more computationally heavy, it is also likely to have a better performance. Therefore, we propose these two models in our paper, one more efficient, and the other more accurate.

4 Models

Among the subtasks, picking the values in the WHERE clause conditions is the most challenging. Considering the time constraint, we left this one out and tackled the rest of the subtasks. Here, we present two BERT-based models and their variants to conquer this challenge.

4.1 Single-BERT model

Our first model draws inspiration from the winning M-SQL model in the competition created by Zhang et al. (2020) that is discussed in Section 3 Related Work and employs a single BERT multi-task learning model.
For the input, we concatenate the Chinese question and all the headers in the corresponding table with [CLS] as the beginning token and [SEP] as the separator. Then, as usual, we also have type embeddings that specify whether or not the part of the sequence is question or headers; we have the position embeddings that highlight the differences of position. An example is shown in Fig. 7.

Then, we pass the input sequence into the Chinese pretrained BERT, hfl/chinese-bert-wwm-ext (Cui et al., 2019). This pretrained BERT employs whole word masking on Chinese phrases instead of characters like Google's Chinese BERT and achieves higher performance on multiple tasks.
Among the encoded h vectors, $h_{[CLS]}$ and $h_{[SEP]}$ are picked out for the following classification tasks. $h_{[CLS]}$ is used to predict the number of columns in the SELECT clause, the number of conditions in the WHERE clause and the operator in each WHERE clause. The $h_{[SEP]}$ vectors are used to predict the headers in the SELECT clause, the aggregate functions for each header, the headers in the WHERE clause and the operator after each header name.

Fig. 8 gives us more detailed information. The number of headers in the SELECT clause and the operator in each condition in the WHERE clause are classification problems of three classes; the number of conditions in the WHERE clause is that of 4 classes, so the logits output is of size (B,

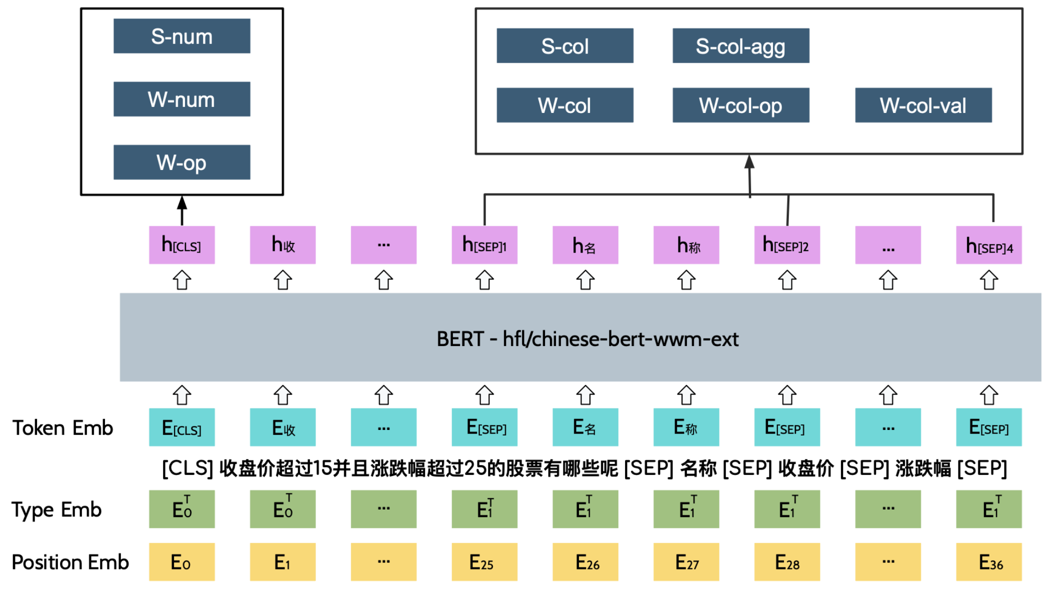
S-num
W-num
W-op
S-col
S-col-agg
W-col
W-col-op
W-col-val
h[CLS] h收 ··· h[SEP]1 h名 h称 h[SEP]2 ··· h[SEP]4
BERT - hfl/chinese-bert-wwm-ext
Token Emb
E[CLS] E收 ··· E[SEP] E名 E称 E[SEP] ··· E[SEP]
[CLS] 收盘价超过15并且涨跌幅超过25的股票有哪些呢 [SEP] 名称 [SEP] 收盘价 [SEP] 涨跌幅 [SEP]
Type Emb
Position Emb
E0 E1 ··· E25 E26 E27 E28 ··· E36

Figure 7: Single-BERT model structure

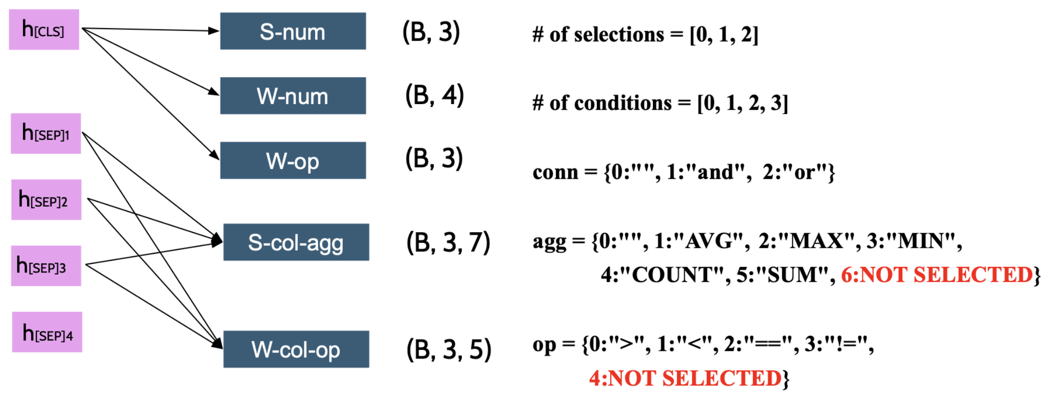
h[CLS]
h[SEP]1
h[SEP]2
h[SEP]3
h[SEP]4
S-num (B, 3) # of selections = [0, 1, 2]
W-num (B, 4) # of conditions = [0, 1, 2, 3]
W-op (B, 3) conn = {0:"", 1:"and", 2:"or"}
S-col-agg (B, 3, 7) agg = {0:"", 1:"AVG", 2:"MAX", 3:"MIN", 4:"COUNT", 5:"SUM", 6:NOT SELECTED}
W-col-op (B, 3, 5) op = {0:">", 1:"<", 2:"==", 3:"!=", 4:NOT SELECTED}

Figure 8: Size Details

3) and (B, 4) respectively. The last encoded [SEP] vector at the end of the sequence is not used in classification, but all the rest encoded [SEP] vectors are utilized for classifying aggregate functions in the SELECT clause and the operators in the WHERE clause. The former is a 7-class classification because we add one more class where the header is not selected; the latter is a 5-class classification for the same reason. In this example, there are three [SEP] vectors, thus the shape of S-col-agg logits output is (B, 3, 7) and the shape of W-col-op logits output is (B, 3, 5). Predicting S-num and W-num is kept to help predict the selected columns.
Since we have multiple subtasks, we experimented with grouping them differently. Here, we present 3 variants: the full variant that trains all subtasks at once (SB_multitask), the subset variant that combines S-num, S-col and S-agg (SB_SELECT) and another subset variant that trains W-num, W-col, W-col-op and W-conn-op (SB_WHERE).

4.2 Double Tower model

Unlike Single-BERT where we have 7 subtasks, the Double Tower model only has 5. This approach encodes the question and the table separately. For question text, we use a standard BERT model to encode the message. For table content, we encode each column in parallel, passing each table header into the same table encoding the BERT model. After encoding, the shape of the encoded question

is (B, N) and the shape of the encoded table content is (B, H, N), where B is the batch size, H is the number of headers in the table, and N is the length of the encoded message.
After encoding, we construct two types of headers for classification. The first header deals with multi-label subtasks including columns in the SELECT clause and columns in the WHERE clause. For each table column, this header computes the dot product of the question and the column and uses this logit to determine whether to select this column. The second header deals with multi-class classification subtasks, including aggregation operators, connection operators, and condition operators. This header projects the encoded question and encoded header onto a vector and uses a linear layer to compute the logits.
For the model, we could choose to train all the sub-task in one model or train a model for each or several tasks together. And we add the losses of all the subtasks to be the model loss function and use that to update the model accordingly. In our project, we experimented with three models, one model for training all subtasks (DT_multitask), one model for training SEL and AGG clause (DT_SELECT), and one model for training condition-related subtasks (DT_WHERE). We refer to the first model as the full double tower variant and the latter two as subset variants. The intuition behind combining SEL and AGG in one model and condition-related model in another model is that the part of the information that is related to SEL and AGG is in one position, while the information regarding conditions is in another part of the sentence. We want to see if dividing the two parts can hurt or improve the performance. As an example, the structure for the multi-task double tower model is shown in Fig 9.

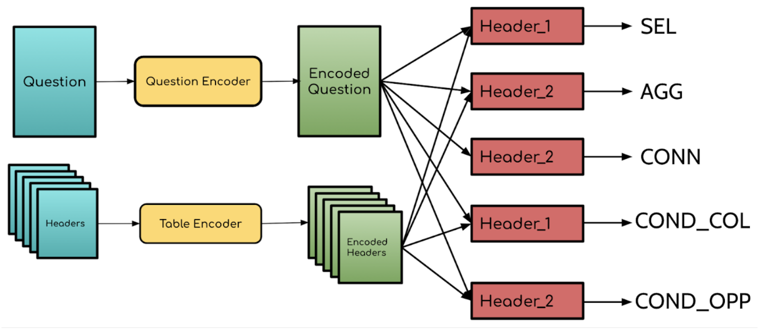

Figure 9: Double Tower Model

5 Evaluation & Results

Since all the 5 subtasks we implemented here can be framed as a classification problem, we can certainly use a confusion matrix to log their accuracy, precision and recall. However, since in our case, we do not have any preference regarding higher precision or higher recall, we will focus on accuracies only. The unit for calculating accuracies is the SELECT clause or WHERE clause, not individual header or column or operator. For example, only if the model gets all the header names in the SELECT clause correctly do we consider it correct. This is a major change from our previous presentation because we believe it makes more sense in terms of answering users' questions correctly.
The first model, Single-BERT Multitask Model, has two extra subtasks that predict the number of SELECT headers and the number of WHERE conditions, which are auxiliary to predicting which headers and columns are chosen in the clauses. The results are listed in Table 2.

The performance of the SELECT subtasks, especially SELECT headers improves if the SELECT clause is trained separately while the WHERE subtasks make little difference. This is probably because more words are used in the question to describe WHERE conditions, thus training the weights for selected headers is harder. As for implementation, all three Single-BERT variants take approximately 20 minutes for each epoch and converge after 5-6 epochs. Therefore, implementation wise, separating the SELECT clause may be a good idea to achieve better performance for it does not cost too many extra resources.

Table 2: Single-BERT Results

	SB_multitask	SB_SELECT	SB_WHERE
S-num	97.29%	97.34%	/
S-col	50.43%	58.69%	/
S-agg	94.33%	96.47%	/
W-num	89.26%	/	88.03%
W-col	35.40%	/	35.35%
W-col-op	94.24%	/	94.59%
W-conn-op	89.92%	/	89.29%

For the Double Tower Models, the results of the three variations are in Table 3:

Table 3: Single-BERT Results

	TD_multitask	TD_SELECT	TD_WHERE
S-col	72.61%	75.5%	/
S-agg	98.24%	98.07%	/
W-col	61.63%	/	58.59%
W-col-op	96.31%	/	96.61%
W-conn-op	88.35%	/	/

It doesn't make too much difference whether or not we choose to train only the SELECT subtasks or the WHERE subtasks. The accuracies for DT_SELECT and DT_WHERE are close to those of variants. As for implementation, however, there is a stark difference between the Double Tower models. During training, all three Double Tower variants take approximately 1 hour for each epoch, even though DT_multitask is much more complicated in its header part. This is because most of the computation is consumed by the BERT encoder, not the actual header prediction. Single task models converge after 8-9 epochs, while the multi-task model converges after 20 epochs. But if we take into consideration the fact that two subset variants are to be trained for the four tasks, but only one full variant is needed to perform the five tasks, the multitask variant is more time-efficient. Similarly, in test time, the multitask model is more efficient than the two single tower variants combined.

To compare both models with the baseline, we list the accuracies Table 4. Please note that we obtained our baseline accuracies using the majority group. The baseline accuracy of S-agg is especially high because most cases do not use any aggregate function, which corresponds to 0:null. Similarly, nearly half of the cases in W-conn-op are 0:null.

Table 4: Results Comparison

	Baseline	Single-BERT-multitask	Double-Tower-multitask
S-col	17.93%	50.43%	72.61%
S-agg	88.31%	94.33%	98.24%
W-col	19.16%	35.40%	61.63%
W-col-op	22.59%	94.24%	96.31%
W-conn-op	48.11%	89.92%	88.35%

We can see that both Single-BERT and Double Tower models have much better performance than the baseline in every single subtask, but the best Double Tower performance is much better than the best Single-BERT model performance in S-col and W-col. It can be concluded that although the Bert encoder is already very powerful, the attention mechanism itself is not enough to extract the information from tokens far from each other. However, there is a tradeoff between training time and accuracy. The Double Tower model takes much longer to train but its performance is better.

6 Conclusion & Future Work

By the time of our presentation, we were still having a lot of trouble converging both of our models. We had a hard time watching our losses go down and then being disappointed again when it goes back up. Although we have tried various learning ratings such as 6.25e-5, 5e-5, 3e-5 and so on, the learning rate was not small enough. Once we started using 1e-5 as the learning rate, both models converge much better, so the BERT model is highly susceptible to the differences in the parameters. Our best sets of parameters are presented in the Appendix section. Also, how to combine and train our subtasks is very important. As we can see from the previous section, the results vary a lot when the subtask grouping is different.
Although our models have achieved relatively high accuracies in general and very high accuracies in certain subtasks, there is still a long way to go. We propose some suggestions for future improvements. First of all, we believe adding manual constraints will improve the overall accuracies. For example, if we already know that a column in the WHERE clause consists of strings only and no numerical values, it is very unlikely that the operator following is going to be $>$ or $<$. Rather, == and != are much more likely. In the subtask of value extraction that we did not implement here, it is also helpful if we can make certain rules to show the model that 18年 is the same as 2018 and that 企鹅 is the same as 腾讯. Of course, another BERT is necessary if we were to tackle the value extraction problem. In fact, we believe some of our fellow classmates' final projects can be retrained and incorporated into our model to fulfill this task. A second suggestion is that we can try weighting the losses with preference. In the current implementation, since we are doing multitask training, we add up the losses from each subtask to form one final loss in the training process. Therefore, we are giving equal weights to each of the subtask losses. It would be interesting to find out if giving certain subtasks higher weights will improve the performance.
In conclusion, we had a challenging yet rewarding experience building the NLP models to tackle a real-life problem. Although our models are not perfect, the process of topic choosing, project planning, model building, result presentation and teamwork is more educational. This project will be a solid foundation for our future journey in NLP.

References

Yiming Cui, Wanxiang Che, Ting Liu, Bing Qin, Ziqing Yang, Shijin Wang, and Guoping Hu. Pre-training with whole word masking for chinese bert. *arXiv preprint arXiv:1906.08101*, 2019.

Ningyuan Sun, Xuefeng Yang, and Yunfeng Liu. Tableqa: a large-scale chinese text-to-sql dataset for table-aware sql generation, 2020.

Xiaoyu Zhang, Fengjing Yin, Guojie Ma, Bin Ge, and Weidong Xiao. M-sql: Multi-task representation learning for single-table text2sql generation. *IEEE Access*, 8:43156–43167, 2020. doi: 10.1109/ACCESS.2020.2977613.

A Hyperparameters

Both models share most of the hyperparameters, except for the loss function. Single-BERT uses solely CrossEntropyLoss while Double Tower uses both CrossEntropyLoss and BCEWithLogitsLoss.

Table 5: List of Hyperparameters

Hyperparameter	**Value(s)**
batch_size	8
gradient_accumulation_steps	1
max_grad_norm	1.0
num_train_epochs	5-20
warmup_steps	0
criterion	CrossEntropyLoss/BCEWithLogitsLoss
optimizer	AdamW
learning_rate	1e-5
adam_epsilon	1e-8