ISSN: 2710-0588

Syntax and Relation Enhanced Query Generation for Text to SQL Parsing

Kun Han[1,2], Xi Xiong[1,2,*]

[1] School of Cybersecurity, Chengdu University of Information Technology, Chengdu 610225, China

[2] Advanced Cryptography System Security Key Laboratory of Sichuan Province, Chengdu 610225, China

Abstract

Text-to-SQL parsing is the task of converting natural language questions into executable SQL queries, a significant branch of semantic parsing, which has gained increasing attention in recent years. This technology lowers the barrier for people to access databases, enhancing the convenience and availability of data. However, the primary challenge for text-to-SQL parsing lies in domain adaptation, which concerns whether the model can be applied to new databases and effectively align natural language questions with the corresponding tables or columns within the database. To address these issues, research has introduced SRSQL (Syntax and Relation-Augmented Query Generation), which incorporates syntax information and predefined relationships into the model, effectively utilizing syntactic dependencies and pattern linking to improve performance. Using a Transformer-based decoder, SRSQL generates SQL queries in the form of Abstract Syntax Trees (AST), significantly boosting prediction accuracy. Experimental results show that SRSQL outperforms comparative models, particularly on challenging benchmarks like Spider and Spider-SYN.

Keywords

Text-to-SQL; Syntactic Dependencies; Transformer; Abstract Syntax Trees.

1. Introduction

As the era of information technology advances, vast amounts of data are generated daily from individuals and businesses. To facilitate management and operations, these data are commonly stored in relational databases. To retrieve the needed data from these databases, Structured Query Language (SQL) or similar structured query languages are employed. Despite the growing popularity of relational databases, non-expert users often face limitations in accessing information due to the need to understand complex query languages, creating a certain level of usability barrier. Consequently, Text-to-SQL [1] parsing has garnered significant attention, aiming to directly convert natural language questions into corresponding SQL query statements. This task alleviates the challenges faced by non-expert users when interacting with relational databases, to a certain extent.

Developing large-scale annotated question datasets along with corresponding SQL queries has propelled advancements in the field. In contrast to previous efforts on parsed datasets [2], the new datasets such as WikiSQL [3] and Spider [4] heavily test models' ability to generalize to unseen database schemas. Each query in these new tasks is based on a multi-table database architecture, and database schemas do not repeat between training and test sets. A key issue in achieving domain generalization [5] is the need for complex reasoning to generate structurally rich SQL queries. To accurately contextualize user queries with specific databases, this involves explicit relationships (such as those defined by the database schema for tables and

ISSN: 2710-0588

columns) and implicit relationships (such as determining whether phrases in the query correspond to specific columns or tables).

However, this implies that models need to predict queries in database contexts unseen during training and accurately express query intentions through SQL logic. Therefore, text-to-SQL parsers across databases cannot solely rely on observed SQL schema; they must also accurately model natural language questions, database structures, and the context between them.

Current research typically adopts the following strategies to enhance the cross-domain generalization ability of models. First, by learning schema-based embedding functions, questions and database schemas are contextualized mutually [6]. Secondly, pretrained language models such as BERT [7] and RoBERTa [8] have been shown to improve prediction accuracy by acquiring semantic relationships in different contexts and capturing distant dependencies. Methods that incorporate syntax-enhanced synthetic examples into the BERT pretraining framework, alongside a basic semantic parser, have shown promising results.

This study introduces SRSQL, which integrates joint encoding of questions and schemas into a novel Transformer [9] variant for text-to-SQL parsing. By representing each natural language question as a graph with multiple relations, including syntactic dependencies and part-of-speech, and a database schema as a graph composed of tables, columns, and their relationships, SRSQL employs a relation-aware Transformer to learn the connections between the schema and the question. A Transformer-based tree decoder is then proposed to generate SQL queries in Abstract Syntax Tree (AST) form. Experiments on the Spider benchmark show that SRSQL attains a 74.5% Exact Match (EM) accuracy on the test set, outperforming the baseline model significantly.

2. Related Work

In earlier research, a commonly used approach was sketch-based slot filling, which employs different modules to predict various parts of the generated SQL query. This method decomposes the SQL generation task into several independent sketches and utilizes different classifiers to predict each part, such as SQLNet [10], TypeSQL [11], RYANSQL [12], X-SQL [13], among others. Most of these methods only handle simple queries, making it challenging to apply them to more complex scenarios.

There are various methods to address the challenges posed by complex SQL tasks. One common approach is to use generic neural network-enhanced encoders for global reasoning over natural language questions and database schemas. For instance, IRNet [14] employs LSTM and self-attention mechanisms to encode questions and schemas separately, while BRIDGE [15] serializes questions and schemas into token sequences and maximizes the utilization of BERT and database content to capture the linking relationships between questions and schemas.

On the other hand, many studies utilize graph structures to represent a series of complex relationships. For example, Global-GNN [16] employs graph neural networks (GNN) to output queries selecting subsets of tables or columns, while ShadowGNN [17] introduces a graph projection neural network to abstract representations of questions and schemas. Further developments include SADGA [18], which separately encodes question graphs and schema graphs based on dependency relationship structures and database schemas, and SDSQL [19], which improves structured reasoning by modeling relationships between schemas and questions.

Recent work has demonstrated the effectiveness of fine-tuning pretrained models. For instance, Shwa et al. [20] showed that fine-tuning the pretrained T5-3B model can yield highly competitive results. Building upon this, PICARD [21] was introduced, a technique that restricts the autoregressive decoder by applying incremental parsing during inference. It real-time filters

ISSN: 2710-0588

out syntactically incorrect sequences during beam search, significantly enhancing the quality of generated SQL.

In summary, due to the complexity of semantic understanding and data dependency relationships, generating complex query statements can lead to syntax errors, inappropriate semantics, and database connection errors. The SRSQL model proposed in this paper effectively alleviates these issues.

3. Question Definition

The task aims to transform the input natural language question Q into the corresponding Abstract Syntax Tree (AST) representation of the SQL query y, given a database schema S= (T, C).

Specifically, Let Q = (q_1, q_2, ..., $q_{|Q|}$) be a sequence of natural language tokens. $|Q|$ is the length of the natural language problem. The database schema S includes multiple tables T = (t_1, t_2, ..., $t_{|T|}$) and column C = (c_1, c_2, ..., $c_{|C|}$), $|T|$ and $|C|$ respectively represent the number of tables and columns in the database. The table name for each table is denoted as $t_i \in T$, it can be represented as a sequence of tokens t_i=(t_{i1},t_{i2},⋯,$t_{i|t_i|}$), $|t_i|$ represents the number of tokens for table names, Similarly, each column name in table t is represented as c_i={c_{i1},c_{i2},⋯,$c_{i|c_i|}$}.$|c_i|$ represents the number of tokens for column names.

4. Model

4.1. Model Framework

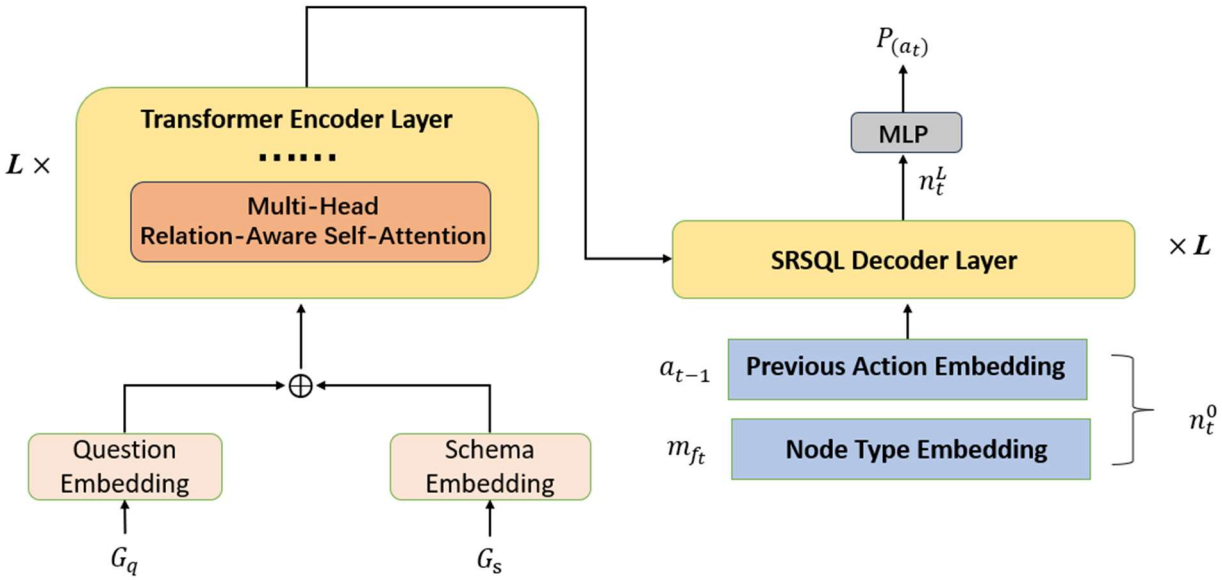

Figure 1. Model Framework Diagram

The overall structure of the SRSQL model is illustrated in Figure 1. Model inherits the end-to-end capabilities of Transformer and treats the text-to-SQL problem as a translation task. The model consists of L layers of encoders and decoders, with the encoding of the question and the database schema concatenated as the model's input. We utilize a relation-aware Transformer [22] as the encoder (see Section 4.3), which employs relation-aware self-attention [23] to replace the original self-attention mechanism. Additionally, we extend the Transformer decoder (see Section 4.4), integrating node types and embeddings of previous actions to autoregressively generate SQL queries. These queries are a series of actions derived according to SQL syntax.

ISSN: 2710-0588

4.2. Question Graph and Schema Graph Construction

In this section, we will present the process of creating the model's input. The input to the model is a combination of a natural language question and a database schema. To represent these, we employ a graph-based modeling approach, with the detailed methodology described below.

Question Graph Construction: The natural language question can be represented as a graph $G_Q = (Q, R)$, Where the node set Q consists of natural language tokens, $R = (r_1, r_2, \ldots, r_{|R|})$ represents the relationships between words. In this work, we construct a graph using the syntactic dependency relationships between words in the question. Figure 2 is an example of a question relation graph, showing the part-of-speech tagging and dependency relations for each token. In this example, the token "names" has an object dependency relation with the token "Find", and both "Find" and "names" are tagged as a verb (VB) and a noun (NN), respectively. We employ a Graph Attention Network (GAT) [24] to encode the question graph $Z_i \in R^d$, where $i \in \{1, \ldots, |Q|\}$, d is the size of the hidden layer.

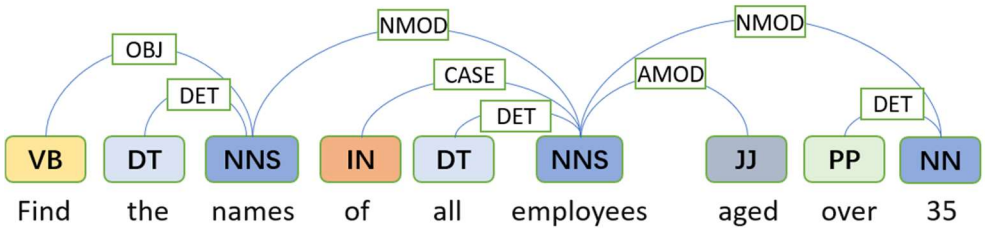

Figure 2. Example of a Question Graph with Part-of-Speech Tagging and Dependency Relations

Database Schema Graph Construction: The database schema graph is represented as a graph $G_S = (S, R)$, where Node Set $S = (T, C)$ representing tables T and columns C in the database schema. The text is written in Chinese. Here's the translation into English: $R=\{r_1,\ldots,r_{|R|}\}$represents the structural relationship between entities and attributes in a schema. We employ classic database-specific relationships, such as whether a column belongs to a table, whether it is a primary key of that table, and whether it is a foreign key referencing another column. Figure 3 illustrates an example of a database schema diagram. We also encode the database schema diagrams using GAT [24] and obtain vector representations for each database schema through global average pooling.

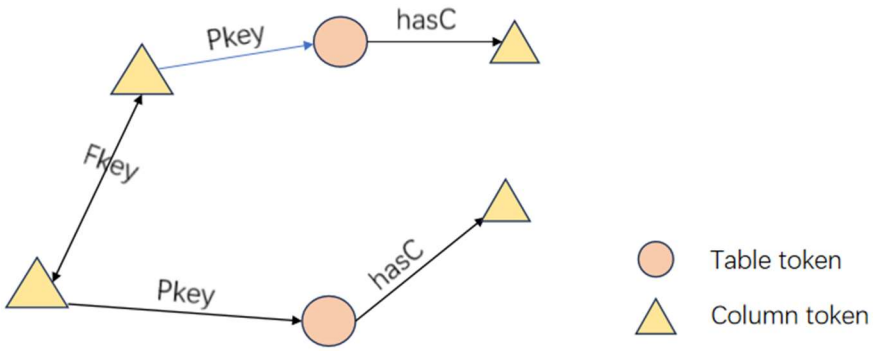

Figure 3. Example of a Database Schema Graph

4.3. Relational-Aware Transformer Encoder

The relational-aware Transformer replaces the conventional self-attention mechanism in Transformers with relation-aware self-attention. This is an embedding module designed for semi-structured sequences, which jointly encodes both the inherent relationships between

ISSN: 2710-0588

elements within the input and predefined relations. We utilize it to encode the linkages between questions and database schemas, which will be elaborated on in Section 4.4.

Suppose that the input to each encoder layer in the original Transformer model is an n-dimensional word vector sequence $X=(x_1,\ldots,x_n)$, where $x_i \in R^{d_x}$ represents the input, and the output is a new sequence y where $y_i \in R^{d_y}$. The model consists of multiple stacked self-attention layers, each containing H attention heads. The output element y is calculated by first linearly transforming the input elements x and then performing a weighted sum using self-attention, following this computational process:

$$e_{ij}^{(h)} = \frac{x_i W_Q^{(h)} (x_j W_K^{(h)})^T}{\sqrt{d_z/H}}, \alpha_{ij}^{(h)} = softmax\left\{e_{ij}^{(h)}\right\} \tag{1}$$

$$z_i^{(h)} = \sum_{i=1}^{n} \alpha_{ij}^{(h)} (x_j W_V^{(h)}), z_i = Concat(z_i^{(1)}, \cdots, z_i^{(H)}) \tag{2}$$

$$\bar{y}_i = LayerNorm(x_i + z_i) \tag{3}$$

$$y_i = LayerNorm(\bar{y}_i + FC(ReLU(FC(\bar{y}_i)))) \tag{4}$$

Where $W_Q, W_k, W_V \in R^{d_x \times d_z}$ stands for the learnable weight matrix, $1 \leq h \leq H$, where FC refers to a fully connected layer, and Layer Norm denotes layer normalization (Layer Normalization). In a Transformer model, each attention head in every layer computes implicit relationships between input elements, with the strength of these relationships encoded in the attention weights α_{ij}. However, in practical applications, if we have prior knowledge about certain relationships between input elements and wish to guide the model to learn these, we can employ a relation-aware Transformer by incorporating explicit relationships into the attention module. This is done by modifying equations (1) and (2) as follows:

$$e_{ij}^{(h)} = \frac{x_i W_Q^{(h)} (x_j W_K^{(h)} + r_{ij}^K)^T}{\sqrt{d_z/H}} \tag{5}$$

$$z_i^{(h)} = \sum_{i=1}^{n} \alpha_{ij}^{(h)} (x_j W_V^{(h)} + r_{ij}^V) \tag{6}$$

4.4. Schema Linking

Schema linking refers to the operation of aligning tables or columns mentioned in a natural language question with those in a database. We utilize relation-aware self-attention for encoding the schema linking relationships. To model the relationship of schema links, we introduced the interaction graph $G_R = (V, \beta_R)$, which has a structure similar to a database schema diagram. In this graph, $V = Q \cup T \cup C$ encompasses problem keywords, table names, and column names, with $\beta_R = \beta \cup \beta_{Q \leftrightarrow S}$representing the schema link relationship between the problem words and the database schema. schema linking generally occurs in two ways: name-based linking and content-based linking, and we will delve into these methods in detail next.

ISSN: 2710-0588

4.4.1. Name-based Linking

We follow the approach employed by Wang et al. [22], using n-gram matching to determine the degree of match between items mentioned in the question and the corresponding items in the schema: whether it matches the table or column names exactly or partially. Consequently, for each edge (i,j) in the relationship interaction graph, we categorize the relationship based on the types of x_i and x_j. The types of relationships we consider are: QUESTION-COLUMN-M and QUESTION-TABLE-M. Where M is one of EXACTMATCH, PARTIALMATCH and NOMATCH. These relationships are all asymmetric.

4.4.2. Database Content Linking

If the question directly mentions the value in a column of the database, rather than mentioning the table name or column name, database content linkage can be used. We follow the same process as Lin et al. [15] to capture the mentioned database content. Firstly, fuzzy string matching is performed between the question tokens and the values in each column of the database. Then, the matched values are inserted into the input sequence after the corresponding column name. This relationship is represented as VALUE-MATCH.

4.5. Autoregressive Tree Decoder

In the decoding phase, previous work primarily relied on LSTM-based decoders to generate Abstract Syntax Tree (AST). Our approach, however, extends the original Transformer decoder in an autoregressive manner for AST generation. This method offers an advantage over LSTM-based decoders by better preserving the context of previously generated query parts for longer sequences.

The decoder structure of SRSQL, as depicted in Figure 4, differs from the original Transformer decoder. It does not employ masked self-attention but utilizes cross-attention instead, with H heads. The input to the decoder consists of the hidden states from the encoder, the current node type, and the action from the previous time step. The node type is incorporated as a residual term within the multi-head attention mechanism. The decoder then goes through L layers, ultimately outputting probabilities for generating the next action at each time step.

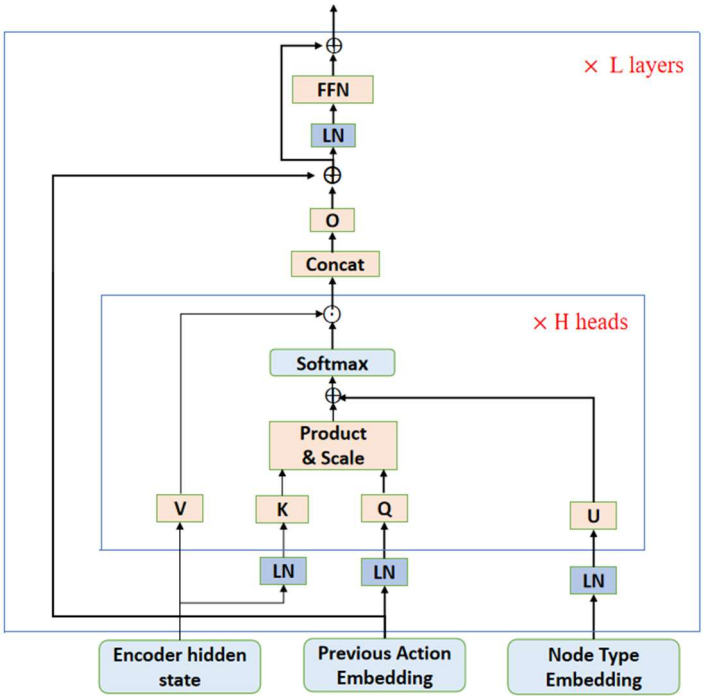

Figure 4. Decoder structure diagram

ISSN: 2710-0588

In the SRSQL decoder, each node possesses two attributes: a node type and the action from the previous time step. We denote the vector representation of the current node type as m_{f_t}, and the vector representation of the action from the previous time step as a_{t-1}.

Specifically, node types include SQL keywords, table names, and column names from the database. Decoding actions are divided into two categories: (1) applying the generated rules to the current syntax tree, which is the APPLYRULE action, and (2) selecting a table or column from the database schema, which are the SELECTTABLE and SELECTCOLUMN actions. The process of generating the Abstract Syntax Tree (AST) involves sequential application of these actions, with a depth-first traversal order for constructing a SQL query *y*. As shown in Figure 5, SQL statements based on the AST are generated using context-free grammar.

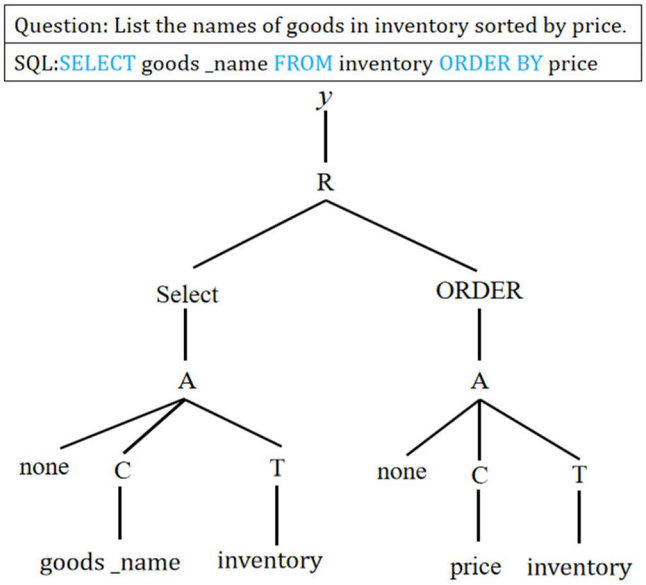

Figure 5. Example of an Abstract Syntax Tree

When incorporating m_{f_t} as a bias term in the calculation of attention, the formula takes the following form:

$$A = \frac{QK^T}{\sqrt{d_k}} + U \tag{7}$$

$$U = W_m \times m_{f_t} \tag{8}$$

Here, the query vector Q is derived from the attention vector a_{t-1}, while the key vector K and value vector V come from the hidden states of the encoder. The vector U is derived from the node type vector m_{f_t}, with W_m being a learnable weight matrix. In the l-th layer, the residual update for the node vector n_t^l can be represented as follows, where || denotes concatenation, and H is the number of attention heads:

$$n_t^l = n_t^{l-1} + O^l ||_{h=1}^{H} \sum_{j=1}^{N} \left(softmax\left(A_h^l\right) V_h^l \right) \tag{9}$$

ISSN: 2710-0588

Consequently, after passing through L layers of the decoder, the decoder state n_t^lat the current time step is fed into an action output MLP (Multi Layer Perceptron), which computes the probability distribution $P(a_t)$ for the APPLYRULE action at that step, with the specific calculation formula being:

$$P(a_t = APPLYRULE[R]|a_{<t}, y) = softmax(W_R g(n_t^L)) \tag{10}$$

Here, g $(\cdot)$represents an MLP with a tanh activation function, and $a_{<t}$denotes all action pairs up to time step t for the SELECTTABLE action. We calculate using the following formula:

$$P(a_t = SELECTTABLE[i]a_{<t}, y) = softmax(W_S n_t^L) \tag{11}$$

The calculation method for the "SELECT COLUMN" action is similar, and the prediction of the AST form of the final SQL query can be decoupled into a sequence of actions $a = (a_1, \cdots, a_{|a|})$, with the resulting training objective being:

$$L = -\sum_{p=1}^{|a|} \log P(a_p|a_{<p}, S, Q) \tag{12}$$

5. Experiment

5.1. Dataset

Our experiment employed the Spider and Spider-Syn datasets [25]. The Spider dataset is a large, complex, and cross-domain semantic parsing and text-to-SQL dataset, consisting of nine classic datasets like Scholar [26], WikiSQL, GeoQuery [27], etc. It contains 8,659 training samples, 1,034 development samples, and 2,147 test samples, spanning 138 domains across 200 complex databases. The Spider-Syn dataset, derived from the Spider benchmark, is a manually curated dataset where NL problems are modified from Spider by replacing words related to patterns with synonyms chosen to reflect real-world problem interpretations. It consists of 7,000 training samples and 1,034 development samples.

5.2. Assessment Metrics

Following Yu et al.'s [3] metrics, we compute the Exact Match accuracy (EM) for all examples, grouped by difficulty. This is done by dividing the predicted SQL and the actual SQL into distinct subsets based on keywords, and then checking if the predicted set matches the actual one. EM assesses whether the predicted SQL query matches the ground truth exactly. Like previous work on Spider, these metrics do not consider the model's performance in generating values within the SQL.

5.3. Model Configuration

We utilized stanza [28] for tokenization, word segmentation, part-of-speech tagging, and dependency parsing. For training, we set the maximum input length to 1024, the maximum number of generated nodes in the AST to 200, the batch size to 32, and the maximum training steps to 40,000. The encoder and decoder had 6 layers with a dimension of 512 and 8 attention heads. The vector dimensions for tables and columns were set to 512, and the embeddings for node types and actions were of size 512. We employed Adafactor as the optimizer, with a learning rate of 1e-4 and a dropout rate of 0.1.

ISSN: 2710-0588

5.4. Comparative Experiment

The EM accuracy of the model on the Spider dataset is presented in Table 1, where we initially compared it with other advanced models. In Table 1, our proposed model demonstrates competitive performance. Our method outperforms T5-3B+ PICARD [21], which heavily fine-tunes a language model with a large number of parameters, by 2.6% on the test set, indicating that our approach can still exhibit strong effectiveness with fewer parameters. Compared to $S^2$2SQL [29], our model achieves a 2.4% absolute improvement in EM.

Table 1. EM results on Spider's test set and development set.

Method	Dev	Test
RATSQL + BERT[22]	69.7	65.6
ShadowGNN + RoBERTa[17]	72.3	66.1
SADGA + GAP[18]	73.1	70.1
LGESQL + ELECTRA[30]	75.1	72.0
RASAT+PICARD[31]	75.3	70.9
T5-3B + PICARD [21]	75.5	71.9
S^2SQL + ELECTRA [29]	76.4	72.1
ours	77.2	74.5

Furthermore, we conducted a fine-grained analysis of accuracy based on the query difficulty levels defined by Yu et al. [3] (easy, medium, hard, and very hard). In Table 2, we compare the EM precision of our method against the latest baseline for these four query difficulty subsets. As expected, the model's performance significantly deteriorates with increasing query complexity, with the accuracy dropping from 92.2% for simple queries to 50.6% for very hard queries. In terms of the most complex query types, SRSQL outperforms RAT-SQL [22] by 7.2% and 7.7% in hard and very hard queries, respectively. This demonstrates the ability of our Transformer-based SQL decoder to capture longer sequence contexts. Moreover, SRSQL consistently outperforms the baseline across all four subsets, providing evidence for the effectiveness of our approach.

Our model was also tested for EM accuracy on the Spider-SYN dataset, as shown in Table 3. The performance of SRSQL is superior to all baseline models, indicating that our model maintains good robustness when facing more flexible and complex problems.

Table 2. EM accuracy of Spider queries across different difficulty levels.

Method	Easy	Medium	Hard	Extra
RAT-SQL+BERT[22]	86.4	73.6	62.1	42.9
SADGA[18]	90.3	72.4	63.8	49.4
LGESQL[30]	91.5	76.7	66.7	48.8
GRAPHIX-T5-3B[32]	91.9	81.6	61.5	50
ours	92.2	82.5	69.3	50.6

Table 3. EM results on the development set of Spider-SYN.

Method	EM accuracy (%)
IRNet[14]	28.4
RAT-SQL+BERT[22]	48.2
LGESQL+ELECTRA[30]	64.6
GRAPHIX-T5-3B[32]	66.9
Ours	67.5

ISSN: 2710-0588

5.5. Ablation Experiment

To better validate the effectiveness of each component in our model, we conducted a series of ablation experiments on the development set of the Spider dataset. As shown in Table 4, we tested the impact of removing four crucial design elements from the model: syntax dependencies from the input, part-of-speech tagging, pattern linking in the encoder, and the use of an LSTM-based decoder.

Table 4. Ablation study of EM accuracy on the development set for SRSQL (±95% confidence interval)

Method	Scheme 1
SRSQL	77.2 ± 0.76
SRSQL w/o syntactic dependency	76.1 ± 0.73
SRSQL w/o Part-of-Speech tagging	76.3 ± 0.56
SRSQL w/o schema linking relations	73.8 ± 0.80
SRSQL encoder + LSTM-based decoder	72.9 ± 0.38

As shown in Table 4, the most significant impact among these designs came from the choice of decoder. After switching to an LSTM-based decoder, the EM accuracy dropped from 77.2% to 72.9%, resulting in a 4.3% decrease in performance, highlighting the superiority of a Transformer-based decoder. Second, the removal of pattern linking had a considerable effect on the model, with EM accuracy dropping by 3.4%. This is because the task of matching questions to database schemas became more challenging, and previous research has already confirmed the importance of this component for text-to-SQL parsing. Lastly, removing syntax dependencies and part-of-speech tagging had a smaller effect on performance, with decreases of 1.1% and 0.9% respectively.

6. Conclusion

In this paper, we present SRSQL, a syntax and relation-enhanced text-to-SQL parser that stands out with its autoregressive SQL query prediction based on Transformers. By incorporating relation-aware self-attention, SRSQL integrates pattern linking relationships into its encoder. The Transformer-based tree decoder, grounded in joint encoding, integrates node types and prior actions to generate SQL queries during the learning process. Notably, SRSQL demonstrates state-of-the-art performance on the Spider and Spider-SYN datasets. However, our model still has some limitations, with this study primarily focusing on the grammatical aspects of text-to-SQL conversion. Future work could explore incorporating large pre-trained language models or leveraging techniques from large-scale prompt-based models.

References

[1] John M. Zelle and Raymond J. Mooney: Learning to parse database queries using inductive logic programming. Proceedings of the thirteenth national conference on Artificial intelligence (Portland, Oregon, 1996). Vol.2, p1050–1055.

[2] Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev: Improving Text-to-SQL Evaluation Methodology. Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Melbourne, Australia, July ,2018). Vol.1, p351–360.

[3] Victor Zhong, Caiming Xiong, and Richard Socher: Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning.arXiv:1709.00103, 2017.

ISSN: 2710-0588

[4] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev: Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (Brussels, Belgium, October-November ,2018), p3911–3921.

[5] Binyuan Hui, Ruiying Geng, Lihan Wang, Bowen Qin, Bowen Li, Jian Sun, and Yongbin Li: S^2SQL: Injecting syntax to question-schema interaction graph encoder for text-to-sql parsers.2022. arXiv:2203.06958.

[6] Wonseok Hwang, Jinyeong Yim, Seunghyun Park, and Minjoon Seo: A Comprehensive Exploration on WikiSQL with Table-Aware Word Contextualization. arXiv:1902.01069, 2019.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Minneapolis, Minnesota, June,2019).

[8] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov: RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv:1907.11692, 2019.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin: Attention Is All You Need. arXiv:1706.03762, 2017.

[10] Xiaojun Xu, Chang Liu, and Dawn Song: SQLNet: Generating Structured Queries From Natural Language Without Reinforcement Learning.arXiv:1711.04436, 2017.

[11] Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev: TypeSQL: Knowledge-based Type-Aware Neural Text-to-SQL Generation.ArXiv:1804.09769, 2018.

[12] DongHyun Choi, Myeong Cheol Shin, EungGyun Kim, and Dong Ryeol Shin: RYANSQL: Recursively Applying Sketch-based Slot Fillings for Complex Text-to-SQL in Cross-Domain Databases.ArXiv: 2004.03125,2020.

[13] Pengcheng He, Yi Mao, Kaushik Chakrabarti, and Weizhu Chen: X-SQL: reinforce schema representation with context. ArXiv:1908.08113, 2019.

[14] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao,Jian-Guang Lou, Ting Liu, and Dongmei Zhang: Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation. ArXiv:1905. 08205, 2019.

[15] Xi Victoria Lin, Richard Socher, and Caiming Xiong: Bridging textual and tabular data for crossdomain text-to-SQL semantic parsing. Findings of the Association for Computational Linguistics: EMNLP 2020(Online, November,2020), p4870–4888.

[16] Ben Bogin, Matt Gardner, and Jonathan Berant: Global reasoning over database structures for text-to-SQL parsing. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (Hong Kong, China, November,2019), p3657–3662.

[17] Zhi Chen, Lu Chen, Yanbin Zhao, Ruisheng Cao, Zihan Xu, Su Zhu, and Kai Yu: Shadowgnn: Graph projection neural network for text-to-sql parser. Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies(Online, June,2021), p 5567–5577.

[18] Ruichu Cai, Jinjie Yuan, Boyan Xu, and Zhifeng Hao: SADGA: Structure-aware dual graph aggregation network for text-to-sql. Advances in Neural Information Processing Systems (2021). Vol.34, p7664–7676.

[19] Binyuan Hui, Xiang Shi, Ruiying Geng, Binhua Li, Yongbin Li, Jian Sun, Xiaodan Zhu: Improving Text-to-SQL with Schema Dependency Learning. arXiv:2103.04399,2021.

[20] Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova: Compositional generalization and natural language variation: Can a semantic parsing approach handle both? In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the

ISSN: 2710-0588

11th International Joint Conference on Natural Language Processing (Online, August,2021). Vol.1, p922–938, Online.

[21] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau: PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (Online and Punta Cana, Dominican Republic.November,2021), p9895–9901.

[22] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson: RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (Online, July,2020), p7567-7578.

[23] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani: Self-Attention with Relative Position Representations. Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (New Orleans, Louisiana, June,2018). Vol.2, p464–468.

[24] Petar Velickovi ˇ c, Guillem Cucurull, Arantxa ´Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio: Graph Attention Networks.arXiv:1710.10903,2018.

[25] Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R. Woodward, Jinxia Xie, and Pengsheng Huang: Towards robustness of text-to-SQL models against synonym substitution. Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Online, August,2021). Vol.1,p2505-2515.

[26] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer: Learning a neural semantic parser from user feedback. Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Vancouver, Canada, July,2017). Vol .1, p963–973.

[27] John M. Zelle and Raymond J. Mooney: Learning to parse database queries using inductive logic programming. Proceedings of the Thirteenth National Conference on Artificial Intelligence (1996). Vol. 2, p1050–1055.

[28] Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning: Stanza: A Python Natural Language Processing Toolkit for Many Human Languages. Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations (Online, July,2020), p101–108.

[29] Binyuan Hui, Ruiying Geng, Lihan Wang, Bowen Qin, Bowen Li, Jian Sun, and Yongbin Li: S^2SQL: Injecting Syntax to Question-Schema Interaction Graph Encoder for Text-to-SQL Parsers.arXiv: 2203.06958,2022.

[30] Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu.: LGESQL: Line Graph Enhanced Text-to-SQL Model with Mixed Local and Non-Local Relations. ArXiv:2106.01093,2021.

[31] Jiexing Qi, Jingyao Tang, Ziwei He, Xiangpeng Wan, Yu Cheng, Chenghu Zhou, Xinbing Wang, Quanshi Zhang, and Zhouhan Lin: RASAT: Integrating Relational Structures into Pretrained Seq2Seq Model for Text-to-SQL. arXiv:2205.06983,2022.

[32] Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li: Graphix-t5: Mixing pretrained transformers with graph-aware layers for text-to-sql parsing. arXiv:2301.07507,2023.