Journal of Network Intelligence
Taiwan Ubiquitous Information
 ISSN 2414-8105 (Online)
Volume **6**, Number **4**, November **2021**

Generalizable semantic parsing for tables through object-aware and overlapping relation mapping

Xia Ye

Department of Computer Science
Xi' an Research Institute of Hi-Tech
Xi' an, 710038, China
yex_qing@126.com

Rui-Heng Liu*

Department of Computer Science
Xi' an Research Institute of Hi-Tech
Xi' an, 710038, China
Corresponding author: liuruih@foxmail.com

Zeng-Ying Yue

Department of Computer Science
Xi' an Research Institute of Hi-Tech
Xi' an, 710038, China
zengying2020@gmail.com

Jin-Jin Zhang

Department of Computer Science
Xi' an Research Institute of Hi-Tech
Xi' an, 710038, China
wangzhao@st.xatu.edu.cn

Received June 2021; revised August 2021

Abstract. *Mapping natural languages into query languages such as SQL that can be executed on structured tables has become an important part of semantic parsing research. Most existing methods use column names as table features and combine the syntax of the query language to design the model. However, for large tables, the scope of representation by column names is limited, and they ignore some common problems in practical applications, such as a column in a table is called repeatedly, keywords between natural language and table are not matching exactly, which makes it difficult to generalize these methods to real scenarios. In this paper, we propose a new semantic parsing model named TableSQL for database tables based on object-aware and overlapping relation mapping. First, we extract information about objects in the table that are related to natural language queries. Then, we integrate this information with inherent dimensional characteristics of the table to enhance the relational mapping between natural language queries and structured tables. We also implement overlapping relation mapping on the same column by defining special labels. Finally, we design a fuzzy matching method based on span prediction to alleviate the problem of incomplete keyword matching. Experimental results on the TableQA dataset show that the method significantly outperforms existing methods in terms of logical formal and execution accuracy.*
Keywords: Semantic parsing; Data retrieval; Query language; Database table structure

1. **Introduction.** Semantic parsing is a task of mapping natural language utterances to logically structured representations. In recent years, this work has been extended to the generalizable Natural Language Interface to Database (NLIDB) [1], where natural language queries from ordinary users are mapped to Structured Query Language (SQL) that can be executed on relational databases, and the database used in the test is not visible during training. This process can be understood as a Text-to-SQL task designed to facilitate the interaction between non-professional users and structured tabular knowledge, and can be applied to several scenarios such as cross-domain data retrieval and intelligent Q&A (Question and Answer). In this task, Chinese is a low-resource language [2] and has not been sufficiently studied. Unlike English, which has been extensively studied, Chinese has unique language forms and customs. In English, there are a large number of dummy words between real words to help identify the components of the sentence and there is a separator between words, while the structure of Chinese sentences is generally loose, sometimes focusing on the meaning rather than the form, and there is no separator between words. These lead to challenges for this research. Therefore, we focus on semantic parsing for tables in Chinese scenarios.

To fulfill this task, many existing methods understand it as an encoding-decoding process, where the encoding process is to obtain the mapping between natural language queries and database tables to analyze user query intent and decoding process requires generating executable SQL statements based on the results of previous process. Seq2SQL [3] models the task based on the inherent syntactic structure of SQL and uses a reinforcement learning strategy to generate query condition. SQLNet [4] divides SQL statements into SELECT and WHERE clauses, while refining the sketch of each clause to transform the Sequence to Sequence (Seq2Seq)[5] generation task into a sequence-to-set slot-filling task. SQLova [6] uses BERT [7] to obtain contextual features from queries and tables, which also investigates the effect of different decoders. IE-SQL [8] aligns and labels query sequences based on SQL statements, and learns the role of each sequence in the SQL statement by automatically generating annotations.

Query: 麻烦帮我查查在中国内地或在中国香港上映的3D电影都有那些啊?

English: Could you please help me to check which 3D movies are shown in mainland China or Hong Kong, China?

Table :

序号/ Number	影片名称/ Film	...	封装/ Package	地区/ Place	类型/ Type
1301-01	超人总动员2/ The Incredibles 2	...	2D	美国/ America	喜剧-动作-动画/ Comedy-Action-Cartoon
1301-03	狄仁杰 3/ Detective Dee 3	...	3D	中国内地-中国香港/ Mainland China-Hong Kong, China	动作-悬疑-武侠-古装/ Action-Suspense-Swordsmen-Costume
1301-04	黑色党徒/ BlacKkKlansman	...	2D	美国/ America	喜剧-传记-犯罪/ Comedy-Biographical-Crime
1301-08	我不是药神/ Dying to Survive	...	4K	中国内地/ Mainland China	剧情-喜剧/ Feature-Comedy
...					

SQL:
```
SELECT  影片名称/Film
FROM Table
WHERE  地区/Place = '中国内地-中国香港/Mainland China-Hong Kong,China'
       and 封装/Package = '3D'
```

Answer: 狄仁杰 3/Detective Dee 3

FIGURE 1. An example of semantic parsing for tables task.

Although these methods achieve promising results, they have three limitations. First, they all use column names in table as representatives to discover the mapping between natural language queries and database tables, while ignoring the table contents. This is intuitively natural, as the column names cover common attributes of table content, and the sheer volume of table contents are not conducive to organization and utilization. However, in practical scenarios, the queries raised by users are often colloquial and do not explicitly specify column names in table. As the example in Figure 1, giving a natural language query and a corresponding database table, generating the relevant SQL statement and executing them to get the answer. The darker parts of the query indicate the different attributes of the related objects and the darker rows in the table are query-sensitive candidate rows (Section 3.1.1). For example, the query is "麻烦帮我查查在中国内地或在中国香港上映的 3D 电影都有哪些啊?(Could you please help me to check which 3D movies are shown in mainland China or Hong Kong, China?)" , Combined with its corresponding SQL statement, we find multiple entities in the query establish mapping relationships with the table. These entities include "中国内地 (mainland China)" , "中国香港 (Hong Kong, China)" and "3D" , where "中国内地 (mainland China)" and "中国香港 (Hong Kong, China)" correspond to the column "地区 (Place)" in the table and the relationship between them is not obvious. The more serious one is that the relationship between "3D" and its corresponding column "封装 (Package)" is imperceptible. However, most existing methods use column names such as "地区 (Place)" , "封装 (Package)" , etc. to characterize the table in encoding process and ignore the concrete content of the table, which may be hard to represent the mapping between heterogeneous data. In addition, words with similar meanings often appear in the same table, such as "通用名 (generic names)" and "商品名 (trade names)" , "商户名称 (merchant names)" and "门店名称 (store names)" , "地区 (regions)" and "省份 (provinces)" , etc., which is easy to cause confusion on the selection of columns in the mapping process.

Second, in the process of generating the SQL statements, they do not take into account the fact that the same column in a table is called multiple times. This situation generally occurs in the WHERE clause. For the example in Figure 1, if user makes a query "哪些电影是 3D 或者 4D 的? (Which movies are 4K or 3D?)" on the table and its corresponding SQL statement is "SELECT 影片名称 (Film); From Table; WHERE 封装 (Package)= 4K or 封装 (Package)=3D" This is a simple and common problem, but what makes it special is that in WHERE clause, the column "封装 (Package)" is combined with "4k" and "3D" respectively to form the condition, in other words, the column "封装 (Package)" is used twice at the same time. However, most existing methods cannot generate such SQL statement, they usually calculate the probability value of each column in the table being called, and then pick the column with high probability. That means when they are selecting columns as conditions in the WHERE clause, each column has only one chance to be selected. Therefore, they cannot achieve the mapping of overlapping relations on the same column.

Finally, most of the existing works ignore the fact that the keywords for specified conditions mentioned in query may not exactly match the information in database tables. Especially in the practical application of Chinese scenes, the raised queries are often general, because users are not aware of the database details. Therefore, when expressing key information, some abbreviations, aliases and even misspellings often appear. For example, "索尼 (Sony)" in query corresponds to "索尼公司 (Sony Corporation)" in table, or "首都师范大学 (Capital Normal University)" to "北京师范大学 (Beijing Normal University)" , etc. To alleviate this problem, although some methods try to identify keywords through sequence annotation[8], this usually requires additional annotation work, so keyword information matching remains difficult.

To overcome the above problems, we focused on the TableQA dataset published in the literature [9], which is a Q&A task on database tables. We propose TableSQL, which is a new semantic parsing model for heterogeneous data. In the process of encoding, in order to solve the problem of inadequate use of tables, we try to incorporate table contents into a unified representation of two heterogeneous data from an object perspective. We consider each row of data in table contents as a description of different attributes of an object, and each column of data is a summary of different objects on the same attribute. In addition, multiple objects in a table are often involved in a user query. Therefore, we consider incorporating information about multiple objects related to the query in the encoding process. We first extract candidate rows from the table that are sensitive to user's query as object sets and treat each candidate row as an object. Then, we concatenate the attributes of each object extracted with the information of the column in which they are located. For each column in the table, we further distinguish different mapping relationships by introducing custom labels, which can implement overlapping relational mappings, and train the model to determine whether each column in the table needs to be called repeatedly based on the query. Horizontally, we use RoBERTa [10] as an initialization to encode the concatenated sequence jointly with query to capture mapping relationship between the query and columns based on a single object. Finally, in the vertical direction, we integrate information about different objects from same query by an alignment self-attention mechanism to fuse multiple objects attribute features on the same column in the table. For decoding, we design a generic SQL statement template and decouple it into different components. In the process of finding the keywords used for condition values in WHERE clause, we extract the correct keywords from the table to improve the accuracy of the query results by predicting the span of keywords in the query and using fuzzy matching. This can effectively deal with the difficulty of keyword matching. Specifically, in model training, we first slice the query sequence by n-gram. Then, we designed a fuzzy matching method to determine the span of keywords in the query by calculating the similarity of each candidate term to the keyword in ground truth. In model inference, the inverse process is executed, we first predict the span of keywords in the query and then finding the correct keywords in the database table by fuzzy matching methods. In order to enhance the syntactic connection between individual SQL components, we use a multi-task learning method so that different tasks optimize the same objective function during the training process. The results show that we have achieved 89.7% of the logical form accuracy and 92.0% of the execution accuracy on the TableQA dataset, which is a competitive result compared with the existing methods. In this paper, we make four contributions.

(1)We design TableSQL, a semantic parsing model for tables in Chinese scenarios. It can effectively address three limits in existing methods, which include inadequate use of tables, limited number of relationship mappings on the same column in table and difficulty in matching keywords.

(2)We propose an object-aware multi-dimensional table content enhancement encoding method, which enhances the perception and mapping capabilities between queries and tables while helping to remove the semantic ambiguity between the columns in table.

(3)We propose a custom labels method in encoding process, which implements overlapping relationship mapping on the same column in table, to solve the problem that existing methods cannot make repeated calls to the same column when generating WHERE conditions.

(4)We propose a fuzzy keyword matching method based on span prediction, which ensures that the generated SQL statements are executed accurately on the table by finding

the keywords related to query in table contents and the whole process does not require additional labeling.

The rest of this paper has the following structure. In the second section, we sort out related work. In the third section, we propose our methodology and illustrate each part of it in detail. Experimental details and analysis of results are presented in the fourth section. Finally, in the fifth section we summarize the full work.

2. **Related Works.**

Semantic analysis of heterogeneous data has become one of the research hotspots in Natural Language Processing (NLP) in recent years. Such as program code [11], SPARQL Protocol and RDF Query Language [12], SQL [13], etc. Semantic parsing of tables is the process of converting natural language queries into SQL statements that can be executed on a database to get the corresponding answer. With the creation of large-scale datasets such as WikiSQL [5], Spider [14] and Sparc [15]. Data-driven deep neural networks have become the dominant approach in the field.

Most of the existing works develop their model based on a Sequence to Sequence (Seq2Seq) framework and divide the whole task into encoding and decoding processes. Subsequently, many methods have improved on this basis. Dong et al. [16] obtained the state-of-the-art result on ATIS (5,410 queries to a flight booking system) and GeoQuery (880 language queries to a database of U.S. geography) datasets by introducing the attention mechanism. Wang et al. [17] uses the attention-based copying mechanism to intercept keywords from natural language queries during the decoding process, thus reducing the error rate of the traditional Seq2Seq method. TypeSQL [18] further considers the type of entity on the basis of SQLNet to improve the entity mapping process between utterance and table. In order to make full use of the relational database structure, different from the methods based on Seq2Seq, SyntaxSQLNet [19] and IRNet [20] use trees to represent the structured knowledge in database, while Global-GNN [21] and RAT-SQL [22] adopt a graph neural network method to model table features.

Previous models mostly design feature extractor based on Recurrent Neural Networks (RNN), Long-Short Term Memory (LSTM) and Graph Neural Networks (GNN) when encoding heterogeneous data. In recent years, pretraining technology provides an effective solution to the large-scale parameter learning in deep neural networks. Especially pretrained models [7, 23, 24] make significant progress in many NLP tasks. BERT is a fully connected neural network structure composed of multiple bidirectional Transformer [25] encoders stacked. It consists of two important components. The Masked Language Modeling (MLM) mechanism is designed to perform tasks similar to cloze filling and the Next Sentence Prediction (NSP) method is designed to determine the context relationship between two sentences. BERT achieves the new state-of-the-art performance on multiple different tasks. As Hwang et al. [6] mentioned, the use of complex BERT on Text-to-SQL tasks to improve performance requires careful design. The key to achieving this goal is to build a semantic analysis bridge between natural language and database table.

With the application of pretrained models, the effect of Text-to-SQL task has been further improved [6, 22, 26]. SQLova is a typical representative that uses BERT to encode natural language queries with table column names, and achieves performance beyond humans on the WikiSQL dataset. X-SQL [26] uses a similar approach, further combines the type information of the column, and adopts multi-task learning method to enhance the structured representation of downstream tasks. Wang et al. [22] propose a relationship-aware self-attention mechanism for database relational encoding and alignment of natural utterances and tables, while further improving the performance through BERT.

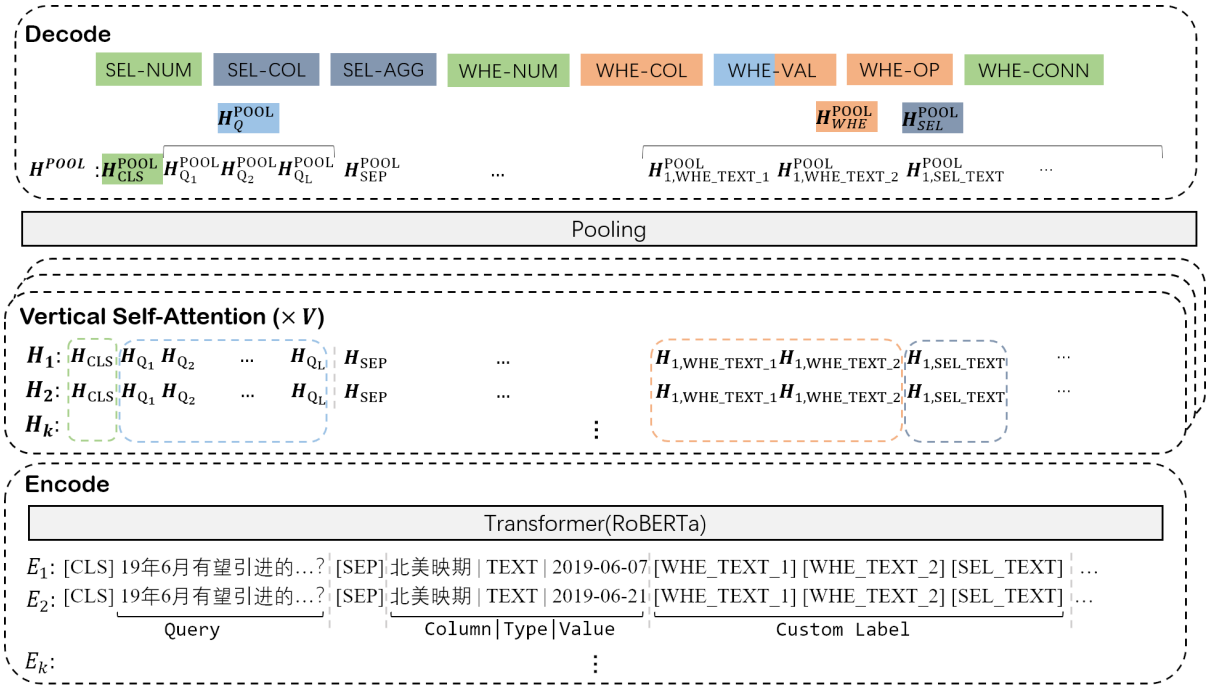

FIGURE 2. The structure of TableSQL.

Since there are few Chinese resources related to Text-to-SQL task, most of the existing works are carried out around English scenarios. In recent years, the proposal of large-scale Chinese datasets such as CSpider [2], DuSQL [27], and TableQA create conditions for research in Chinese scenarios. Sun et al. [9] pointed out that existing Text-to-SQL task datasets (such as WikiSQL) often assume that every natural language query can be answered in a given database table. Besides, the entity information between query and table can be completely matched. But this assumption cannot be extended to actual application scenarios. Therefore, for query answerability and entity linking problem, they propose a challenging large-scale Chinese dataset TableQA. They also found that models such as SQLOVA, X-SQL, Coarse-to-Fine [28], which perform well on the WikiSQL dataset have significantly lower performance on TableQA. This illustrates the challenge and necessity of studying Text-to-SQL task in Chinese scenarios.

3. **Methodology.** Our entire model architecture is shown in Figure 2. In encoding process, we combine a natural language query with the most relevant k rows in the candidate table separately for horizontal domain joint encoding. In vertical self-attention section, we facilitate the flow of semantic information on the same column through vertical alignment attention mechanism. In decoding process, each subtask uses the vectors at different positions after the pooling operation for the respective semantic parser. The inference process of the model can be understood as giving a natural language query and a database table, the model outputs the SQL statements that can be executed on the database. In this section, we present the details of the model, which can be divided into encoding and decoding processes.

In the process of encoding. First, we extract objects from table contents that are sensitive to the query raised by user to form a candidate table (Section 3.1.1). We calculate the edit distance of each cell to the query, and then the sum of edit distance of all cells in the same row is used as the sensitivity of the object represented by the row to the query. The composition of the candidate table is determined by the level of sensitivity. Then, we encode different candidate rows with the query separately in horizontal direction (Section

3.1.2). We design a sequence code for each cell value in the candidate row, which consists of the name of column in which the cell is located, the type of the column, cell value, and custom labels. The query is concatenated with all sequence codes of a single candidate row and fed to the encoder together. Finally, for each candidate row involved in the encoding result, the vectors corresponding to the special labels are fused in the vertical direction by an attention mechanism to achieve the integration of the same attributes among different objects (Section 3.1.3).

In the process of decoding, a generic SQL sketch is designed and decoupled into eight components, each of which is treated as a subtask. With multi-task learning we achieve uniform optimization of all subtasks and we will introduce these subtasks separately in Section 3.2.

3.1. **Joint Encoding of Query and Table Structure.** When representing two heterogeneous data, we first select query-sensitive rows from the table in database to build a candidate table, then integrate the horizontal and vertical semantic information in the table to ensure that it is fully utilized.

3.1.1. *Query-Sensitive Rows Extraction.* Semantic parsing for tables requires mapping relationships between natural language queries and database tables. However, a natural language query usually only involves a few rows of objects in a table which contains a lot of data. If all table contents are used for feature learning, many irrelevant information may not only bring noise, but also causes difficulties to the encoding processing. Therefore, it is necessary to filter the original table.

We propose a simple and effective row extraction strategy. As shown in Figure 1, we filter table based on attributes of different objects mentioned in the natural language query to get query-sensitive rows, which are used to form a candidate table. Giving a natural language query q and a corresponding table T. We calculate the edit distance [29] between each table content cell p in T and q according to Levenshtein distance [30], which is denoted as $D_{p,q}(m,n)$. m, n denote the indexes of the characters in p and q respectively, Where $m \in \{1,...,|p|\}$, $n \in \{1,...,|q|\}$, and $|p|$, $|q|$ denote the string length of p and q respectively. The calculation follows the process.

$$D_{p,q}(m,n) = \begin{cases} \max(m,n) & , \min(m,n) = 0 \\ \min \begin{pmatrix} D_{p,q}(m-1,n) + C_{del}(p_m) \\ D_{p,q}(m,n-1) + C_{ins}(q_n) \\ D_{p,q}(m-1,n-1) + C_{sub}(p_m,q_n) \end{pmatrix} & , \min(m,n) \neq 0 \end{cases} \tag{1}$$

Where p_m denotes the m-th character in p. Similarly, q_n denotes the n-th character in q. C refers to the cost function of operating on characters, which generally takes the value 1. $C_{del}(x)$, $C_{ins}(y)$, $C_{sub}(x,y)$ respectively represent the cost of deleting x, inserting y, and substituting x for y. Note that when $x = y$, $C_{sub}(x,y) = 0$.

Then, accumulate the edit distance of all cells in each row to get similarity score S between the row object and q. We sort the similarity scores of all rows and select the top k rows with the highest scores to form a final candidate table.

$$S_i = \sum\nolimits_{j=1}^{T_b} D_{p_j^i,q}(m,n) \tag{2}$$

Here, S_i refers to the similarity score of the i-th row. p_j^i denotes the j-th cell of the i-th row in table T, and $i \in \{1,...,T_a\}$, $j \in \{1,...,T_b\}$. T_a, T_b refer to the number of rows and columns in table T content respectively.

3.1.2. *Horizontal Cross-Structural Joint Encoding.* We jointly encode candidate tables and natural language queries on the horizontal domain. First, we use the same method as TABERT [31] to create serialization codes for the data in candidate table corresponding to different queries. Specifically, the cell information in table is represented in a linear sequence of column name, column type and cell value, they separated by a special symbol "|". The sequence construction follows.

$$Column\,|Type|\,Value \tag{3}$$

Where $Column$ represents the column name, $Type$ represents the column type, and $Value$ represents the value of the cell.

Most of existing works adopt independent modeling methods when selecting columns from the table [4, 32]. Specifically, a binary classifier is designed for each column to predict whether the column is selected, but this independent modeling approach is not only difficult to deal with the repeated use of columns in SQL clauses, but also splits or even lose the correlation between the columns during training and prediction.

In response to the above problems, we propose a new custom labels method to model this situation. From TableQA, we find that in SELECT clause, the same column in table can only be selected once at most. In WHERE clause, the same column in table is used twice at most. Therefore, in order to compare and select a particular column multiple times in the global space. We set different labels for each column. Specifically, an example as shown in Figure 2, three custom labels are added after the linearization sequence of each column. For a query and each row of data in the candidate table, it is expressed in the following way.

$$\begin{aligned}&[\text{CLS}], Q_1, Q_2, ..., Q_L, [\text{SEP}],\\ &COL_1|TYPE_1|VAL_1, [WHE\_TYPE_1\_1], [WHE\_TYPE_1\_2], [SEL\_TYPE_1], ...,\\ &COL_c|TYPE_c|VAL_c, [WHE\_TYPE_c\_1], [WHE\_TYPE_c\_2], [SEL\_TYPE_c], [\text{SEP}]\end{aligned} \tag{4}$$

Where Q_i indicates the i-th character in query Q, L indicates the length of query and $i \in \{1, ..., L\}$. COL_j indicates the name of the j-th column in table, $j \in \{1, ..., c\}$ and c is the number of columns. $TYPE_j$ indicates the value type of the j-th column and $TYPE_j \in \{\text{TEXT}, \text{REAL}\}$, which denotes text and number column types. VAL_j is the value corresponding to the j-th column in the candidate row. $[WHE\_TYPE_j\_1]$, $[WHE\_TYPE_j\_2]$, $[SEL\_TYPE_j]$ are custom labels. [CLS] and [SEP] are special labels defined by RoBERTa. [CLS] is used to represent the input global semantic information, [SEP] is used to separate queries and tables.

Custom labels can serve the following three purposes. First, they separate different columns in a table and ensure independent input of columns. Second, it gives a representation of how each column is used in SQL statements. In the three labels for each column, $[WHE\_TYPE_j\_1]$ means the first time selected by WHERE clause, $[WHE\_TYPE_j\_2]$ means the second time selected by WHERE clause, and $[SEL\_TYPE_j]$ means selected by SELECT clause. Different labels capture corresponding semantic information according to their meanings during training. This method provides a global perspective for the comparison of columns, and avoids the local effect caused by the independent modeling of each column. Third, in the process of constructing custom labels, we also distinguish the types of column by $TYPE_j$, which means columns of type TEXT and REAL have different labels. This approach can input the column type information into encoder as a kind of prior knowledge, so as to further learn the attributes between different columns during the training process.

As shown in Figure 2, according to a natural language query and a query-sensitive candidate table containing k rows, we construct input sequences $[E_1, E_2, ..., E_k]$ and send

them to a Transformer encoder initialized with RoBERTa in turn to obtain the hidden layer vector $[\mathbf{H}_1, \mathbf{H}_2, ..., \mathbf{H}_k]$. Where $\mathbf{H}_i \in \mathbb{R}^{|E_i| \times d}$, $|E_i|$ indicates the length of the sequence E_i, d denotes the output dimension and $i \in \{1, ..., k\}$.

3.1.3. *Attribute Fusion Based on Vertical Alignment Attention Mechanism.* We expand joint encoding representation of a natural language query and a database table. First extracting k rows query-sensitive objects to form a candidate table. Then in horizontal domain, each row in the candidate table is jointly encoded with the query. Finally, the corresponding hidden layer representation of k rows is obtained. However, this approach fragments semantic information of the same attributes between different objects in the candidate table and this expansion leads to a query corresponding to multiple hidden layer representations, so they need to be integrated.

In order to enable the semantic information to flow in the vertical direction. For different encoding outputs $[\mathbf{H}_1, \mathbf{H}_2, ..., \mathbf{H}_k]$ of the same query, we design a V-layer stacked vertically aligned self-attention module to fuse the semantic characteristics of different objects (rows) under the same attribute (column) in table, as shown in Figure 2. Unlike TABERT, to reduce the computational cost, we no longer perform vertically aligned attention operations on entire sequence. We focus selectively on vectors with special label positions of the sequence, like [CLS] which comes from RoBERTa, and special labels which we defined additionally in section 3.1.2.

In each layer, align the output of the k rows hidden layers and perform a self-attention operation on the vertically aligned vectors. We use $\mathbf{H}_V \in \mathbb{R}^{d \times k}$ to denote any set of alignment label vectors in the same vertical direction. First, the vector for each label is linearly mapped to three different spaces, the corresponding matrix vectors $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{D \times k}$ are calculated respectively and D denotes the space vector dimension .

$$\mathbf{Q} = \mathbf{W}_Q \mathbf{H}_V \tag{5}$$

$$\mathbf{K} = \mathbf{W}_K \mathbf{H}_V \tag{6}$$

$$\mathbf{V} = \mathbf{W}_V \mathbf{H}_V \tag{7}$$

Here, $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{D \times d}$ represent the trainable parameter matrix respectively and $\mathbf{Q} = [\mathbf{q}_1, ..., \mathbf{q}_k]$, $\mathbf{K} = [\mathbf{k}_1, ..., \mathbf{k}_k]$, $\mathbf{V} = [\mathbf{v}_1, ..., \mathbf{v}_k]$.

Then, denoting k groups of input sequences with $(\mathbf{K}, \mathbf{V}) = [(\mathbf{k}_1, \mathbf{v}_1), ..., (\mathbf{k}_k, \mathbf{v}_k)]$ and calculating attention scores with corresponding vector $\mathbf{q}_j$ to get the output vector $\mathbf{H}^{out} = [\mathbf{h}_1^{out}, ..., \mathbf{h}_k^{out}] \in \mathbb{R}^{D \times k}$. For each vector $\mathbf{h}_j^{out}$, $j \in \{1, ..., k\}$, we calculate.

$$\begin{aligned} \mathbf{h}_j^{\text{out}} &= \text{Attention}\left((\mathbf{K}, \mathbf{V}), \mathbf{q}_j\right) \\ &= \sum_{i=1}^{k} \alpha_{ji} \mathbf{v}_i \\ &= \sum_{i=1}^{k} \text{softmax}(\text{f}(\mathbf{k}_i, \mathbf{q}_j)) \mathbf{v}_i \end{aligned} \tag{8}$$

Where α_{ji} represents the attention distribution weight of the j-th output corresponding to the i-th input and $i \in \{1, ..., k\}$. $f(\cdot)$ takes the method of scaling the dot product.

We also use a multi heads self-attention [25] mechanism to capture different interactive information in multiple projection spaces. The self-attention operation is performed in N projection spaces, we have.

$$\text{MultiHead}(\mathbf{H}_V) = \mathbf{W}_M [\mathbf{H}_1^{out}, ..., \mathbf{H}_N^{out}] \tag{9}$$

Where $\mathbf{H}_n^{out}$ is the output vector $\mathbf{H}^{out}$ in the n-th projection space, which is calculated following formula (5)-(8). The trainable parameters in different projection spaces are not shared and $n \in \{1, ..., N\}$. MultiHead$(\cdot)$ indicates multi heads self-attention calculation operation, $\mathbf{W}_M$ indicates the trainable matrix.

Finally, the results of all vertical layers are aggregated into a mean pooling layer to calculate the fused output for different specified label which is defined as $\mathbf{H}^{POOL}$. Subsequently, different parts of $\mathbf{H}^{POOL}$ will be used for the corresponding downstream semantic parser.

3.2. **Multi-Task Learning.**

3.2.1. *Divide Subtask by SQL Component.* To distinguish the modeling process for two SQL components containing SELECT and WHERE, as shown in Figure 2, we extract corresponding feature vectors from $\mathbf{H}^{POOL}$ based on the custom labels in formula (4). In other words, the semantic features of each column in different SQL components are represented by the vectors corresponding to the custom labels, as shown in the following formula.

$$\mathbf{H}^{\text{POOL}}_{\text{j,WHE\_TYPE}_\text{j}\_1}, \mathbf{H}^{\text{POOL}}_{\text{j,WHE\_TYPE}_\text{j}\_2}, \mathbf{H}^{\text{POOL}}_{\text{j,SEL\_TYPE}_\text{j}} \tag{10}$$

Then, we reorganize the feature vectors of all columns according to SQL components including SELECE and WHERE. They are denoted respectively as $\mathbf{H}_{SEL}^{POOL}$ and $\mathbf{H}_{WHE}^{POOL}$, which are calculated as follows.

$$\mathbf{H}^{\text{POOL}}_{\text{SEL}} = \text{contact}\left(\mathbf{H}^{\text{POOL}}_{\text{1,SEL\_TYPE}_1}, ..., \mathbf{H}^{\text{POOL}}_{\text{c,SEL\_TYPE}_c}\right) \tag{11}$$

$$\mathbf{H}^{\text{POOL}}_{\text{WHE}} = \text{contact}\left(\mathbf{H}^{\text{POOL}}_{\text{1,WHE\_TYPE}_1\_1}, ..., \mathbf{H}^{\text{POOL}}_{\text{c,WHE\_TYPE}_c\_2}\right) \tag{12}$$

Here, contact indicates the splicing operation on vectors, and $\mathbf{H}^{\text{POOL}}_{\text{SEL}} \in \mathbb{R}^{D\times c}$, $\mathbf{H}^{\text{POOL}}_{\text{WHE}} \in \mathbb{R}^{D\times 2c}$.

```
SELECT ($SEL-AGG  $SEL-COL)*
WHERE  $WHE-COL  $WHE-OP  $WHE-VAL
       ($WHE-CONN
        $WHE-COL  $WHE-OP  $WHE-VAL)*
```

FIGURE 3. SQL sketch. $ refers to the slot that needs to be filled. * indicates that the number of corresponding slots.

Following the slot filling method proposed by the SQLNet model, we create a SQL sketch as shown in Figure 3. The advantage of this approach is that it utilizes the structural information of SQL. In order to fill contents of different slots in the sketch, the generation of SQL statement can be decoupled into eight subtasks, which include SEL-NUM, SEL-COL, SEL-AGG, WHE-NUM, WHE-COL, WHE-OP, WHE-VAL and WHE-CONN. Different colors are used in Figure 2 to indicate the use of vectors on different subtasks. We use $\mathbf{W}$ with different subscripts to indicate trainable parameters of each subtask and note that they are not shared in all tasks.

• **SEL-NUM.** Predict the number of selected columns in SELECT clause. We found that this number is generally limited. So, we set an upper limit M for the number of columns in SELECT clause and use the global vector $\mathbf{H}^{\text{POOL}}_{\text{CLS}}$ which is corresponding to

label [CLS] to predict the number of columns. Thus transforming the SEL-NUM task into a M+1 classification problem (0 to M, 0 means that there is no column in SELECT clause). The calculation follows.

$$P_{\text{SEL-NUM}} = \text{softmax}(\mathbf{W}_{\text{SEL-NUM}}\mathbf{H}_{\text{CLS}}^{\text{POOL}}) \tag{13}$$

• **SEL-COL.** Select the columns in SELECT clause from the table. We use the semantic representation vector $\mathbf{H}_{SEL}^{POOL}$ which corresponds to the label $[SEL\_TYPE_j]$ to obtain the probability distribution of all columns occurring in SELECT clause.

$$P_{\text{SEL-COL}} = \text{softmax}(\mathbf{W}_{\text{SEL-COL}}\mathbf{H}_{\text{SEL}}^{\text{POOL}}) \tag{14}$$

• **SEL-AGG.** Predict the aggregate function of the selected column in SELECT clause. Every column in SELECT clause correspond to an aggregate function, and the aggregate function in database include NULL, AVG, MAX, MIN, COUNT, SUM. Note that NULL means no aggregation function, AVG means taking the average value, MAX, MIN respectively indicates the maximum and minimum value. Similar to the SEL-COL task, we also use vector $\mathbf{H}_{SEL}^{POOL}$ for prediction, with the difference that SEL-AGG task is a six-classification problem when predicting the aggregate function for each column.

$$P_{\text{SEL-AGG}} = \text{softmax}(\mathbf{W}_{\text{SEL-AGG}}\mathbf{H}_{\text{SEL}}^{\text{POOL}}) \tag{15}$$

• **WHE-NUM.** Predict the number of selected columns in WHERE clause. Similar to SEL-NUM, the same vector corresponding to label [CLS] is used and an upper limit N is also designed for the number, which can be understood as a N+1 classification problem.

$$P_{\text{WHE-NUM}} = \text{softmax}(\mathbf{W}_{\text{WHE-NUM}}\mathbf{H}_{\text{CLS}}^{\text{POOL}}) \tag{16}$$

• **WHE-COL.** Predict the columns in WHERE clause. Similar to the SEL-COL task, we use semantic representation vector $\mathbf{H}_{WHE}^{POOL}$ which corresponds to the labels $[WHE\_TYPE_j\_1]$ and $[WHE\_TYPE_j\_2]$ to calculate the probability of each column being called once and twice in WHERE clause, respectively.

$$P_{\text{WHE-COL}} = \text{softmax}(\mathbf{W}_{\text{WHE-COL}}\mathbf{H}_{\text{WHE}}^{\text{POOL}}) \tag{17}$$

• **WHE-VAL.** Predict the values corresponding to different conditional columns in WHERE clause. Since in actual scenarios, the keywords of natural language queries often do not exactly match the values in table. To ensure the execution accuracy of SQL statements, we use the value in the database as the standard.

Unlike in English, where the word is the smallest semantic unit. When dealing with Chinese inputs, characters are often used as the minimum encoding units, and most of the individual characters do not have complete semantics, which makes the extraction of conditional values difficult. At the same time, the execution of SQL is rigorous. To ensure that the final generated SQL statements can be executed to get the correct results, we adopt a fuzzy keyword matching method based on span prediction in model training and inference to regulate and correct the keywords in the query.

Specifically, we first slice the entire query by n-gram to obtain candidate words. For the range of n, we determine dynamically by the sequence length of the $Value$ in ground truth for each example and $n \in \{n_{\min}, ..., n_{\max}\}(n_{\min} \geq 1)$.

$$n_{\min} = \text{len}(Value) - i \tag{18}$$

$$n_{\max} = \text{len}(Value) + i \tag{19}$$

Where, $\text{len}(\cdot)$ denotes the calculation of sequence length. i is the hyperparameter, usually taken as $i = 3$.

Then we propose a heuristic boundary matching method to calculate the span of keywords in the query. As shown in Figure 2, using the different pooled vector to predict the start and end positions of keywords from the query to determine the approximate range. The specific calculations are as follows.

$$\mathbf{H}^{\text{START}}_{\text{Q\_COL}} = \text{relu}\left(\mathbf{W}^{\text{START}}_{\text{H}}\mathbf{H}^{\text{POOL}}_{\text{Q}} + \mathbf{W}^{\text{START}}_{\text{WHE}}\mathbf{H}^{\text{POOL}}_{\text{WHE}}\right) \tag{20}$$

$$P^{\text{START}}_{\text{WHE-VAL}} = \text{softmax}\left(\mathbf{W}^{\text{START}}_{\text{Q\_COL}}\mathbf{H}^{\text{START}}_{\text{Q\_COL}}\right) \tag{21}$$

$$\mathbf{H}^{\text{END}}_{\text{Q\_COL}} = \text{relu}\left(\mathbf{W}^{\text{END}}_{\text{H}}\mathbf{H}^{\text{POOL}}_{\text{Q}} + \mathbf{W}^{\text{END}}_{\text{WHE}}\mathbf{H}^{\text{POOL}}_{\text{WHE}}\right) \tag{22}$$

$$P^{\text{END}}_{\text{WHE-VAL}} = \text{softmax}\left(\mathbf{W}^{\text{END}}_{\text{Q\_COL}}\mathbf{H}^{\text{END}}_{\text{Q\_COL}}\right) \tag{23}$$

Where, $\mathbf{H}^{\text{POOL}}_{\text{Q}} \in \mathbb{R}^{D\times L}$ indicates the vector after pooling operation corresponding to the query in formula (4). $P^{\text{START}}_{\text{WHE-VAL}}, P^{\text{END}}_{\text{WHE-VAL}} \in \mathbb{R}^{L\times 2c}$ are predicted indexes of start and end positions in the query respectively.

We divide keyword mismatch problem into two categories according to column type. First is TEXT type, which mainly includes the keywords in query are abbreviations or aliases for the ground truth in table. For this type of case, we directly use the fuzzy matching method. Then is REAL type, which means that numeric units of the keywords in query and table are inconsistent. For example, the "20000 yuan" in query corresponds to "2" (ten thousand yuan as the unit) in table. This inconsistency also includes between Chinese and Arabic numerals. For REAL type keywords, we first unify their format by regular expressions. However, unlike TEXT type, values of REAL type may not be forced to originate from tables. It can be determined by the will of the user and the SQL statement can execute this intent. Therefore, when dealing with values of numeric type, we only convert their units and no longer match similar values from the table.

After obtaining the span of the keyword, we design an evaluation function to match it with ground truth for TEXT type columns, which consists of two matching mechanisms. One is the edit distance we used in row extraction (Section 3.1.1). For the other method, we use the difflib module in python library. This is a method for comparing the similarity between pairs of hash sequences, to avoid recurring nonsensical sequences and thus matching the longest consecutive subsequence. Our evaluation function $E(\cdot)$ is defined as follows.

$$E(x, y) = \sigma S_1(x, y) + (1 - \sigma)S_2(x, y) \tag{24}$$

Where, x, y indicates a span of the keyword and a ground truth value respectively. S_1 and S_2 are two matching methods, both of them return a similarity score from 0 to 100. σ is a hyperparameter and we take it as 0.5.

Finally, we select the value with the highest evaluation function score from the corresponding column in table to generate the SQL statement.

• **WHE-OP.** Predict the operator which determines the relationship between each column and its corresponding value in WHERE clause. All operators include $>, <, =, != =$. Similar to the SEL-AGG task, we use $\mathbf{H}^{\text{POOL}}_{\text{WHE}}$ to design it as a four-classification problem, calculated as follows.

$$P_{\text{WHE-OP}} = \text{softmax}(\mathbf{W}_{\text{WHE-OP}}\mathbf{H}^{\text{POOL}}_{\text{WHE}}) \tag{25}$$

• **WHE-CONN.** Predict the connection relationship between the columns in WHERE clause, which includes NULL, AND, OR. Note that NULL means no connection relationship. We use vector $\mathbf{H}^{\text{POOL}}_{\text{CLS}}$ to transform it into a triple classification problem.

$$P_{\text{WHE-CONN}} = \text{softmax}(\mathbf{W}_{\text{WHE-CONN}}\mathbf{H}^{\text{POOL}}_{\text{CLS}}) \tag{26}$$

3.2.2. *Loss Function.* We optimize the loss function for each subtask. SEL-NUM, SEL-AGG, WHE-NUM, WHE-OP, WHE-VAL and WHE-CONN use the cross-entropy loss function. To compare different columns in SELECT and WHERE clauses on the global space, we use Kullback-Leibler [33] divergence as the loss function to learn the probability distribution of each column being called in SQL clause. We also find that the subtasks are not independent of each other. For example, SEL-NUM and SEL-COL respectively predicts the number and name of the columns in the SELECT clause, and there is a correlation between them. Inspired by multi-task learning [34], we adopt a regularized learning strategy to accumulate the loss function of all subtasks to form the overall objective function. so as to play a role similar to the mean constraint on the parameters of each subtask in training and optimizing the objective function.

4. **Experiments.**

4.1. **Dataset.** We use the TableQA dataset, which is the only public Chinese single-table query dataset we know. As a large-scale cross-domain Q&A dataset based on tables, TableQA contains 64891 natural language queries, 20311 unique SQL statements and more than 6000 tables. It is similar in data scale to the English dataset WikiSQL, but further considers entity links and the answerability of queries, which is closer to the real scene. So it is more challenging. In order to further test the generalization ability of the model, the tables in dev and test sets are not visible to the train set.

4.2. **Evaluation Metric.** We follow the evaluation metrics used on WikiSQL, which includes logical form accuracy ($\mathrm{ACC_{lf}}$) and execution accuracy ($\mathrm{ACC_{ex}}$). In addition, we further calculate the average of them ($\mathrm{ACC_{mean}}$). Specifically, $\mathrm{ACC_{lf}}$ denotes the proportion of the number of generated SQL statement that can accurately match the ground truth to the total number of samples. $\mathrm{ACC_{ex}}$ denotes the proportion of the total number of samples that can obtain consistent results with ground truth after executing the SQL statement. $\mathrm{ACC_{mean}}$ is a comprehensive effect on two evaluation indicators.

4.3. **Implementation Details.** The experiment uses the model given by TableQA as the baseline, which is improved on SQLNet. We developed TableSQL on pytorch [35], and initialize its encoder from a Chinese RoBERTa model , which uses the Whole Word Mask method to mask the whole words in the Chinese sentence instead of a single character in pretraining process. We train TableSQL on a NVIDIA TITAN RTX GPU with 24GB memory. The encoder based on Transformer has 12 layers, and each layer has 12 self-attention heads, with a hidden layer size of 768, and the hyperparameters in vertical attention mechanism are set in line with encoder. Maximum input length of the encoder is 512 and the number of candidate rows $k = 3$. We adopt BertAdam as the optimizer, which learn rate is 12e-6, and warm up is 2e-2. We use the same training, validation, and testing datasets as the TableQA release. When training, we randomly permute the example order and the column order in a table. The batch size is set to 16, and training 60 epochs. We check the overall accuracy once on each epoch and stop the training if there is no improvement in 8 checks.

4.4. **Results and Discussion.** We compare five state-of-the-art methods on this dataset. SQLNet is our baseline model which proposes a sequence to set architecture based on SQL syntax. It is based on a bidirectional LSTM of the glove initialization word embedding to design the encoder. SQLova introduces BERT in the encoding, which encodes natural language queries and column names together. Then, they are fed into a separate LSTM structure with an attention mechanism that combines column names. Coarse2Fine first generates a rough sketch according to input query. Then, it further fills in missing details.

X-SQL has redesigned the method to capture global information in the process of encoding based on SQLova, and uses execution guidance to enhanced decoding effects.

TABLE 1. Comparison of TableSQL with other methods on dev and test sets.

Methods	Dev			Test		
	ACC_{lf}	ACC_{ex}	ACC_{mean}	ACC_{lf}	ACC_{ex}	ACC_{mean}
SQLNet	61.3	66.2	63.7	61.4	67.2	64.3
SQLova	81.4	85.3	83.3	81.7	85.8	83.7
Coarse2Fine	73.0	76.9	74.9	72.6	76.7	74.7
MQAN	75.7	79.2	77.4	74.8	78.8	76.8
X-SQL	82.9	87.0	84.9	83.3	87.6	85.4
TableSQL	**87.7**	**90.2**	**89.0**	**89.7**	**92.0**	**90.9**

Table 1 shows a comparison of TableSQL with other methods on dev and test sets. Note that the performance of the methods used for comparison are modified and reproduced from their original papers, as these methods originally used a different dataset than ours. The experimental results show that TableSQL consistently outperforms the compared methods in terms of logical form accuracy, execution accuracy and their comprehensive effects. Compared to the last state-of-the-art method X-SQL, we achieve absolute improvements of 6.4%(ACC_{lf}), 4.4%(ACC_{ex}) and 5.5%(ACC_{mean}) on test set.

To further investigate the advantages of TableSQL, we also provide a breakdown of the logical form accuracy by component on dev set in Table 2. Note that the results used for comparison in the experiment are published along with TableQA. The experimental results show that compared to the best results available, TableSQL achieves significant advantages on two subtasks WHERE-COL (98.0 vs 79.4) and WHERE-VAL (94.4 vs 49.5). We also achieve an average absolute advantage of 9.6% over the latest improved version of SQLova (End2End) on 8 SQL components. Notably, in WHERE-COL and WHERE-VAL task, we get a significant improvement of 18.8% and 44.9% compared to SQLova (End2End), which is the key to the overall performance improvement of our model. We attribute this to TableSQL better captures the mapping relationship between natural language queries and database tables.

4.5. **Ablation Study.** Since the process of learning mapping relationships between queries and tables is implicit in our method (Section 3.1.2). To further visualize the ability of TableSQL to establish mappings between heterogeneous data. Inspired by BertViz [36],

TABLE 2. Performance of each method on different SQL components (dev set).

SQL Causes	Components	SQLNet	SQLova	SQLova (offline)	SQLova (End2End)	TableSQL
SELECT	NUM	98.6	99.3	99.3	99.2	**99.6**
	COL	91.5	96.1	96.1	96.2	**97.4**
	AGG	93.7	98.1	98.1	**98.2**	**98.2**
WHERE	NUM	90.1	95.7	95.7	95.1	**98.1**
	COL	71.2	79.4	79.4	79.2	**98.0**
	OP	86.5	93.3	93.3	93.1	**98.9**
	VAL	43.2	44.1	47.4	49.5	**94.4**
	CONN	91.2	95.9	95.9	95.2	**98.1**

we qualitatively evaluate the distribution of attention at key locations during horizontal cross-structural joint encoding. Specifically, we visualize the effect of multi heads self-attention for some layers in the Transformer structure initialized with RoBERTa.

We use a Transformer structure with 12 attention layers where each layer contains 12 headers and use the variable l to denote the different attention layers. To show the role of custom labels in our method when mapping overlapping relationships, we plot the distribution of attention weights for custom labels to the entire input sequence on different layers. We use a vanilla version of our model which takes only column names when serializing encoding (Section 3.1.2) to eliminate the effect of types and values. To facilitate graph readability, we choose some samples with small query length and table size.

As shown in Figure 4, when the query is "可以告诉我锦州港和华夏航空的股票代码分别是多少吗? (Can you tell me the stock symbols of Jinzhou Port and China Airlines?)", its corresponding table contains a total of seven columns. It can be seen that both "锦州港 (Jinzhou Port)" and "华夏航空 (China Airlines)" correspond to column "名称 (Name)" in the table, that means column "名称 (Name)" is called twice in the query. In our method, we found that two custom labels [whe_text_1] and [whe_text_2] of column "名称 (Name)" are strongly related to the keywords "锦州港 (Jinzhou Port)" and "华夏航空 (China Airlines)" in the query respectively. Specifically, in Figure 4.A, [whe_text_1] for column "名称 (Name)" has significantly higher attention weights on "锦州港 (Jinzhou Port)" than other tokens. In Figure 4.B, [whe_text_2] for column "名称 (Name)" has significantly higher attention weights on "华夏航空 (China Airlines)" than other tokens.

We further find that our method has some robustness when we enter an example that tends to confuse the model. As shown in Figure 5, when the query is "请问一下快递公司 2018 年单量大于 10 亿件并且 2017 年大于 10 亿件的市占率最大是多少? (What is the maximum market share of courier companies with a unit volume greater than 1 billion in 2018 and greater than 1 billion in 2017?)". Two column names "2018" and "2017" mentioned in the query are not easily distinguishable. Other than that, the operation of both columns is "大于 10 亿件 (greater than 1 billion pieces)". But we find that our model can still distinguish them in some attention layers. Specifically, in Figure 5.A, when $l = 9$, [whe_real_1] for column "2018" has a higher attention weight on "大于 10 亿 (greater than 1 billion)" than other tokens. In Figure 5.B, when $l = 11$, [whe_real_1] for column "2017" has significantly higher attention weights on "大于 10 亿 (greater than 1 billion)" which close to the fragment "2017" than other tokens. In Figure 5.C, when $l = 9$, [sel_real] for column "市占率 (market share)" has significantly higher attention weights on "最大 (maximum)" than other tokens, which is the operation function for column "市占率 (market share)" in SELECT clause.

To further evaluate the effect of other techniques on model performance, we conduct ablation studies by removing the technique to be evaluated from the model. As shown in the Table 3, we still use $\mathrm{ACC}_{\mathrm{lf}}$, $\mathrm{ACC}_{\mathrm{ex}}$ and $\mathrm{ACC}_{\mathrm{mean}}$ as evaluation metrics to analyze the gains generated by each method. We also try to initialize encoder with different pretrained model such as Chinese BERT to explore the impact on TableSQL. Specifically, we first study the number of candidate rows k in the query-sensitive rows extraction process. We find that the selection of k has a significant impact on the model performance, and the model performance drops sharply when k is taken as 4. We speculate that this may be due to the incorporation of too much noise. When $k = 3$, the mean accuracy of the model reaches the best result of 87.8% (Dev) and 90.0% (Test). We also find a slight performance degradation on the mean accuracy 1.2% (Dev) and 0.9% (Test) in TableSQL under this condition compared to the RoBERTa-based model in Table 2. Nevertheless, the performance of the TableSQL still outperforms X-SQL 2.9% (Dev) and 4.6% (Test),

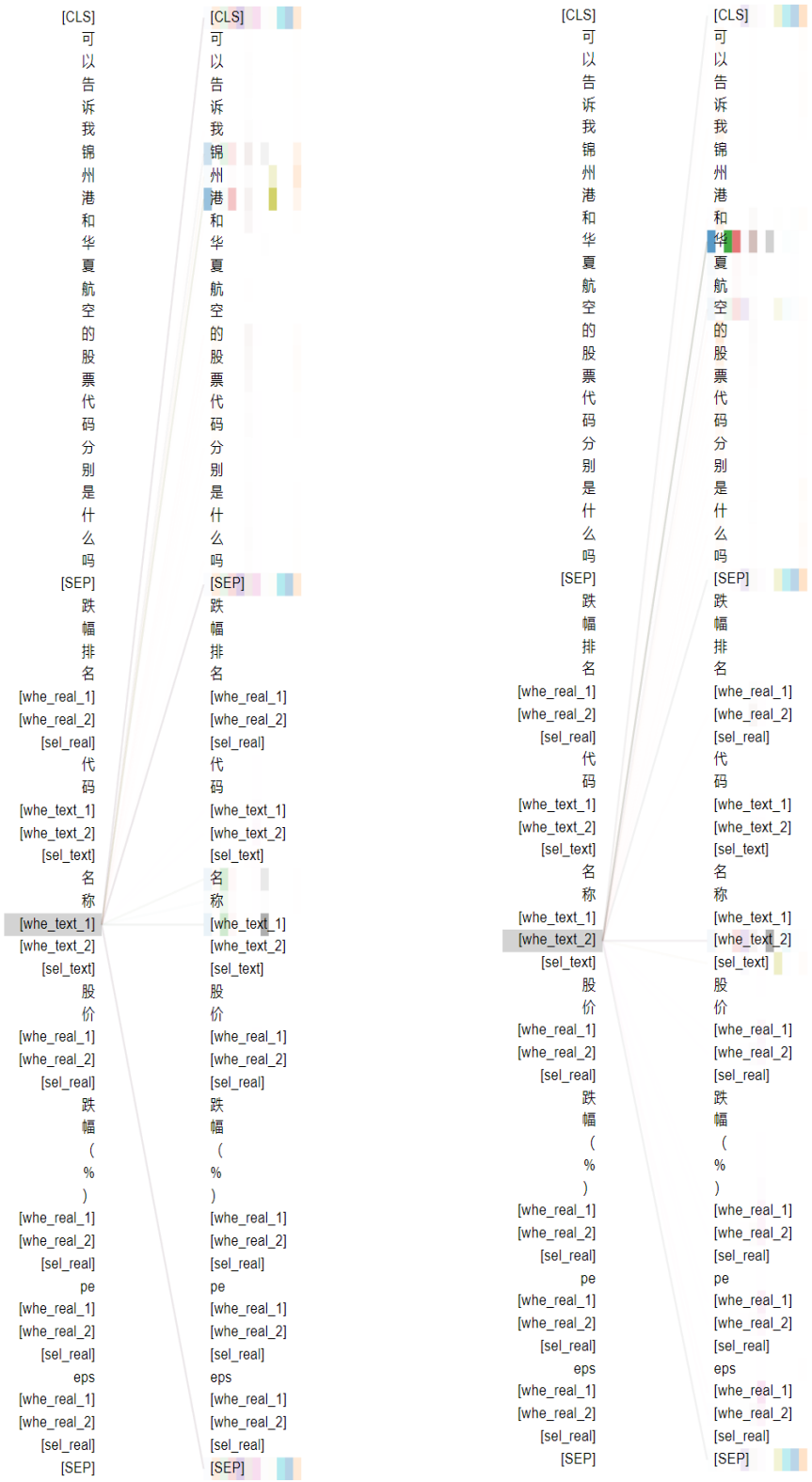

FIGURE 4. Visualization of custom labels on Transformer attention layer, where $l = 10$.

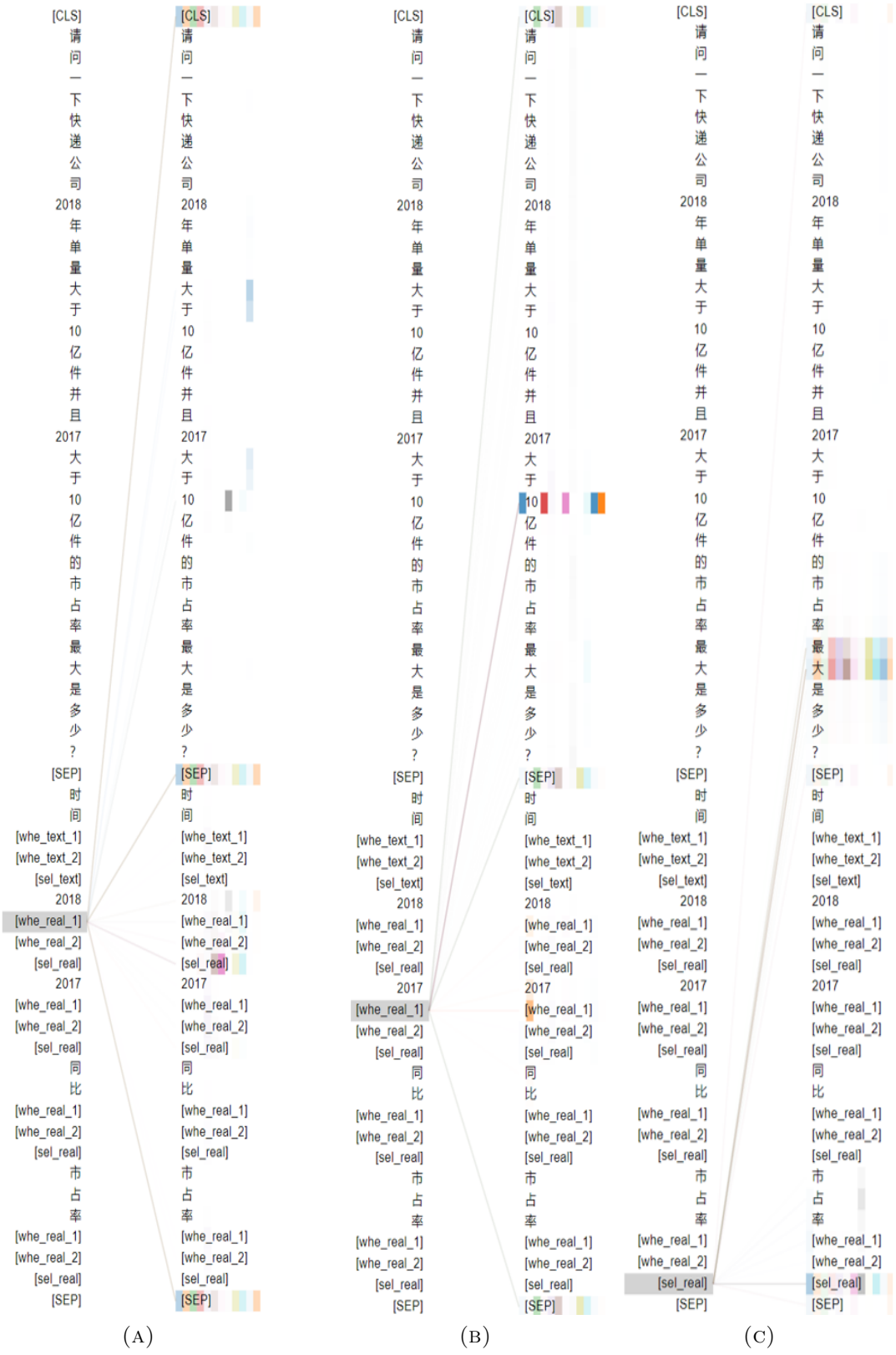

FIGURE 5. Visualization of custom labels on different Transformer attention layers.

TABLE 3. Results of the ablation study. "-" in the setting indicates the removal of a method.

TableSQL(with different k)	**Dev**			**Test**		
	ACC_{lf}	ACC_{ex}	ACC_{mean}	ACC_{lf}	ACC_{ex}	ACC_{mean}
$k = 2$	86.1	89.1	87.6	88.2	90.5	89.4
$k = 3$	86.3	89.3	87.8	88.8	91.1	90.0
$k = 4$	85.4	88.2	86.8	87.0	89.5	88.3
Setting($k = 3$)						
-Rows Extraction	85.6	88.6	87.1	87.8	90.2	89.0
-Vertical Alignment Attention	85.9	88.8	87.4	87.0	89.7	88.4
-Serialization Encoding	85.5	88.9	87.2	86.8	89.7	88.3
-Fuzzy Matching	80.8	84.2	82.5	83.6	86.3	85.0

which illustrates the effectiveness of our approach from another perspective. Then, based on the model with the number of candidate rows $k = 3$, we observe and analyze the gains generated by other techniques.

First, for query-sensitive rows extraction strategy, we use random sampling to replace it. We find it brings an absolute improvement of 0.7% (Dev) and 1.0% (Test) to the mean accuracy. Second, we study the effect of the vertical alignment attention mechanism, which is same as the situation that the number of candidate row k is 1. We find that the mean accuracy is improved by 0.4% (Dev) and 1.6% (Test), which indicates that the method of using attention mechanism to fuse information in vertical direction is effective. Third, when evaluating the gain from serialized encoding, we use a similar approach as in previous visualization studies, considering only the column names in encoding process. We find that serialization encoding of table content and column type can further improve the mean accuracy of 0.6% (Dev) and 1.7% (Test). At last, in the extraction of value in WHERE clause, we analyze the effect of the fuzzy matching method. Compared to just getting the keyword span from the query, further fuzzy matching with database tables could bring an absolute improvement of 5.3% (Dev) and 5.0% (Test) to the mean accuracy of the model. The impact on execution accuracy is significant, because some SQL statements generated by relying only on key information in the query are difficult to execute to get the correct results. The accuracy of logical form also decreases significantly in this process, which we speculate may be related to the multi-task learning during training.

4.6. **Error Analysis.** Compared with the ground truth in TableQA dev set, the reasons for the wrong prediction results are mainly concentrated in the following categories. (1) REAL type keyword error. The numerical data in ground truth comes from the table, but the values we predict come from user queries, and some of them do not appear in the table. However, SQL language have the ability to process numerical data during execution and could still execute user's intention. Although this error brings about the loss of the accuracy of the logical form, it has less impact on execution accuracy. (2) Errors, missing and redundant columns in the WHERE clause. Such errors mostly occur in specialized fields such as materials and finance, which contain a large number of specialized terms. Although pretrained models can help TableSQL working on cross-domain data, there are limitations for some specific domains. In the future we will consider reducing this error by additional pretraining. (3) The order of the condition in the SELECT and WHERE clauses are different with ground truth. This kind of error is acceptable, which will not affect the execution accuracy of the SQL statement, but only reduces the accuracy of the logical form.

5. **Conclusion.** In this paper, we propose a new generalizable semantic parsing model TableSQL for database tables. Firstly, it encodes two heterogeneous data, which includes automatically sensing the objects mentioned in the table and integrating information about the same object with different attributes and different objects with same attributes. This method can take full advantage of the structured knowledge and helps to better capture the mapping relationship between two heterogeneous data. In addition, we design a new label-based encoding method which implements multiple relational mappings on the same column of a table, compensating for the shortcomings of traditional methods. In decoding, we reorganize the SQL generation process based on the semantic knowledge and SQL syntax features obtained from the encoder, and design a new keyword matching method, which helps the model to learn the matching relationship between fuzzy words with the same meaning but different forms. Overall, TableSQL achieves competitive performance, with a 5.5% absolute gain of mean accuracy to the last state-of-the-art method. Through ablation experiments, we observe that various parts of our method are effective. In the future, we hope to extend our model to SQL generation tasks that include more complex components such as multi-table linking and nesting.

Acknowledgment. This work is partially supported by the National Natural Science Foundation of China under Grant 61673387.

Data Availability Statement. The pretrained model, python toolkit and dataset used in this paper are from https://github.com/ymcui/Chinese-BERT-wwm, https://docs.python.org/3/library/difflib.html and https://github.com/ZhuiyiTechnology/TableQA.

REFERENCES

[1] C. Baik, H. V. Jagadish and Y. Li, Bridging the semantic gap with sql query logs in natural language interfaces to databases, *35th IEEE International Conference on Data Engineering*, pp. 374-385, 2019.

[2] Q. Min, Y. Shi and Y. Zhang, A pilot study for chinese sql semantic parsing, *2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3650-3656, 2019.

[3] V. Zhong, C. Xiong and R. Socher, Seq2sql: Generating structured queries from natural language using reinforcement learning, *arXiv preprint*, arXiv:1709.00103, 2017.

[4] X. Xu, C. Liu and D. Song, Sqlnet: Generating structured queries from natural language without reinforcement learning, *arXiv preprint*, arXiv:1711.04436, 2017.

[5] I. Sutskever, O. Vinyals, and Q. V. Le, Sequence to sequence learning with neural networks, *Advances in neural information processing systems*, pp. 3104-3112, 2014.

[6] W. Hwang, J. Yim, S. Park and M. Seo, A comprehensive exploration on wikisql with table-aware word contextualization, *arXiv preprint*, arXiv:1902.01069, 2019.

[7] J. Devlin, M. W. Chang, K. Lee and K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, *2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, vol. 1, pp. 4171–4186, 2019.

[8] J. Ma, Z. Yan, S. Pang, Y. Zhang and J. Shen, Mention extraction and linking for sql query generation, *2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6936–6942, 2020.

[9] N. Sun, X. Yang and Y. Liu, Tableqa: a large-scale chinese text-to-sql dataset for table-aware sql generation, *arXiv preprint*, arXiv:2006.06434, 2020.

[10] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer and V. Stoyanov, Roberta: A robustly optimized bert pretraining approach, *arXiv preprint*, arXiv:1907.11692, 2019.

[11] M. Rabinovich, M. Stern and D. Klein, Abstract syntax networks for code generation and semantic parsing, *55th Annual Meeting of the Association for Computational Linguistics*, vol. 1, pp. 1139–1149, 2017.

[12] D. Keysers, N. Schärli, N. Scales, H. Buisman, D. Furrer, S. Kashubin, N. Momchev, D. Sinopalnikov, L. Stafiniak, T. Tihon, D. Tsarkov, X. Wang, M. Zee and O. Bousquet, Measuring compositional generalization: A comprehensive method on realistic data, *8th International Conference on Learning Representations (ICLR)*, https://openreview.net/forum?id=SygcCnNKwr, 2020.

[13] X. V. Lin, R. Socher and C. Xiong, Bridging textual and tabular data for cross-domain text-to-sql semantic parsing, *2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pp. 4870–4888, 2020.

[14] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang and D. Radev, Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task, *2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3911–3921, 2018.

[15] T. Yu, R. Zhang, M. Yasunaga, Y. C. Tan, X. Lin, S. Li, H. Er, I. Li, B. Pang, T. Chen, E. Ji, S. Dixit, D. Proctor, S. Shim, J. Kraft, V. Zhang, C. Xiong, R. Socher and D. Radev, Sparc: Cross-domain semantic parsing in context, *57th Annual Meeting of the Association for Computational Linguistics*, pp. 4511–4523, 2019.

[16] L. Dong, M. Lapata, Language to logical form with neural attention, *54th Annual Meeting of the Association for Computational Linguistics*, vol. 1, pp. 33–43, 2016.

[17] C. Wang, M. Brockschmidt and R. Singh, Pointing out SQL queries from text, https://www.microsoft.com/en-us/research/publication/pointing-sql-queries-text, 2017.

[18] T. Yu, Z. Li, Z. Zhang, R. Zhang and D. Radev, Typesql: Knowledge-based type-aware neural text-to-sql generation, *2018 Conference of the North American Chapter of the Association for Computational Linguistics*, vol. 2, pp. 588–594, 2018.

[19] T. Yu, M. Yasunaga, K. Yang, R. Zhang, D. Wang, Z. Li and D. Radev, Syntaxsqlnet: Syntax tree networks for complex and cross-domaintext-to-sql task, *2018 Conference on Empirical Methods in Natural Language Processing*, pp. 1653–1663, 2018.

[20] J. Guo, Z. Zhan, Y. Gao, Y. Xiao, J. G. Lou, T. Liu and D. Zhang, Towards complex text-to-sql in cross-domain database with intermediate representation, *57th Conference of the Association for Computational Linguistics (ACL)*, vol. 1, pp. 4524–4535, 2019.

[21] B. Bogin, M. Gardner and J. Berant, Global reasoning over database structures for text-to-sql parsing, *2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3657–3662, 2019.

[22] B. Wang, R. Shin, X. Liu, O. Polozov and M. Richardson, Rat-sql: Relation-aware schema encoding and linking for text-to-sql parser, *58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 7567–7578, 2020.

[23] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee and L. Zettlemoyer, Deep contextualized word representations, *2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, vol. 1, pp. 2227–2237, 2018.

[24] A. Radford, K. Narasimhan, T. Salimans and I. Sutskever, Improving language understanding by generative pre-training,https://www.cs.ubc.ca/amuham01/LING530/papers/radford2018improving.pdf, 2018.

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, Attention is all you need, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*, pp. 5998–6008, 2017.

[26] P. He, Y. Mao, K. Chakrabarti and W. Chen, X-SQL: reinforce schema representation with context, *arXiv preprint*, arXiv:1908.08113, 2019.

[27] L. Wang, A. Zhang, K. Wu, K. Sun and H. Wang, Dusql: A large-scale and pragmatic chinese text-to-sql dataset, *2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6923–6935, 2020.

[28] L. Dong, M. Lapata, Coarse-to-fine decoding for neural semantic parsing, *56th Annual Meeting of the Association for Computational Linguistics (ACL)*, vol. 1, pp. 731–742, 2018.

[29] G. Navarro, A guided tour to approximate string matching, *ACM computing surveys*, vol. 33, no. 1, pp. 31-38, 2001.

[30] S. Yavuz, I. Gur, Y. Su and X. Yan, What it takes to achieve 100 percent condition accuracy on wikisql, *2018 Conference on Empirical Methods in Natural Language Processing*, pp. 1702–1711, 2018.

[31] P. Yin, G. Neubig, W. Yih and S. Riedel, TaBERT: Pretraining for joint understanding of textual and tabular data, *58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 8413–8426, 2020.
[32] T. Shi, K. Tatwawadi, K. Chakrabarti, Y. Mao, O. Polozov and W. Chen, Incsql: Training incremental text-to-sql parsers with non-deterministic oracles, *arXiv preprint*, arXiv:1809.05054, 2018.
[33] P. Bojanowski, E. Grave and A. Joulin, Enriching word vectors with subword information, *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135-146, 2017.
[34] X. Liu, P. He, W. Chen and J. Gao, Multi-task deep neural networks for natural language understanding, *57th Conference of the Association for Computational Linguistics (ACL)*, vol. 1, pp. 4487–4496, 2019.
[35] A. Paszke, S. Gross and S. Chintala, Automatic differentiation in pytorch, https://compcalc.github.io/public/pytorch/ad_pytorch.pdf, 2017.
[36] J. Vig, A multiscale visualization of attention in the transformer model, *57th Conference of the Association for Computational Linguistics (ACL)*, vol. 3, pp. 37–42, 2019.