A Text-to-SQL Query Parser with Integrating Instances

Xuxu Cui, *Member, IAENG,* Derong Shen

***Abstract*—Text-to-SQL is a natural language processing task focused on translating natural language queries into Structured Query Language (SQL), aiming to facilitate interaction between non-technical users and relational databases. Currently, many existing Text-to-SQL models based on deep learning methods concentrate on encoding natural language questions and database schemas, while often neglecting the inclusion of database instance information. Additionally, these models are constrained by SQL syntax during the decoding process, which limits their generalization ability. This paper presents a Text-to-SQL query parser that integrates database instances to address these limitations. By incorporating database instance information, both natural language queries and database schema-instance pairs are represented as graph structures for encoding. Predefined relationships are utilized to unify the encoding of natural language questions, database schemas, and database instances. An instance-aware query parsing method is then applied during the decoding phase, allowing the model to fully leverage database instance information when generating SQL queries, which improves its scalability. The method is evaluated using the cross-domain Spider dataset, and pre-trained language models such as GloVe, GAP, and BERT are employed to enhance model performance. Experimental results show that the proposed approach significantly improves prediction accuracy in Text-to-SQL tasks.**

***Index Terms*—text-to-sql, natural language processing, deep learning, semantic parsing, database query.**

I. Introduction

In recent years, the development of deep learning techniques and the availability of large-scale datasets [1] have led to the increasing application of deep learning in the field of natural language processing (NLP), driving significant technological advancements. Specifically, deep learning has produced notable results in tasks such as named entity recognition in Chinese electronic medical records [2] and dimensionality reduction for Internet of Things (IoT) intrusion detection systems [3]. These developments have not only advanced NLP technologies but have also reinvigorated academic interest in the Text-to-SQL task.

The Text-to-SQL task aims to directly convert natural language queries into database query languages, which is essential for understanding the implicit semantics of natural language. Traditionally, rule-based Text-to-SQL systems have limitations in capturing the deeper meanings of natural language and face challenges related to model scalability. In contrast, recent Text-to-SQL systems based on deep learning methods, as discussed in [4], have demonstrated the ability to extract more complex semantic information from natural language, making them a current focus of research.

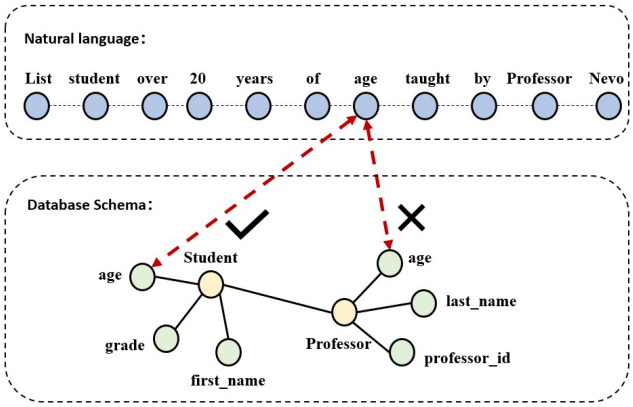

Fig. 1. Coding and linking issues with existing methods.

Today, numerous deep learning-based Text-to-SQL models have been proposed. Most existing models treat natural language as a sequence and represent the database schema as a schema graph during the encoding phase. These models establish relationships between the natural language query and the database schema through schema links. Notable systems employing this encoding approach include RAT-SQL [5], GNNSQL [6], LEGSQL [7], Proton [8], GASQL [9], and SADGA [10].

Fig. 1 illustrates the encoding and schema linking process in an existing model. Given a natural language query, such as "List students over 20 years of age taught by Professor Nevo", and the corresponding database schema, the query is treated as a sequence, while the database schema is represented as a directed graph. In this graph, yellow nodes indicate table names, and green nodes represent column names. A string matching method or an attention mechanism is used to establish links between the query and the schema. However, this method may cause linking errors. For example, the term "age" in the query is expected to link to the "age" column in the "student" table, but may also link to the "age" column in the "Professor" table due to the reliance on string matching, which creates links to both. Additionally, important information, such as the number "20" in the query, is not mapped to the database schema. These models primarily focus on encoding natural language queries and database schemas while overlooking database instance information, resulting in incomplete utilization of available data.

Inspired by the aforementioned model, this paper incorporates database instance information during construction and encodes both natural language queries and database schema-instance pairs as graph structures. During the schema linking

Manuscript received September 15, 2024; revised March 17, 2025.

Xuxu Cui is a teaching assistant of School of Computer Science and Software Engineering, University of Science and Technology Liaoning, Liaoning 114051, China. (e-mail:515216585@qq.com).

Derong Shen is a professor of School of Computer Science and Engineering, Northeastern University, Liaoning 110819, China. (corresponding author to provide phone: +8618698762081; e-mail:shendr@mail.neu.edu.cn).

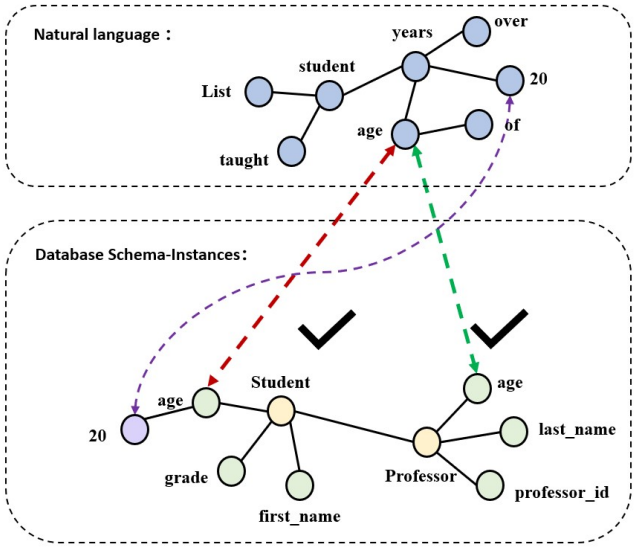

Fig. 2. Coding and linking methods for this article.

process, a combination of string matching and attention mechanisms is used to identify both global and local links between nodes in the natural language query graph and those in the database schema-instance graph. This approach helps identify strong and weak links between natural language queries and schema instances, effectively reducing structural discrepancies during the linking process.

As shown in Fig. 2, a purple node is added to the database schema-instance graph to represent the content of the database instance. The linking method proposed in this paper captures not only the weak link between the term “age” in the natural language query and the “age” column in the “Professor” table, but also the mapping between the value “20” in the query and the corresponding database schema-instance.

Optimization of the existing model at the decoding stage presents two key challenges. First, when using the Relational Attention Transformer (RAT) for unified encoding, both the natural language query and the database schema are encoded, but the integration of database instance values is limited. To address this, this paper proposes a multi-feature fusion method based on RAT, which incorporates relational information from different features. This method enables the unified encoding of natural language, database schema, and database instances. By leveraging the relationships between these elements, the model’s expressive capability is enhanced, leading to the generation of more accurate SQL queries.

Second, during the decoding process of generating SQL query statements from natural language questions, current systems focus on generating database tables and columns while omitting database instance values. At this stage, existing models do not use intermediate representations; they generate SQL queries directly from the natural language input. Systems such as IRNet [11] and RAT-SQL [5] follow this approach. As a result, these models cannot generate queries that are independent of a specific database query language and are subject to the constraints of SQL syntax, which limits their generalization and scalability. To address this issue, this paper employs abstract syntax trees (ASTs) as intermediate representations for generating SQL queries rather than directly translating natural language into SQL. This method decouples the generated queries from specific database query languages and reduces the limitations imposed by SQL syntax. In addition, because ASTs possess a degree of universality and scalability, they can be converted and applied across different database query languages, thereby enhancing the model’s generalization and scalability.

The main contributions of this paper are as follows:

- A Text-to-SQL graph mapping model is proposed, which integrates database instance information. By encoding natural language questions and database schema instances as a graph structure, the model fully leverages instance data, improving both accuracy and generalization.
- A multi-feature fusion method based on the Relational Attention Transformer (RAT) is introduced, which unifies the encoding of natural language questions, database schemas, and instances, treating them as a whole.
- An instance-aware query parsing method is presented, using an abstract syntax tree as an intermediate representation for generating SQL query statements. This approach mitigates the limitations of directly generating SQL from natural language, enhancing both the generalization and scalability of the model.

II. RELATED WORK

Recent developments in deep learning have led to the creation of many cross-domain Text-to-SQL systems designed to address the challenges of user access to databases [12]. Although notable advancements have been made in optimizing both the encoding and decoding stages, current Text-to-SQL tasks continue to face challenges due to the inherent complexity and diversity of natural language [13].

RAT-SQL [5] introduces a unified encoding mechanism to improve the joint representation of questions and patterns. LGESQL [7] employs line graphs to update edge features in heterogeneous graphs for Text-to-SQL, focusing on both local and non-local, dynamic, and static edge features. SADGA [10] utilizes a unified dual-graph framework for queries and database schemas, incorporating a structure-aware graph aggregation mechanism to effectively capture global and local structural information in the query-schema links.

IRNet [11] utilizes a string-matching strategy to separately encode queries and patterns using LSTM, then decodes abstract intermediate representations (IR). BRIDGE [14] serializes queries and patterns into tagged sequences, leveraging BERT [15] and database content to capture query-pattern links. SmBoP [16] introduces the first semi-autoregressive bottom-up semantic parser for the Text-to-SQL decoding stage.

Graph encoders have been widely used in the cross-domain Text-to-SQL field. Literature [6] first introduced graph neural networks (GNNs) to encode database schemas, while Global-GNN [17] applied GNNs to soft-select subsets of tables and columns for query generation. ShadowGNN [18] proposed a graph projection neural network (GPNN) to abstract the representation of queries and database schemas using a simple attention mechanism.

Most of these models treat natural language queries as sequences and database schemas as graph structures. To

capture more semantic information from natural language queries and reduce the gap between query sequences and schema graphs, this paper constructs natural language queries as graph structures. Additionally, existing models typically focus only on schema information, such as tables and columns, without fully utilizing the instance data within the database. To extract more relevant information, this paper represents database schema instances as graphs.

Based on this, the paper proposes a Text-to-SQL graph mapping method that integrates both schema and instance data, allowing for richer semantic information extraction. To further enhance encoding, a unified method based on the Relational Attention Transformer (RAT) is introduced, encoding natural language queries, database schemas, and instance data together. During decoding, abstract syntax trees are used as intermediate representations to overcome SQL syntax limitations and generate SQL queries.

III. METHOD

A. Method overview

This paper presents a Text-to-SQL query parser that incorporates database instances, employing a classic encoder-decoder architecture. The framework consists of three main components: a Text-to-SQL graph mapping module, a multi-feature fusion module based on the Relational Attention Transformer (RAT), and a query parsing module, as shown in Fig. 3.

The Text-to-SQL graph mapping module consists of three submodules: graph construction, graph encoding, and graph mapping. First, natural language queries and database schema instances are converted into graph structures. Then, a Gated Graph Neural Network (GGNN) [19] is used to encode both graphs. Finally, the mapping relationships between the nodes in the natural language graph and the database table names, column names, and instance values are established, resulting in node representations for both graphs.

The multi-feature fusion module based on RAT integrates the encoding of natural language queries, database schema, and database instances. Finally, the instance-aware query parsing module generates more accurate SQL queries by emphasizing the decoding of database instance values during output, ensuring that the generated SQL queries include the relevant instance information.

B. Instance-integrated text-to-sql graph mapping model

1) ***Graph Construction****:* First, a question graph is constructed for natural language queries based on predefined relationships between query terms. Next, a schema graph is built to represent the relationships between tables and columns in the database schema. Since database instance values are discrete, the most relevant instance values to the natural language query are selected and extracted to form a database instance graph. Finally, these instance values are integrated into the schema graph to establish a schema-instance relationship graph.

The natural language query is represented as a graph $G_Q(Q, R_q)$,where the node set Q represents the words in the query, and the set R_q denotes the dependency relationships between the words, with t representing the dependency between q_i and q_j.

As shown in Fig. 4, in this natural language query, the word "taught" has a first-order word distance relationship with "age" and "by", meaning these words are adjacent. Similarly, "taught" has a second-order word distance relationship with "of" and "Professor", indicating that there is one word between them. The relationship between "taught" and "student" is identified using the StanfordCoreNLP [20], specifically referring to students who have been taught, functioning as a modifier in a relative clause.

The database schema graph is constructed by representing table and column names as nodes, with edges defined by their relationships in the schema. The schema is formalized as $G_s = (S, R_s)$, where the node set S consists of table and column names. Columns are denoted as $C = \{c_1, \ldots, c_{|c|}\}$ and tables as $T = \{t_1, \ldots, t_{|t|}\}$, where each column c_i comprises words $c_{i,1}, \ldots, c_{i,|c_i|}$, and each table t_i comprises words $t_{i,1}, \ldots, t_{i,|t_i|}$. The set R_s defines the relationships between table and column names, with a total of six predefined relationships existing between them [10].

Fig. 5 illustrates the process of constructing a database schema diagram. In this example, we focus on the column "professor_id". In the "Student" table, the column "professor_id" shares a same-table matching relationship with the columns "name" and "age". Specifically, the "Student" table and the "professor_id" column are in a table-column matching relationship, meaning that the "professor_id" column belongs to the "Student" table. Additionally, since both the "Student" and "Professor" tables contain the "professor_id" column, they form a primary-foreign key relationship.

In the database, instance values are fixed and discrete, lacking inherent relationships with each other. To address this, an embedding neural network is employed to identify the instance values most relevant to the natural language query. First, the data in the database is processed and encoded using one-hot vectors. During training, labels are used to obtain the specific encoding of the database instance values. The one-hot vector representation is then mapped to a continuous vector representation through an embedding layer, as shown in formula (1).

$$\text{Vector}_v = \text{embedding}\,(X_t) \tag{1}$$

Among them, X_t represents the one-hot vector representation of the instance value at the current time t, and $Vector_v$ represents the vector representation of the current instance value. The final instance values are selected and embedded into the database schema through this mapping relationship.

2) ***Graph coding****:* After constructing the question relationship graph and the database model-instance relationship graph, a neural network with a gated recurrent unit, namely GGNN [19], is used to encode the nodes in the graph. GGNN is based on GRU [21] and enables the transmission of information within the graph. It primarily relies on edge transmission features and supports multiple edge types, allowing information to be transmitted between two nodes on an edge. In each time step, it constructs the input and output edge feature matrices to connect the features obtained from all edge types. The gated graph module is then used to update the current feature representation, and the output features are finally calculated.

Given a graph $G\,(V, E, T)$, where $v_i \in V$ represents the set of nodes, E represents a directed labeled edge

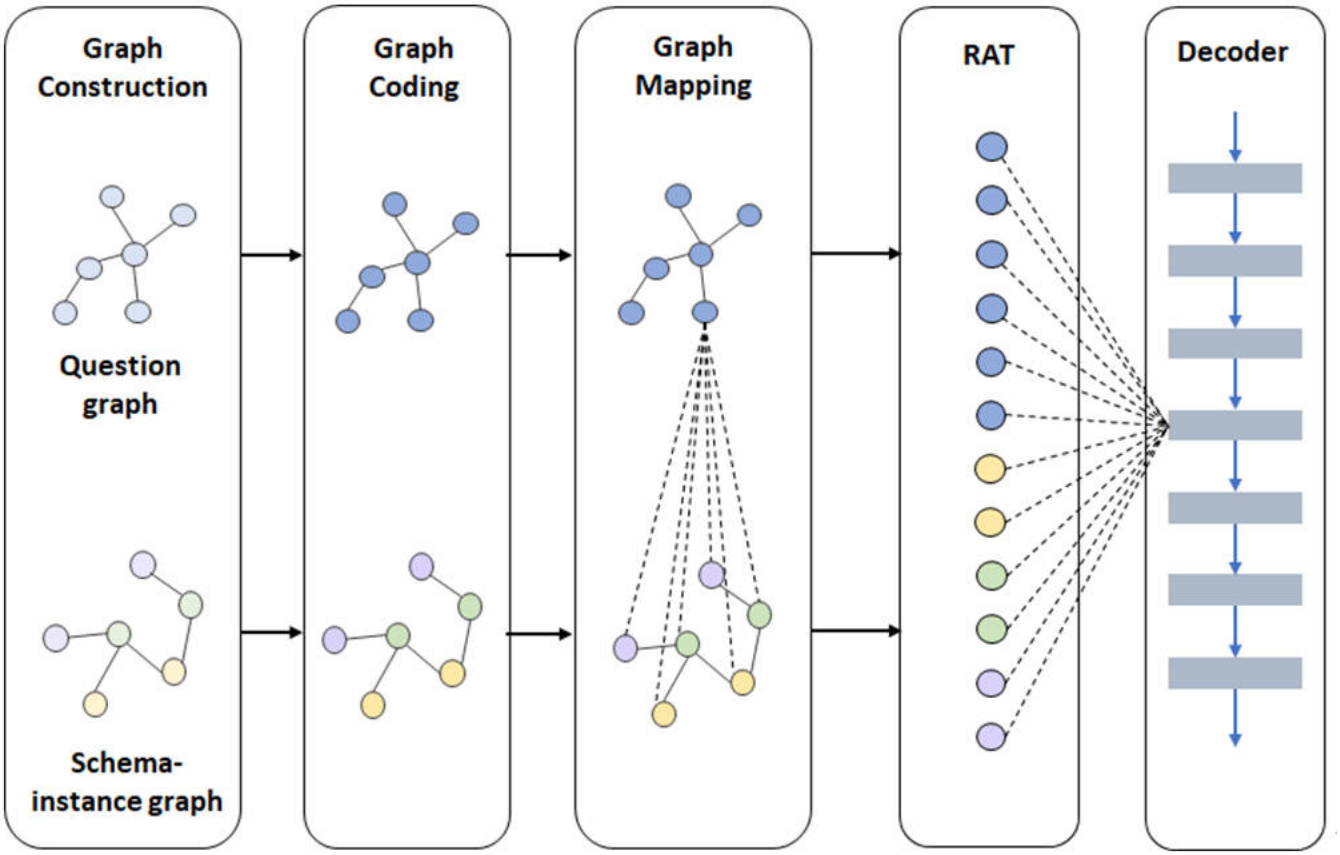

Fig. 3. Overview of the proposed model.

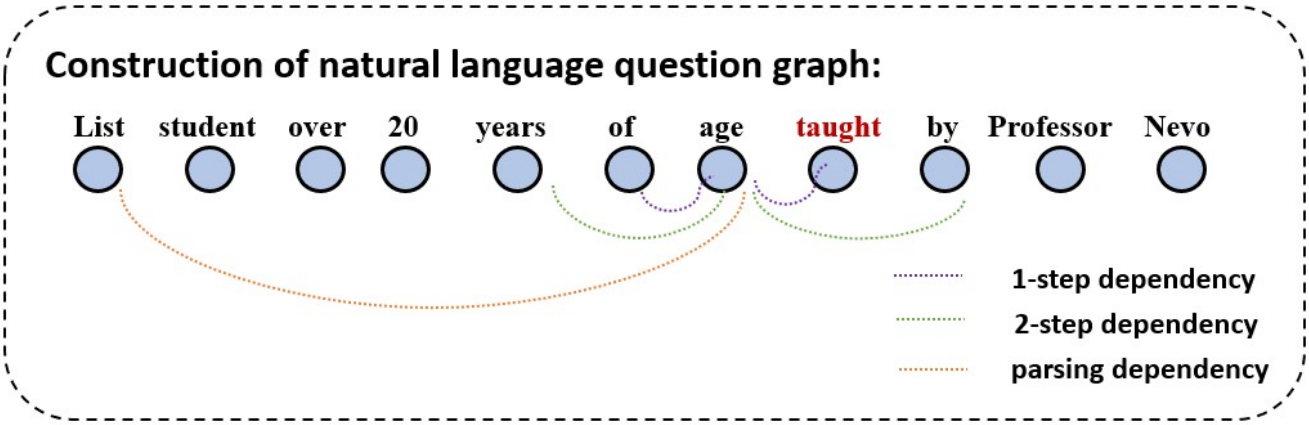

Fig. 4. Overview of the proposed model.

(v_s, T, v_d), with v_s as the source node, v_d as the target node, and T as the edge type. The encoding process consists of two stages: aggregating information and updating the node representations.

First, the representations of the neighboring nodes of the i-th node, $\mathrm{h}_k^{(l-1)}$, are aggregated. The specific calculation is shown in formula (2), where W_t and b_t are the trainable parameters for each edge type t.

$$\mathrm{f}_i^{(l)} = \sum_{t \in T} \sum_{(i,k) \in E_t} \left(W_t \mathrm{h}_k^{(l-1)} + b_t \right) \tag{2}$$

Secondly, the aggregate vector $\mathrm{f}_i^{(l)}$ is input to the GRU layer to update the representation of the node $\mathrm{h}_i^{(l-1)}$ from the previous step. The specific calculation is shown in formula (3).

$$\mathrm{h}_i^{(l)} = \mathrm{GRU}\left(\mathrm{h}_i^{(l-1)}, \mathrm{f}_i^{(l)} \right) \tag{3}$$

The advantage of GGNN is its ability to capture global semantic information and process any graph data, making it highly applicable to Text-to-SQL tasks. Compared to traditional sequence models, GGNN is more effective in capturing semantic information from natural language queries and integrating it with database schema and instance data to generate more accurate SQL queries.

*3) **Graph mapping**:* The graph mapping method aggregates information from the database model-instance relation-

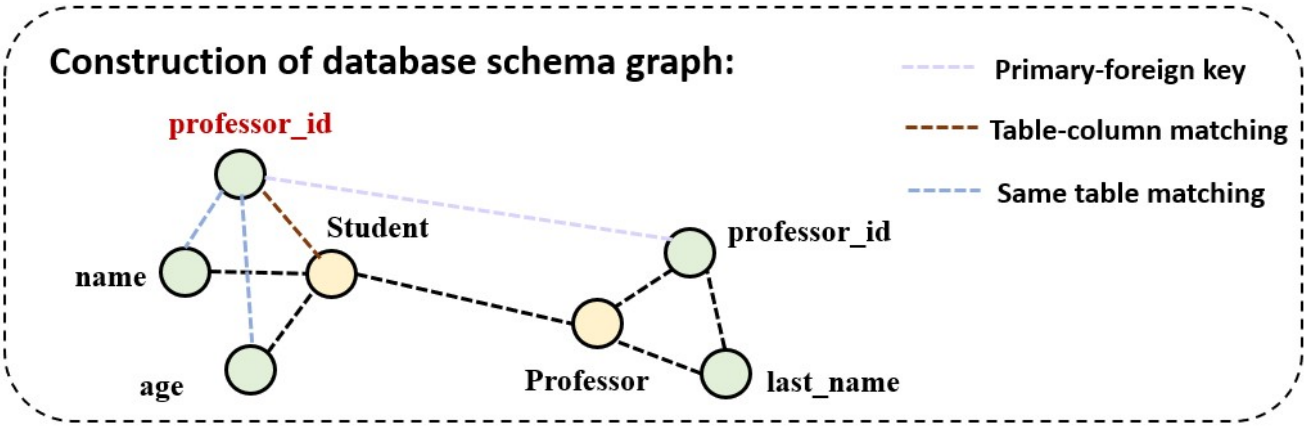

Fig. 5. Overview of the proposed model.

ship graph and the natural language question graph into a unified graph. This method enables the extraction of mapping information between the question graph G_Q and the database model-instance graph G_S. The specific mapping process is described by formulas (4) and (5). In this approach, the G_S graph is used to update the G_Q graph, and vice versa, the G_Q graph can be used to update the G_S graph. This paper demonstrates the implementation of the graph mapping method by using the G_S graph to update the G_Q graph.

$$\mathrm{G}_Q^{Map} = \text{GraphMap}\left(G_Q, G_S\right) \tag{4}$$

$$\mathrm{G}_S^{Map} = \text{GraphMap}\left(G_S, G_Q\right) \tag{5}$$

The embedded nodes in the natural language question graph are denoted as h_i^q, and the embedded nodes in the database schema-instance graph are denoted as h_j^k. These are vector representations obtained after dual-graph encoding. The graph mapping method updates the question graph using both global and local information from the nodes in the database schema-instance graph. The update process involves applying global average pooling to the node representation h_i^q in the question graph to obtain the global embedding representation h_{glob}^q. Subsequently, to incorporate globally relevant information, the node embedding h_j^k in the database schema-instance graph is updated, as described by formulas (6), (7), and (8).

$$\mathrm{h}_{glob}^q = \frac{1}{m}\sum_{i=1}^{m}\mathrm{h}_i^q \tag{6}$$

$$e_j = \theta\left(\mathrm{h}_{glob}^{q\ T} W_g \mathrm{h}_j^k\right) \tag{7}$$

$$\mathrm{h}_j^k = \left(1 - e_j\right) W_{qg}\mathrm{h}_{glob}^q + e_j W_{kg}\mathrm{h}_j^k \tag{8}$$

Where W_g, W_{qg}, and W_{kg} are trainable parameters, θ is the sigmoid function, and e_j represents the correlation score between the node in the j-th pattern-instance graph and the global query graph.

In the global linking process, each node in the problem graph evaluates the link score with nodes in the database schema-instance graph, represented by the global attention score $\alpha_{i,j}$. This score is computed by assessing the similarity between the node embedding h_i^q in the problem graph and the node embedding h_j^k in the database schema-instance graph. The specific calculation procedure is outlined in formula (9).

$$\alpha_{i,j} = \text{softmax}_j\left\{\sigma\left(\mathrm{h}_i^q W_q\left(\mathrm{h}_j^k + \mathrm{R}_{ij}^E\right)^T\right)\right\} \tag{9}$$

Where α is a nonlinear activation function, and R_{ij}^E represents the relationship feature of the pre-defined database pattern-instance graph between the i-th problem node and the j-th pattern-instance node.

In the local linking process, each node in the problem graph evaluates the link score between itself and the neighboring nodes in the database schema-instance graph, represented by the local attention score $\beta_{i,j,t}$. This score is computed by assessing the similarity between the i-th node in the problem graph and the t-th neighboring node of the j-th node in the database schema-instance graph. The detailed calculation procedure is provided in formula (10).

$$\beta_{i,j,t} = \text{softmax}_t\left\{\sigma\left(\mathrm{h}_i^q W_{nq}\left(\mathrm{h}_t^k + \mathrm{R}_{ij}^E\right)^T\right)\right\} \quad (t \in N_j) \tag{10}$$

Among them, N_j represents the set of neighboring nodes of the j-th node, and h_t^k is the vector representation of the t-th neighboring node of the pattern-instance graph.

C. Multi-feature fusion method based on RAT

In the multi-feature fusion method based on RAT, the RAT framework is used to learn a unified representation from three inputs: natural language queries, database schemas, and database instances. RAT, an extension of the Transformer model proposed by Vaswani et al. [5], incorporates predefined relationships into the self-attention mechanism. This enhancement allows the framework to integrate information from the graph mapping module, unifying the representations of natural language queries and database schema instances. The method combines these embedding representations and processes them through a multi-layer RAT neural network to achieve the fusion of the three features.

Given a set of inputs $X = \{x_i\}_{i=1}^n$, where $x_i \in \mathbb{R}^d$, r_{ij} represents the relationship between any two elements x_i and x_j in the input set X. The RAT layer consists of H head attentions, and each self-attention layer operates on the input

sequence x_i. Each information update transformation with respect to the relationship of x_i is represented as y_i.

In Text-to-SQL parsing, if certain relationships between inputs are known in advance, it is desired that the encoder prioritize them during encoding. Therefore, this paper proposes an encoding method that incorporates known relationships into the attention mechanism, as shown in formulas (11) and (12).

$$\mathrm{e}_{ij}^{(h)} = x_i \mathrm{W}_k^{(h)} \left(x_i \mathrm{W}_k^{(h)} + \mathrm{r}_{ij}^K \right)^T \tag{11}$$

$$\mathrm{z}_{ij}^{(h)} = \sum_{j=1}^{n} \alpha_{i,j}^{(h)} \left(x_i \mathrm{W}_K^{(h)} + \mathrm{r}_{ij}^V \right) \tag{12}$$

The term r_{ij} encodes the known relationship between two elements in the input. To obtain a unified encoding of the entire input graph and the question, this paper proposes the use of a relation-aware self-attention mechanism. The encoder's input consists of the joint representation of all nodes in the graph, including the table, column, value, and question. At each layer, self-attention is applied to all elements of the input graph to generate a new contextual representation that integrates the question terms with the database structure.

D. Instance-aware query parsing method

The instance-aware query parsing method uses a tree structure system [22] for decoding. This process translates the SQL query into an abstract syntax tree using a depth-first traversal order. Initially, the operation sequence of the abstract syntax tree is generated using LSTM [23]. Subsequently, the abstract syntax tree is converted into a sequential SQL query.

For input, the decoder receives the final representations of the question words, database table names, column names, and values from the RAT encoder. It has two output actions: first, it can expand the last generated node using the APPLYRULE grammar rule, which involves appending nodes to the derived abstract syntax tree (AST). Second, it selects a column, table, or value from the schema when finalizing a leaf node, corresponding to SELECTCOLUMN, SELECTTABLE, and SELECTVALUE. The goal of the decoder is to produce a series of rules that generate the SQL abstract syntax tree.

The probability of generating an abstract syntax tree is shown in formula (13).

$$Pr\left(P \mid h\right) = \prod_t Pr\left(Rule_t \mid Rule_{<t}, h\right) \tag{13}$$

Where P represents the SQL statement corresponding to the final generated abstract syntax tree, $h = [h^q, h^t, h^c, h^v]$ represents the final encoding of the natural language problem and the database schema-instance, $Rule_t$ is the rule generated at time t, and $Rule_{<t}$ represents all the rules before time t.

To select the table/column rule, the alignment matrices M^T, M^C, and M^V between the entity (problem word, table, column, value) and the table, column, and value are constructed. The relation-aware self-attention mechanism is then used as the pointer mechanism. The detailed implementation process is shown in formulas (14), (15), and (16).

$$\overline{\mathrm{M}}_{i,j}^T = h_i \mathrm{W}_Q^t \left(\mathrm{h}_j^t \mathrm{W}_K^t \right)^T, \quad \overline{\mathrm{M}}_{i,j}^T = \mathrm{softmax}_j \left\{ \overline{\mathrm{M}}_{i,j}^T \right\} \tag{14}$$

$$\overline{\mathrm{M}}_{i,j}^C = h_i \mathrm{W}_Q^c \left(\mathrm{h}_j^c \mathrm{W}_K^c \right)^T, \quad \overline{\mathrm{M}}_{i,j}^C = \mathrm{softmax}_j \left\{ \overline{\mathrm{M}}_{i,j}^C \right\} \tag{15}$$

$$\overline{\mathrm{M}}_{i,j}^V = h_i \mathrm{W}_Q^v \left(\mathrm{h}_j^v \mathrm{W}_K^v \right)^T, \quad \overline{\mathrm{M}}_{i,j}^V = \mathrm{softmax}_j \left\{ \overline{\mathrm{M}}_{i,j}^V \right\} \tag{16}$$

Where $M^T \in \mathbb{R}^{(|q|+|t|+|c|)\times|t|}$, $M^C \in \mathbb{R}^{(|q|+|t|+|c|)\times|c|}$, and $M^V \in \mathbb{R}^{(|q|+|t|+|c|)\times|v|}$. Finally, the score for the j-th column/table/value is calculated. The specific calculation process is shown in formulas (17) to (19).

$$Pr\left(Rule_t = \mathrm{Table}[j] \mid Rule_{<t}, h\right) = \sum_{i=1}^{|q|+|t|+|c|} \alpha_i \mathrm{M}_{i,j}^T \tag{17}$$

$$Pr\left(Rule_t = \mathrm{Column}[j] \mid Rule_{<t}, h\right) = \sum_{i=1}^{|q|+|t|+|c|} \alpha_i \mathrm{M}_{i,j}^C \tag{18}$$

$$Pr\left(Rule_t = \mathrm{Value}[j] \mid Rule_{<t}, h\right) = \sum_{i=1}^{|q|+|t|+|c|} \alpha_i \mathrm{M}_{i,j}^V \tag{19}$$

IV. EXPERIMENTS

A. Dataset

The Spider [24] dataset is the most widely used and challenging cross-domain Text-to-SQL benchmark. It is divided into three subsets: the training set, the development set, and the test set. The training set consists of 146 databases and contains 8,659 examples; the development set includes 20 databases with 1,034 examples; and the test set consists of 40 databases with 2,147 examples. The official Spider test set has not been released for evaluation, making it unavailable for direct benchmarking. However, the Spider dataset enables transfer learning by allowing Text-to-SQL systems to train on queries against the databases in the training set and evaluate their performance on queries against unseen databases in the development set.

B. Metrics

The exact match rate [25] is defined as the percentage of cases where the predicted SQL statement exactly matches the reference SQL statement. This metric measures whether the predicted query is identical to the ground truth query. It evaluates only the database schema, without considering the underlying data. The specific calculation method is shown in formula (20).

$$Acc_{qm} = \frac{\text{Problems predicting success}}{\text{All questions}} \tag{20}$$

C. Embedding Initialization

To enhance the robustness of the model, this paper initializes the embeddings for tables, columns, values, and natural language question words using pre-training methods. Among the embedding methods, GloVe is the most commonly used for initialization. For pre-trained language models, BERT is the most widely adopted embedding initialization method, specifically utilizing two models: BERT-base and BERT-large. Additionally, some pre-trained language models are specialized for specific fields, such as GAP, which leverages prior Text-to-SQL knowledge more effectively. Therefore, this paper employs four pre-training methods—GloVe, GAP, BERT-base, and BERT-large in the experiments.

D. Implementation

In the experimental parameter settings, the batch size is set to 20, the initial learning rate to 7×10^{-4}, and the maximum number of steps to 4,000. The Adam optimizer with default hyperparameters is used. For BERT configurations, a separate learning rate of 3×10^{-6} is applied for fine-tuning, the initial learning rate is adjusted to 2×10^{-4}, and the maximum number of training steps is increased to 90,000. All other settings remain unchanged and follow the original configurations.

E. Results

Table I presents the experimental results of the exact match between our model and other benchmark models on the Spider development and test sets.

As shown in Table I, without using any language pre-training model, the model proposed in this paper outperforms the RATSQL model, which only considers pattern links, by 2.1%; the GASQL model by 1.2%; the ValueNet model, which focuses only on instance value generation, by 2.8%; and the SADGA model, which considers both pattern links and instance value generation, by 0.1% in the Spider development set. The experiments demonstrate that the Text-to-SQL graph mapping model with integrated instances and the RAT-based instance-aware query parsing model proposed in this paper effectively improve the accuracy of Text-to-SQL tasks.

TABLE I
ACCURACY RESULTS ON THE SPIDER DEVELOPMENT SET AND TEST SET (%)

Model	Dev	Test
GNN	40.7	39.4
Global-GNN	52.7	47.4
IRNet	53.2	46.7
RAT-SQL	62.7	57.2
GASQL	63.6	58.5
ValueNet	62.0	-
SADGA	64.7	-
Ours	**64.8**	-

The comparative experimental results presented in Fig. 6, Fig. 7, and Fig. 8 show that the Text-to-SQL query parser model developed in this study effectively predicts correct SQL query statements. The model outperforms other benchmark models when enhanced with GloVe, BERT-base, and BERT-large pre-trained language models. Specifically, with GAP enhancement, the model achieves an accuracy of 73.2%, which is 1.4% higher than RATSQL and 0.1% higher than the SADGA experiment. Additionally, on the Spider test set, the model demonstrates compatibility with BERT-large and GAP models, confirming its effectiveness in improving Text-to-SQL tasks.

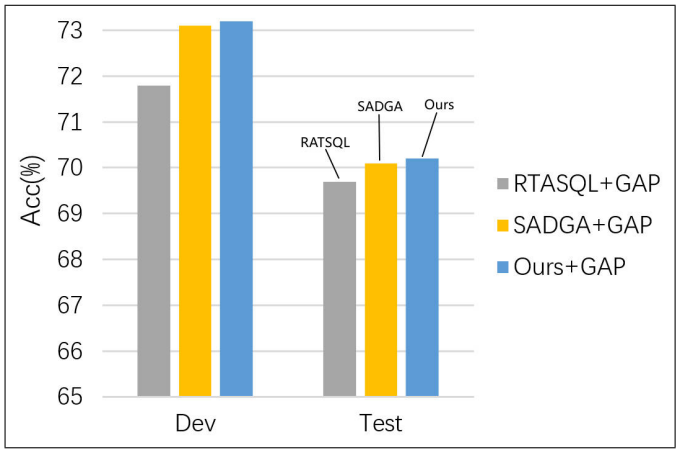

Fig. 6. Experimental results using GAP augmentation on the Spider development and test set (%).

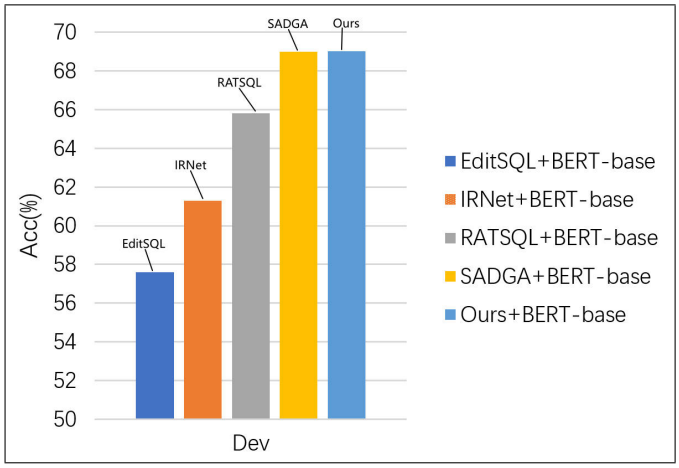

Fig. 7. Experimental results using BERT-base enhancements on the Spider development set (%).

V. CONCLUSION

This paper presents a Text-to-SQL query parser that integrates database instance content with a neural semantic parser. By utilizing database instance information and employing a relation-aware self-attention mechanism, the approach achieves effective fusion encoding of natural language questions and database schema-instance graphs. This method improves the model's accuracy and generalization capabilities. Additionally, a stack-based decoder and an abstract syntax tree are used to generate SQL query statements, further enhancing the model's precision. Future work will explore the application of transfer learning techniques to enable the model to adapt across different domains.

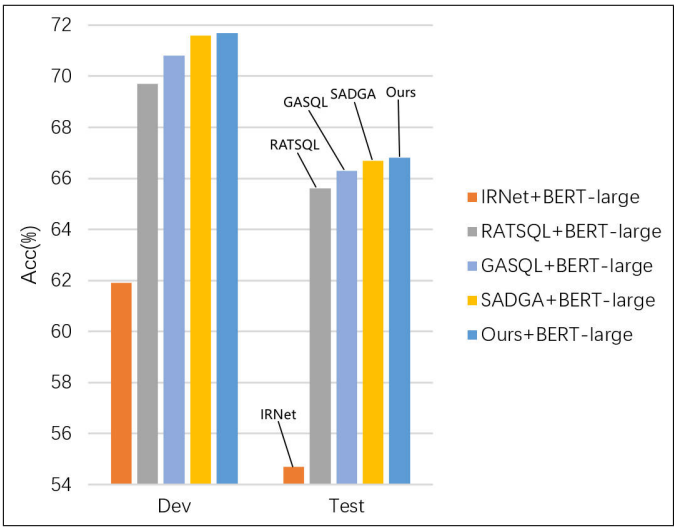

Fig. 8. Experimental results using BERT-large enhancements on the Spider development and test set (%).

REFERENCES

[1] D. Gao, H. Wang, Y. Li, X. Sun, Y. Qian, B. Ding, and J. Zhou, "Text-to-sql empowered by large language models: A benchmark evaluation," *arXiv preprint arXiv:2308.15363*, 2023.

[2] G. Ding, "Research on record named entity recognition of chinese electronic medical based on lstm-crf," in *Lecture Notes in Engineering and Computer Science: Proceedings of The International MultiConference of Engineers and Computer Scientists*, 2023.

[3] A. Ajiboye, M. Olumoye, D. Aleburu, A. Olayiwola, D. Olayiwola, and S. Ajose, "Dimensionality reduction for deep learning based intrusion detection systems for iot," in *Lecture Notes in Engineering and Computer Science: Proceedings of The International MultiConference of Engineers and Computer Scientists*, 2023, pp. 76–81.

[4] A. B. KANBUROĞLU and F. B. TEK, "Text-to-sql: A methodical review of challenges and models," *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 32, no. 3, pp. 403–419, 2024.

[5] B. Wang, R. Shin, X. Liu, O. Polozov, and M. Richardson, "Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers," *arXiv preprint arXiv:1911.04942*, 2019.

[6] B. Bogin, M. Gardner, and J. Berant, "Representing schema structure with graph neural networks for text-to-sql parsing," *arXiv preprint arXiv:1905.06241*, 2019.

[7] R. Cao, L. Chen, Z. Chen, Y. Zhao, S. Zhu, and K. Yu, "Lgesql: line graph enhanced text-to-sql model with mixed local and non-local relations," *arXiv preprint arXiv:2106.01093*, 2021.

[8] L. Wang, B. Qin, B. Hui, B. Li, M. Yang, B. Wang, B. Li, J. Sun, F. Huang, L. Si *et al.*, "Proton: Probing schema linking information from pre-trained language models for text-to-sql parsing," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 1889–1898.

[9] Y. Liu, Y. Hu, Z. Li, and Z. Zhu, "Graph alignment for cross-domain text-to-sql," in *2022 7th International Conference on Intelligent Computing and Signal Processing (ICSP)*. IEEE, 2022, pp. 1937–1940.

[10] R. Cai, J. Yuan, B. Xu, and Z. Hao, "Sadga: Structure-aware dual graph aggregation network for text-to-sql," *Advances in Neural Information Processing Systems*, vol. 34, pp. 7664–7676, 2021.

[11] Y. Zhou, H. Chen, J. Xu, Q. Dou, and P.-A. Heng, "Irnet: Instance relation network for overlapping cervical cell segmentation," in *Medical Image Computing and Computer Assisted Intervention–MICCAI 2019: 22nd International Conference, Shenzhen, China, October 13–17, 2019, Proceedings, Part I 22*. Springer, 2019, pp. 640–648.

[12] Z. Gu, J. Fan, N. Tang, L. Cao, B. Jia, S. Madden, and X. Du, "Few-shot text-to-sql translation using structure and content prompt learning," *Proceedings of the ACM on Management of Data*, vol. 1, no. 2, pp. 1–28, 2023.

[13] S. Abbas, M. U. Khan, S. U.-J. Lee, A. Abbas, and A. K. Bashir, "A review of nlidb with deep learning: findings, challenges and open issues," *IEEE Access*, vol. 10, pp. 14 927–14 945, 2022.

[14] X. V. Lin, R. Socher, and C. Xiong, "Bridging textual and tabular data for cross-domain text-to-sql semantic parsing," *arXiv preprint arXiv:2012.12627*, 2020.

[15] J. Devlin, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[16] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[17] B. Bogin, M. Gardner, and J. Berant, "Global reasoning over database structures for text-to-sql parsing," *arXiv preprint arXiv:1908.11214*, 2019.

[18] Z. Chen, L. Chen, Y. Zhao, R. Cao, Z. Xu, S. Zhu, and K. Yu, "Shadowgnn: Graph projection neural network for text-to-sql parser," *arXiv preprint arXiv:2104.04689*, 2021.

[19] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[20] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky, "The stanford corenlp natural language processing toolkit," in *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, 2014, pp. 55–60.

[21] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[22] S. Hochreiter, "Long short-term memory," *Neural Computation MIT-Press*, 1997.

[23] P. Yin and G. Neubig, "A syntactic neural model for general-purpose code generation," *arXiv preprint arXiv:1704.01696*, 2017.

[24] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman *et al.*, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task," *arXiv preprint arXiv:1809.08887*, 2018.

[25] U. Brunner and K. Stockinger, "Valuenet: A natural language-to-sql system that learns from database information," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 2177–2182.