applied sciences

MDPI

Article

Relation-Aware Graph Transformer for SQL-to-Text Generation

Da Ma, Xingyu Chen, Ruisheng Cao, Zhi Chen, Lu Chen and Kai Yu *

X-LANCE Lab, MoE Key Lab of Artificial Intelligence, AI Institute, Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China; mada123@sjtu.edu.cn (D.M.); galaxychen@sjtu.edu.cn (X.C.); 211314@sjtu.edu.cn (R.C.); zhenchi713@sjtu.edu.cn (Z.C.); chenlusz@sjtu.edu.cn (L.C.)

* Correspondence: kai.yu@sjtu.edu.cn

Abstract: Generating natural language descriptions for structured representation (e.g., a graph) is an important yet challenging task. In this work, we focus on SQL-to-text, a task that maps a SQL query into the corresponding natural language question. Previous work represents SQL as a sparse graph and utilizes a graph-to-sequence model to generate questions, where each node can only communicate with k-hop nodes. Such a model will degenerate when adapted to more complex SQL queries due to the inability to capture long-term and the lack of SQL-specific relations. To tackle this problem, we propose a relation-aware graph transformer (RGT) to consider both the SQL structure and various relations simultaneously. Specifically, an abstract SQL syntax tree is constructed for each SQL to provide the underlying relations. We also customized self-attention and cross-attention strategies to encode the relations in the SQL tree. Experiments on benchmarks *WikiSQL* and *Spider* demonstrate that our approach yields improvements over strong baselines.

Keywords: SQL-to-text; relation-aware graph transformer (RGT); abstract SQL syntax tree

Citation: Ma, D.; Chen, X.; Cao, R.; Chen, Z.; Chen, L.; Yu, K. Relation-Aware Graph Transformer for SQL-to-Text Generation. *Appl. Sci.* **2022**, *12*, 369. https://doi.org/10.3390/app12010369

Academic Editors: Julian Szymanski, Andrzej Sobecki, Higinio Mora, Doina Logofătu

Received: 10 November 2021
Accepted: 17 December 2021
Published: 31 December 2021

1. Introduction

SQL (Structured Query Language) is a vital tool to access databases. However, SQL is not easy to understand for the average person. SQL-to-text aims to convert a structured SQL program into a natural language description. It can help automatic SQL comment generation as well as build an interactive question answering system [1,2] for natural language interface to a relational database [3–5]. Besides, SQL-to-text is useful for searching SQL programs available on the Internet. Guo et al. [6] and Wu et al. [7] also demonstrated that SQL-to-text can assist the text-to-SQL task [8–11] by using SQL-to-text as data augmentation. In the real world, it can help people understand complex SQLs quickly by reading corresponding texts.

A naive idea is casting SQL-to-text as a Seq2Seq problem [12,13]. Taking the SQL sequence as input, a Seq2Seq model translates it to natural language. The main limitation is that when the SQL sequence becomes longer, the Seq2Seq model may fail to capture the dependency between complex conditions and operations. SQL is structural and can be converted into an abstract syntax tree, as Figure 1 illustrated. Generally, a tree is a special graph, so SQL-to-text can be modeled as a Graph-to-Sequence [14] task. Xu et al. [15] considers the intrinsic graph structure of a SQL query. They construct the SQL graph by representing each token in the SQL as a node in the graph, and concatenating different units (e.g., column names, operators, values) through SQL keyword nodes (e.g., SELECT, AND). By aggregating information from the *K*-hop neighbors through graph neural network (GNN, Scarselli et al. [16], 2008), each node obtains its contextualized embedding which will be accessed in the natural language decoding phase. Though simple and effective, it suffers from two main drawbacks: (1) poor generalization capability due to the sparsity of the constructed SQL graph, and (2) ignorance of relations between different node pairs, especially the relevance among column nodes.

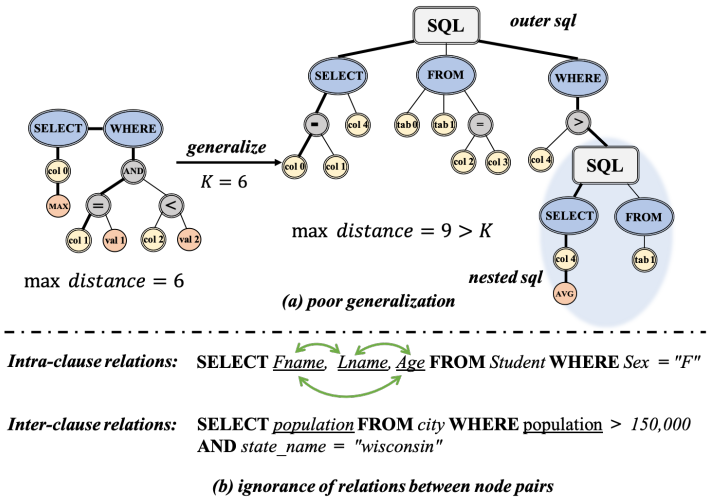

Figure 1. Two major problems incurred by the previous sparse Graph2Seq model. Examples are selected from dataset Spider. (**a**) Poor generalization problem. K is the iteration number. Bold edges indicate the maximum distance between node pairs in the entire graph. (**b**) Ignorance of relations between node pairs.

In particular, Xu et al. [15] only deals with the simple SQL sketch `SELECT $AGG $COLUMN WHERE $COLUMN $OP $VALUE (AND $COLUMN $OP $VALUE)*`. Only one column unit and one single table are mentioned in the sketch, and all constraints are organized via intersections of conditions in the WHERE clause. The model updates the contextualized embedding of each node by a K-step iteration. Each node will only communicate with its 1-hop neighbors in one iteration, thus each node can only "see" nodes within the distance of K at the end of iterations. The performance will easily deteriorate when we transfer to more complicated SQL sketches composed of multiple tables, `GroupBy/HAVING/OrderBy/LIMIT` clauses and nested SQLs.

As the example shown in Figure 1, a Graph2Seq model with $K = 6$ may work well on the simple SQL (shown in the left) while generalizing poorly on the complex SQL with a longer dependency distance (shown in the right). We find that two nodes may share high correlations even though they are far apart in both the serialized SQL query and the parsed abstract syntax tree. For instance, the columns mentioned in the same clause (intra-clause) are tightly related. See the example in Figure 1b. Users always require not only the last name, but also the first name of specific candidates. Similarly, there is a high probability that the column serving as one condition in the `WHERE` clause will also be requested exactly in `SELECT` clause (inter-clause). Previous work pays more attention on the syntactic structure of SQL, but neglects these potential relations at the semantic level.

To this end, we propose a Relation-aware Graph Transformer (RGT) to take into account both the abstract syntax tree of the query and the correlations between different node pairs. The entire node set is split into two parts: *intermediate nodes* and *leaf nodes*. Leaf nodes are usually raw table names or column words, plus some unary modifiers such as `DISTINCT` and `MAX`. Typically, these leaf nodes convey significant semantic information in the query. Intermediate nodes such as `SELECT` and `AND` inherently capture the tree structure of the underlying SQL query and connect the scattered leaf nodes. An example of constructed SQL tree is shown in Figure 2.

We introduce four types of relations into the SQL tree and propose two variants of cross-attention to capture the structural information. All relations are encoded by our proposed RGT model. As a SQL query may involve multiple tables, we first consider the relations among abstract concepts `TABLE` and `COLUMN`, called *databse schema* (DBS). Given two nodes representing `TABLE` or `COLUMN`, they might be two columns in the same table or two tables connected by a foreign key. We define 11 different types of DBS to describe such relations. Besides, the depth of node reflects the amount of information: deeper

nodes contain more semantic information while shallower nodes have more syntactic information. We introduce *directional relative depth* (DRD) to capture the relative depth between intermediate nodes. As for leaf nodes, the most important relation is affiliation. For example, in Figure 2, the leaf nodes *month* and *salary* are connected to the COLUMN node, and the COLUMN and another leaf node *val0* belong to the intermediate node >. These three leaf nodes are highly relevant. We use *lowest common ancestor* (LCA) to measure the closeness of two leaf nodes. As we can see, the LCA of node *month* and *val0* is the node > in Figure 2. Furthermore, to leverage the tree structure of SQL, we use two cross-attention strategies, namely *attention over ancestors* (AOA) and *attention over descendants* (AOD). *Attention over ancestors* only allows leaf nodes to attend their ancestors, and *attention over descendants* forces intermediate nodes to attend only their descendants.

We conduct extensive experiments on benchmarks WikiSQL [17] and Spider [18] with various baseline models. For simple SQL sketches on WikiSQL, our RGT model outperforms the previous best Graph2Seq model [15] and achieves 31.2 BLEU. To the best of our knowledge, we are the first to perform SQL-to-text task on the SQL sketches that involves multiple tables and complex conditions. Results (28.84 BLEU) demonstrate that our model generalizes well compared with other alternatives.

Our main contributions are summarized as follows:

- We propose a relation-aware graph transformer to consider various relations between node pairs in the SQL graph.
- We are the first to perform the SQL-to-text task with much more complicated SQL sketches on the dataset Spider.
- Extensive experiments show that our model is superior to various Seq2Seq and Graph2Seq models. Data and codes of our models and baselines will be public.

This paper is organized as follows: In Section 1, we introduce the task of SQL-to-text, analyze the existing problems of previous work and present our work on the whole. Then, we summarize related work in Section 2. After that, we clarify our method in detail in Section 3, including how to build the SQL tree and the architecture of our model. In Section 4, we conduct our experiments on two public datasets and report all the results. Finally, we conclude and show the expectation of future work in Section 5.

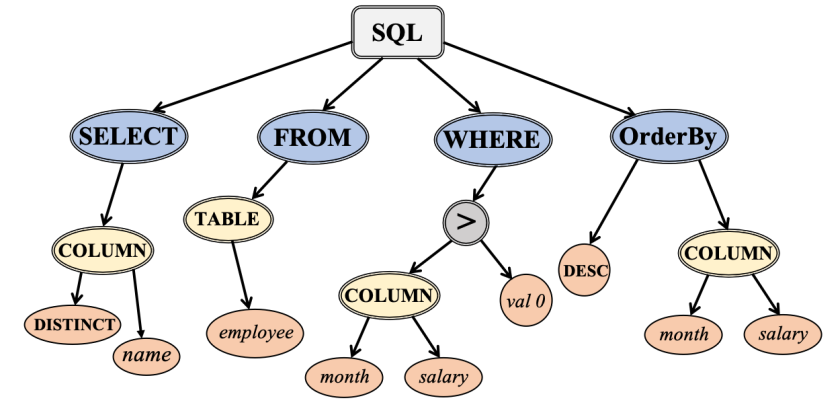

Figure 2. An example of the constructed SQL tree.

2. Related Work

Data-to-text Data-to-text intends to transform non-linguistic input data into the meaningful and coherent natural language text [19]. There are several types of the non-linguistic input data, such as a set of triples (the WebNLG challenge [20]) and some kinds of meaning representations (e.g., several slot-value pairs of the E2E dataset [21], the Abstract Meaning Representation (AMR) graph [22]). The key problem of this task is how to obtain a good

representation of the input data. At first, researchers [23,24] cast the structured input data to sequence and adopt the sequence-to-sequence model, e.g., LSTM. However, this method neglects the intrinsic structure of the input data. To this end, a lot of graph-to-sequence models are proposed. In particular, refs. [25,26] encoded the input data based on a graph convolutional network (GCN [27]) encoder. Ref. [28] extended transformer to the graph input and proposed the graph transformer encoder. In this work, our model is based on the graph transformer encoder.

SQL-to-text This technique can leverage automatically generated SQL programs [17] to create additional (question, SQL) pairs, alleviating the annotation scarcity problem of the complicated text-to-SQL [29] task with data augmentation [6]. Earlier rule-based methods [30,31] heavily rely on researchers to design generic templates, which will inevitably produce rigid and canonical questions. Seq2Seq [13], Tree2Seq [32] and Graph2Seq [15] models have demonstrated their superiority over the traditional rule-based system. In this work, we propose a relation-aware graph transformer to take into account both the graph structure and various relations embedded in different node pairs.

Tree-to-sequence Tree-to-sequence model [32] aims to map a tree structure into a sequence. Each node gathers information from its children nodes when encoding. They apply this technique to neural machine translation. Specifically, they reorganize the input sequence in the source language as a tree according to its constituency structure. In our work, we construct a SQL tree and utilize the Tree LSTM [33] as a baseline.

Graph-to-sequence Graph convolution network (GCN, Kipf and Welling [27], 2016) and graph attention network (GAT, Veličković et al. [34], 2017) have been successfully applied in various tasks to obtain node embeddings. Every node updates its node embedding by aggregating information from its neighbors. There may be labeled relations or features on edges of the graph. Relations or edge features can be incorporated when aggregating information from neighbors [2,35,36] or calculating relevance weights between node pairs [37–40]. We adopt both strategies with our tailored relations for different node pairs.

3. Model

3.1. SQL Tree Construction

The entire node set of the constructed SQL tree V is split into two categories: intermediate nodes $V^I = \{v_i^I\}_{i=1}^{|V^I|}$ and leaf nodes $V^L = \{v_i^L\}_{i=1}^{|V^L|}$. *Intermediate nodes* include three abstract concepts (`SQL`, `TABLE` and `COLUMN`), seven SQL-clause keywords (`SELECT`, `WHERE`, etc.) and binary operators (>, <, =, etc.), while *leaf nodes* contain unary operators, raw table names and column words and placeholders for entity value (entity mentions such as "new york" are replaced with one special token *val*0 during preprocessing, called *delexicalization*). With this partition, the node embeddings of these two types can be updated using different relational information.

Starting from the root node `SQL`, we firstly append the clause-level nodes as its children (see Figure 2). Then concept abstraction nodes, `TABLE` and `COLUMN`, and relevant operator nodes are accordingly attached to their parents. Next, for node `COLUMN` and `TABLE`, we append all the raw words, aggregators, and distinct flags as leaf nodes. Our SQL Tree consists of three levels (see Figure 3): clause level, schema level, and token level. Table 1 shows all types of nodes.

- First, SQL is divided into some clauses such as `SELECT` clause, `WHERE` clause, nested SQL clause and so on (see Figure 3a).
- Then, each clause is composed of several tables, columns, and some other binary operators. Considering that some table and column names have multiple tokens, we design two abstract nodes (`TABLE` and `COLUMN`) to address this problem (see Figure 3c). With these two abstract nodes, the clause nodes can be represented as shown in Figure 3b. Noticing that binary operators can be regarded as a relation between several nodes, we set them as intermediate nodes (parents of some children nodes).
- For other unary operators and tokens (table and column), we put them on leaves.

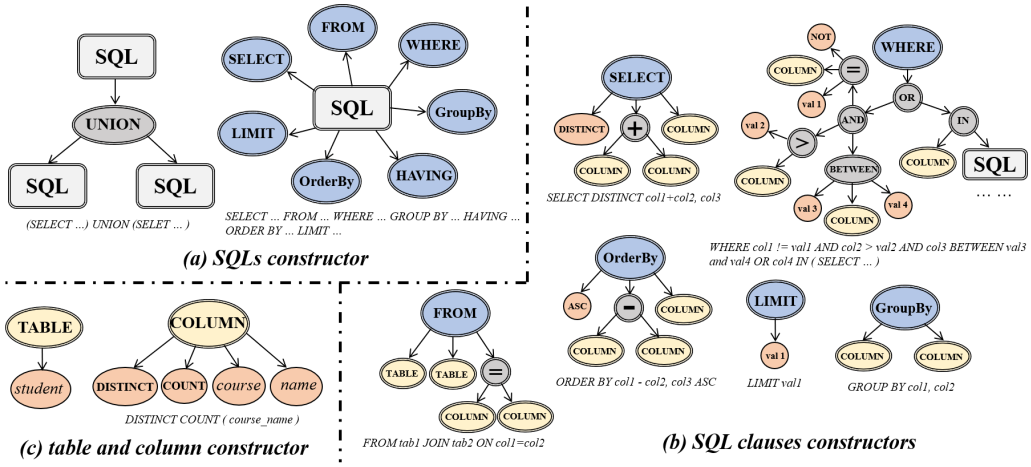

Figure 3. SQL Tree construction procedure. (**a**) is clause level; (**b**) is schema level; (**c**) is token level.

Table 1. Enumeration or examples of all types of nodes.

Node Types			Enumeration
intermediate	concept abstractions		SQL, TABLE, COLUMN
	clause keywords		`SELECT`, `FROM`, `WHERE`, `GROUPBY`, `HAVING`, `ORDERBY`, `LIMIT`
	binary operators	conjuction	`AND`, `OR`, `INTERSECT`, `UNION`, `EXCEPT`
		arithmetic	$+, -, \times, /$
		condition	$>, <, =, \geq, \leq$, `LIKE`, `IN`, `BETWEEN`
leaf	column/table words		e.g., column *state_name* $\longrightarrow$ nodes *state* and *name*
	value placeholders		*val*0, *val*1, *val*2, etc.
	unary operators	aggregation	`SUM`, `COUNT`, `MAX`, `MIN`, `AVG`
		others	`ASC`, `DESC`, `DISTINCT`, `NOT`

3.2. Encoder Overview

The input features include trainable embeddings for all nodes and relations. We use $\mathbf{X}^L \in \mathbb{R}^{|V^L| \times d_x}$ and $R^L = [r^L_{ij}]_{|V^L| \times |V^L|}$ to denote the set of leaf node embeddings and the relation matrix among leaf nodes. Symmetrically, $\mathbf{X}^I \in \mathbb{R}^{|V^I| \times d_x}$ and $R^I = [r^I_{ij}]_{|V^I| \times |V^I|}$ for intermediate nodes.

The encoder is composed of K stacked blocks, as illustrated in Figure 4. The main component is relation-aware graph transformer (RGT), which takes as input the node embedding matrix $\mathbf{X}$, the relation matrix $\mathbf{R}$ and a relation function $\mathbf{E}$ that extracts relation embeddings from $\mathbf{R}$, and outputs the updated node matrix. Each block contains four modules: one RGT for intermediate nodes, one RGT for leaf nodes, and two cross-attention modules. In each block, node embeddings $\mathbf{X}^I$ and $\mathbf{X}^L$ are updated sequentially via self-attention and cross-attention. According to the dataflow in Figure 4, intermediate nodes are first updated by

$$\mathbf{X}^I_{mid} = \mathrm{RGT}(\mathbf{X}^I_{in}, E^I_{rel}, R^I).$$

Then, leaf nodes attend intermediate nodes and update with RGT,

$$\mathbf{X}^L_{mid} = \mathrm{CrossAttention}^{L \leftarrow I}(\mathbf{X}^L_{in}, \mathbf{X}^I_{mid}),$$
$$\mathbf{X}^L_{out} = \mathrm{RGT}(\mathbf{X}^L_{mid}, E^L_{rel}, R^L).$$

Finally, intermediate nodes attend leaf nodes also,

$$\mathbf{X}^I_{out} = \text{CrossAttention}^{I \leftarrow L}(\mathbf{X}^I_{mid}, \mathbf{X}^L_{out}).$$

Subscripts in, mid, out are used to differentiate the inputs and outputs. Definitions of relation embedding functions E^I_{rel} and E^L_{rel}, relation matrix R^I and R^L, and module $\text{CrossAttention}^{I \leftarrow L}(\cdot, \cdot)$ and $\text{CrossAttention}^{L \leftarrow I}(\cdot, \cdot)$ will be elaborated later.

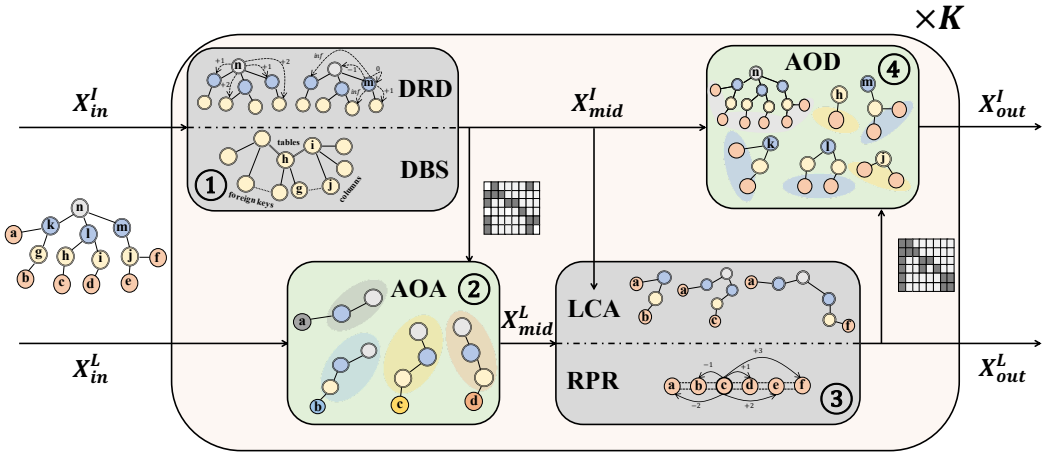

Figure 4. Architecture of the encoder. DRD: *directional relative depth*. DBS: *database schema*. LCA: *lowest common ancestor*. RPR: *relative position relation*. AOD: *attention over descendants*. AOA: *attention over ancestors*. The dataflow is ordered by the number.

3.3. Relation-Aware Graph Transformer

We utilize Transformer [41] as the backbone of our model, which can be viewed as an instance of graph attention network (GAT, Veličković et al. [34], 2017) where the receptive field for each node is the entire node set. We view SQL tree as a special graph. Assume the input graph is $G = (V, R)$, $V = \{v_i\}_{i=1}^{|V|}$, $R = [r_{ij}]_{|V| \times |V|}$, where V is the vertex set and R is the relation matrix. Each node $v_i \in V$ has a randomly initialized embedding $\mathbf{x}_i \in \mathbb{R}^{d_x}$. Shaw et al. [37] proposes to incorporate the *relative position* between nodes v_i and v_j into relevance score calculation and context aggregation step. Similarly, we adapt this technique to our framework by introducing additional relational vectors. Mathematically, given the relation matrix R, we construct a relation embedding function E_{rel} to retrieve the feature vector $\mathbf{e}_{ij} = E_{rel}(r_{ij}) \in \mathbb{R}^{d_x/H}$ for relation r_{ij}. Then, the output embedding $\mathbf{y}_i$ of node v_i after one iteration layer is calculated via

$$\begin{aligned}
\hat{a}^h_{ij} &= \frac{\mathbf{x}_i W^h_Q (\mathbf{x}_j W^h_K + \mathbf{e}_{ij})^T}{\sqrt{d_x/H}}, \quad a^h_{ij} = \text{softmax}_j\{\hat{a}^h_{ij}\}, \\
\mathbf{z}^h_i &= \sum_{j=1}^{n} a^h_{ij} (\mathbf{x}_j W^h_V), \quad \mathbf{z}_i = [\mathbf{z}^1_i; \cdots; \mathbf{z}^H_i] W_O, \\
\hat{\mathbf{y}}_i &= \text{LayerNorm}\{\mathbf{x}_i + \mathbf{z}_i\}, \\
\mathbf{y}_i &= \text{LayerNorm}\{\hat{\mathbf{y}}_i + \text{FC}(\text{ReLU}(\text{FC}(\hat{\mathbf{y}}_i)))\},
\end{aligned}$$

where $\text{FC}(\cdot)$ denotes a fully-connected layer, $\text{LayerNorm}\{\cdot\}$ is layer normalization trick [42], $[\,;\,]$ represents vector concatenation, parameters $W^h_Q, W^h_K, W^h_V \in \mathbb{R}^{d_x \times (d_x/H)}$, $W^h_O \in \mathbb{R}^{d_x \times d_x}$, and $1 \le h \le H$ is the multi-head index. The relation embedding function E_{rel} is shared across different heads and multiple layers unless otherwise specified. For the convenience of discussion, we simplify the notation of our RGT encoding module into

$$\mathbf{X}_{out} = \text{RGT}(\mathbf{X}_{in}, E_{rel}, R),$$

where $\mathbf{X}_{in} = [\mathbf{x}_1; \cdots; \mathbf{x}_{|V|}]$ represent the matrix of input embeddings for all nodes.

3.4. Relations among Intermediate Nodes

As for intermediate nodes, we consider two types of relations: *database schema* (DBS) and *directional relative depth* (DRD). DBS considers the relations among abstract concepts `TABLE` and `COLUMN`. In total, we define 11 relations, which is a subset of relations proposed in Wang et al. [39]. For example, if node v_i^I and v_j^I are nodes of type `COLUMN` and they belong to the same table according to the database schema, the relation r_{ij}^{DBS} is SAME-TABLE. Table 2 shows the complete version of DBS relations. Mathematically,

$$\begin{aligned} r_{ij}^{DBS} &= \text{DataBaseSchema}(v_i^I, v_j^I), \\ \mathbf{e}_{ij}^{DBS} &= E_{rel}^{DBS}(r_{ij}^{DBS}), \end{aligned}$$

where the relation embedding function E_{rel}^{DBS} maps the relation category r_{ij}^{DBS} into a trainable vector $\mathbf{e}_{ij}^{DBS}$.

Table 2. Database schema relation (refer to Wang et al. [39]).

Node v_i^I	Node v_j^I	Relation r_{ij}^{DBS}	Description
COLUMN	COLUMN	SAME-TABLE	v_i^I and v_j^I belong to the same table.
		FOREIGN-KEY-COL-F	v_i^I is a foreign key for v_j^I.
		FOREIGN-KEY-COL-R	v_j^I is a foreign key for v_i^I.
COLUMN	TABLE	PRIMARY-KEY-F	v_i^I is the primary key of v_j^I.
		BELONGS-TO-F	v_i^I is a column of v_j^I (but not the primary key).
TABLE	COLUMN	PRIMARY-KEY-R	v_j^I is the primary key of v_i^I.
		BELONGS-TO-R	v_j^I is a column of v_i^I (but not the primary key).
TABLE	TABLE	FOREIGN-KEY-TAB-F	Table v_i^I has a foreign key column in v_j^I.
		FOREIGN-KEY-TAB-R	Same as above, but v_i^I and v_j^I are reversed.
		FOREIGN-KEY-TAB-B	v_i^I and v_j^I have foreign keys in both directions.
others		NO-RELATION	v_i^I and v_j^I have no above relations.

With the assistance of the underlying directed SQL tree, we can build another relation matrix to indicate the accessibility and relative depth difference between two intermediate nodes v_i^I and v_j^I. Let $d(v_i^I)$ indicates the depth of node v_i^I, e.g., the depth of root SQL node is 1 (see Figure 4). Given the maximum depth difference D,

$$\begin{aligned} clamp(i,j) &= \max(-D, \min(d(v_j^I) - d(v_i^I), D)), \\ r_{ij}^{DRD} &= \begin{cases} clamp(i,j) & \text{if } v_i^I \to v_j^I \text{ or } v_j^I \to v_i^I \text{exists} \\ \text{inf} & \text{otherwise} \end{cases}, \\ \mathbf{e}_{ij}^{DRD} &= E_{rel}^{DRD}(r_{ij}^{DRD}), \end{aligned}$$

where E^{DRD} is the relation embedding module with $2D + 2$ entries. One special entry represents the inaccessibility inf.

The complete relation embedding function E_{rel}^I and relation matrix R^I for intermediate nodes merge *database schema* and *directional relative depth* together.

$$\begin{aligned} E_{rel}^I(r_{ij}^I) &= \text{FC}([\mathbf{e}_{ij}^{DBS}; \mathbf{e}_{ij}^{DRD}]) \in \mathbb{R}^{d_x/H}, \\ r_{ij}^I &= \text{tuple}(r_{ij}^{DBS}, r_{ij}^{DRD}), r_{ij}^I \in R^I, \end{aligned}$$

where affine transformation $\mathrm{FC}(\cdot)$ is used to fuse relation features from two perspectives and $\mathrm{tuple}(\cdot,\cdot)$ means the combination of relations.

3.5. Relations among Leaf Nodes

Leaf nodes mainly consist of raw words, plus a few unary operators as modifiers. Gathering all these nodes into a sequence s^L following their original order in the SQL query, we can obtain *relative position relation* (RPR) among these leaf nodes. Assume the position of node v_i^L in s^L is indexed by $s^L(v_i^L)$ and D is the pre-defined maximum distance, relation features $\mathbf{e}_{ij}^{RPR}$ for nodes v_i^L and v_j^L is defined as

$$\begin{aligned} r_{ij}^{RPR} &= \max(-D, \min(s^L(v_j^L) - s^L(v_i^L), D)), \\ \mathbf{e}_{ij}^{RPR} &= E_{rel}^{RPR}(r_{ij}^{RPR}). \end{aligned}$$

Actually, E_{rel}^{RPR} stores the parameter matrix of shape $(2D+1) \times (d_x/H)$ for retrieval. Tokens in the same clause will cluster together in the sequence s^L. Intuitively, r_{ij}^{RPR} with smaller absolute numerical value will capture the previously mentioned *intra-clause relations*.

Furthermore, we take into account the structure of SQL tree. Let $\mathrm{LCA}(v_i^L, v_j^L)$ denotes the *lowest common ancestor* for leaf nodes v_i^L and v_j^L in the SQL tree. The relation feature $\mathbf{e}_{ij}^{LCA}$ is computed via

$$\begin{aligned} r_{ij}^{LCA} &= \mathrm{LCA}(v_i^L, v_j^L), \\ \mathbf{e}_{ij}^{LCA} &= E_{rel}^{LCA}(r_{ij}^{LCA}). \end{aligned}$$

The relation embedding function E_{rel}^{LCA} simply extracts the current node embedding of intermediate node $\mathrm{LCA}(v_i^L, v_j^L)$ from $\mathbf{X}_{mid}^I$ and transforms it into dimension d_x/H through a trainable linear layer. The relation between remote leaf nodes is reflected by the common ancestor node.

The complete relation embedding function E_{rel}^L for leaf nodes is constructed by combining both the flattened and tree-structured relations

$$\begin{aligned} E_{rel}^L(r_{ij}^L) &= \mathrm{FC}([\mathbf{e}_{ij}^{RPR}; \mathbf{e}_{ij}^{LCA}]) \in \mathbb{R}^{d_x/H}, \\ r_{ij}^L &= \mathrm{tuple}(r_{ij}^{RPR}, r_{ij}^{LCA}), r_{ij}^L \in R^L. \end{aligned}$$

3.6. Cross-Attention between Leaf and Intermediate Nodes

Module $\mathrm{CrossAttention}^{I \leftarrow L}(\cdot,\cdot)$ collects features from leaf nodes V^L to intermediate nodes V^I, such that semantic information can flow into the structural node representations of V^I. For each intermediate node v_i^I, it calculates the attention vector $\mathbf{x}_i^{I \leftarrow L}$ over leaf nodes. Rather than attending all the leaf nodes (*attention over full* nodes, AOF), v_i^I only cares about its descendants determined by the SQL tree. We call this strategy *attention over descendants* (AOD). Let $V^{I \leftarrow L}(v_i^I)$ be the set of leaf nodes that are descendants of intermediate node v_i^I, the update equation for intermediate node v_i^I is

$$\begin{aligned} \hat{a}_{ij}^{I \leftarrow L} &= (\mathbf{x}_i^I W_e)(\mathbf{x}_j^L)^T, \quad v_j^L \in V^{I \leftarrow L}(v_i^I), \\ a_{ij}^{I \leftarrow L} &= \mathrm{softmax}_j(\hat{a}_{ij}^{I \leftarrow L}), \\ \mathbf{x}_i^{I \leftarrow L} &= \sum_j a_{ij}^{I \leftarrow L} \mathbf{x}_j^L, \\ \mathbf{x}_i^I &= \mathrm{LayerNorm}\{\mathbf{x}_i^I + \mathbf{x}_i^{I \leftarrow L}\}, \end{aligned}$$

where $W_e \in d_x \times d_x$ is trainable parameters.

Similarly, module $\mathrm{CrossAttention}^{L \leftarrow I}(\cdot,\cdot)$ collects features from intermediate nodes V^I to leaf nodes V^L, such that the semantic information can be organized referring to the

structural information. Rather than attending all the intermediate nodes, v_i^L only cares about its ancestors in the SQL tree. We call this strategy *attention over ancestors* (AOA), similar to AOD.

3.7. Decoder

After obtaining the final node embeddings $\mathbf{X}_{out}^I, \mathbf{X}_{out}^L$ of intermediate and leaf nodes, we apply an LSTM-based [43] sequential decoder with copy mechanism [44] to generating the natural language sentence. Representations of the raw table and column words in leaf nodes will be extracted for a direct copy before decoding. Placeholders for entities such as *val*0 will be replaced with corresponding nouns (called *lexicalization*) during post-processing. Specifically, we distinguish intermediate nodes from leaf nodes to capture the semantic and structural information differently. Given the final node embeddings $\mathbf{X}_{out}^I$, $\mathbf{X}_{out}^L$, the initial hidden state is

$$h_0 = \text{MaxPooling}(\mathbf{X}_{out}^I) + \text{MaxPooling}(\mathbf{X}_{out}^L),$$

where $\text{MaxPooling}(\cdot)$ is a function of transforming $\mathbf{X}^{n\times d}$ into $x^{d\times 1}$. In particular,

$$\begin{aligned} x &= \text{MaxPooling}(\mathbf{X}), \\ x_j &= \max_i \mathbf{X}_{i,j}. \end{aligned}$$

For each time step t, we get the context vectors c_t^I and c_t^L, respectively.

$$c_t^I = \text{Attention}(h_{t-1}, \mathbf{X}_{out}^I),$$

$$c_t^L = \text{Attention}(h_{t-1}, \mathbf{X}_{out}^L),$$

where $\text{Attention}(\cdot)$ is the same as the cross attention mentioned in Section 3.6.

Afterward, the concatenation of the context vectors and previous hidden state h_{t-1} is fed into the next step.

$$h_t = \text{LSTM}([h_{t-1}; c_t^I; c_t^L])$$

Considering there are many low-frequency words, we incorporate copy mechanism into the decoder. We use $P_{vocab}(y_t)$ and $P_{copy}(y_t)$ to denote the generation probability and copy probability of y_t, respectively. Let p_t^{gen} denote the probability of generating a word at time t. $P_{out}(y_t)$ is the final output probability of y_t. Then,

$$\begin{aligned} P_{vocab}(y_t) &= \text{softmax}(h_t W_{out}), \\ P_{copy}(y_t) &= \text{softmax}((h_t W_{copy})(X_{out}^L)^T), \\ p_t^{gen} &= \text{sigmoid}(h_t W_{gen}), \\ P_{out}(y_t) &= p_t^{gen} P_{vocab}(y_t) + (1 - p_t^{gen}) P_{copy}(y_t), \end{aligned}$$

where W_{out}, W_{copy}, and W_{gen} are trainable parameters.

4. Experiments

4.1. Dataset

WikiSQL We conduct experiments on WikiSQL with the latest version (The size of the latest version is 7019 less than used by [15]). SQLs in WikiSQL only contain `SELECT` and `WHERE` clauses with a short length. We utilize the official train/dev/test splits, ensuring each table only appears in a single split. This setup requires the model to generalize to unseen tables during inference.

Spider We also use Spider, a much more complex dataset. SQLs in Spider are much longer and the data size is much smaller compared to WikiSQL. Furthermore, some other complex grammars such as `JOIN`, `HAVING` and nested SQLs are also involved in Spider.

Thus, the task on Spider is much more difficult. Considering the test split is not public, we only use the train and dev splits.

The statistics of the two datasets are illustrated in Table 3.

Table 3. Statistics for WikiSQL and Spider.

Statistics	WikiSQL			Spider	
	Train	Dev	Test	Train	Dev
SQL number	56,355	8421	15,878	8658	1034
table number	18,585	2716	5230	795	81
database number	18,585	2716	5230	146	20
table number per database	1	1	1	5.45	4.05
average SQL length	8.67	8.70	8.71	29.57	25.93
average question length	11.64	11.73	11.69	12.05	12.36

4.2. Experiment Setup

Hyper parameters All our codes are implemented by Pytorch [45]. We utilize Adam [46] optimizer to train our models with a learning rate of 0.0001. The batch size is 32 for WikiSQL and 16 for Spider. Other hyperparameters can be found in Table 4. Note that the layer number of RGT of leaf nodes and intermediate nodes may not be identical. The update cycle (K) is equal to the minimum layer number. For example, RGT of intermediate nodes has 6 layers and 3 layers of leaf nodes. Thus, K is 3. Two RGT layers encode the intermediate nodes, and a single RGT layer encodes the leaf nodes for each cycle. The motivation is that the structure of SQLs in Spider are complex and vital. Thus, the intermediate nodes (structural part of SQL) require more layers to encode. To ensure fairness, all our embeddings (nodes and relations) are initialized randomly (the same as all baselines). We can also initialize all token embeddings (leaf nodes) with some pre-trained vectors (e.g., GloVe [47] and BERT [48]) to further boost the performance.

Table 4. Hyper parameters for our model on WikiSQL and Spider.

Parameters	WikiSQL	Spider
Embedding dimension		
leaf nodes	300	300
intermediate nodes	100	100
Relation		
maximum depth	4	4
maximum distance	4	4
RGT for intermediate nodes		
layer number	3	6
head number	8	8
hidden size	256	256
feed-forward hidden size	1024	1024
RGT for leaf nodes		
layer number	6	3
head number	8	8
hidden size	512	512
feed-forward hidden size	2048	2048
others		
epoch number	50	50
dropout	0.1	0.1
decoder hidden size	300	300

Metric We use BLEU-4 [49] and NIST [50] as automatic metrics. Each SQL has a single reference in WikiSQL. In Spider, most SQLs have double references because many SQLs

are corresponding to two different natural language expressions. However, there are two threats of this metric: (1) The results may fluctuate seriously. (2) BLUE-4 cannot fully evaluate the quality of the generated text. To alleviate the fluctuation of results, we run all our experiments 5 times with different random seeds. All results are obtained from the mteval-v14.pl(https://github.com/moses-smt/mosesdecoder/blob/master/scripts/generic/mteval-v14.pl, accessed on 9 November 2021) script. Furthermore, we conduct a human evaluation on Spider to compare our model with the strongest baseline.

Data preprocessing For WikiSQL, we omit the `FROM` clause since all SQLs are only related to a single table. For Spider, we replace the table alias with its original name and remove the `AS` grammar. Additionally, the questions are delexicalized as mentioned before.

4.3. Baselines

For all baselines, the same attention-based [51] LSTM decoder with a copy mechanism is utilized, where only the schema-dependent items (table and column tokens) will be copied.

BiLSTM The encoder is a BiLSTM encoder with SQL sequences as input. We report both results with and without a copy mechanism for this baseline.

TreeLSTM The encoder is a Child-Sum TreeLSTM encoder [33] with our SQL Tree as input.

Transformer We investigate the effect of position embedding on the transformer. Specifically, we consider transformer encoder without position embedding, with absolute position embedding [41] and relative position embedding [37].

GCN/GAT Regarding the SQL Tree as a graph, we can employ Graph Neural Networks (GNN), such as Graph Convolutional Network (GCN) and Graph Attention Network (GAT). Additionally, we rerun the code of Xu et al. [15] (https://github.com/IBM/SQL-to-Text, accessed on 10 November 2020).

4.4. Main Results

Table 5 shows the main results, including seq2seq baselines, graph2seq baselines and our model. Our model relation-aware graph transformer (RGT) outperforms all baseline models on both WikiSQL and Spider in both BLEU and NIST. Specifically, RGT outperforms the strongest baseline transformer with relative position by 1.17 BLEU and 0.23 NIST on Spider, 0.42 BLEU, and 0.1 NIST on WikiSQL, indicating the effectiveness of our model.

Table 5. Main results for all models. **Bold** means the best result.

Model	WikiSQL		Spider	
	BLEU (%)	**NIST**	**BLEU (%)**	**NIST**
BiLSTM without copy	29.59 ± 0.81	7.45 ± 0.17	22.22 ± 0.43	4.64 ± 0.06
BiLSTM	30.45 ± 0.17	7.75 ± 0.03	26.09 ± 0.70	5.55 ± 0.14
Transformer-w/o position	25.74 ± 0.32	7.08 ± 0.09	23.06 ± 0.75	5.07 ± 0.18
Transformer-absolute position	29.98 ± 0.30	7.70 ± 0.05	25.85 ± 0.50	5.61 ± 0.06
Transformer-relative position (REL)	30.78 ± 0.42	7.76 ± 0.05	27.67 ± 0.60	5.71 ± 0.07
TreeLSTM	29.18 ± 0.43	7.59 ± 0.09	24.15 ± 0.51	5.35 ± 0.10
Graph2Seq [15]	28.70	7.34	-	-
GCN	28.86 ± 0.17	7.51 ± 0.04	24.65 ± 0.38	5.26 ± 0.05
GAT	29.58 ± 0.81	7.63 ± 0.12	25.94 ± 1.08	5.50 ± 0.09
RGT (ours)	$\mathbf{31.20 \pm 0.23}$	$\mathbf{7.86 \pm 0.04}$	$\mathbf{28.84 \pm 0.22}$	$\mathbf{5.94 \pm 0.06}$

We discover that the GCN does not perform well compared to other baselines. GCN only cares about the structure of the graph without considering any special relations between nodes. We also notice that the transformer with a relative position works well even it only considers the relative position relation. This finding encourages us to consider more relations than the structure.

4.5. Ablation Study

To investigate the influence of relations and cross attention, we conduct two ablation studies, respectively. All our ablation studies are conducted on Spider.

Relation ablation In Table 6, pruning any relation leads to lower performance, indicating that all relations introduced to RGT are reasonable. Specifically, relations among leaf nodes seem more important, verifying the motivation of strengthening relations among semantic SQL tokens (column, table, and so on). We explain the effects of four relations as follows:

- structural relations: Both DBS (DataBase Schema) and DRD (Directional Relative Depth) strengthen the structural representation, but they work differently. DBS is to capture relations about the database schema, such as relations between table and table, table and column, and so on. DRD is to capture the hierarchical structure in SQL. For example (see Figure 2), both `DESC` node and `COLUMN` node (the most right two abstract nodes) are descendants of `OrderBy` node. To express the hierarchy, we incorporate direction into DRD.
- semantic relations: Both LCA (Lowest Common Ancestor) and RPR (Relative Position Relation) enhance the semantic representation. For instance (see Figure 2), the model can realize *month* and *salary* are close and may belong to the same column or table with RPR. With LCA, the model ensures they belong to the same column then.

Table 6. Relation ablation. The upper part is to investigate relations among intermediate nodes and the lower is among leaf nodes. **Bold** means the best result.

Model	BLEU (%)	NIST
RGT	**28.84 ± 0.22**	**5.94 ± 0.06**
w/o DBS	28.47 ± 0.38	5.94 ± 0.11
w/o DRD	28.56 ± 0.59	5.90 ± 0.10
w/o both	28.43 ± 0.52	5.89 ± 0.08
RGT	**28.84 ± 0.22**	**5.94 ± 0.06**
w/o RPR	27.52 ± 0.36	5.80 ± 0.09
w/o LCA	28.02 ± 1.05	5.85 ± 0.12
w/o both	26.72 ± 0.57	5.70 ± 0.12

Cross attention ablation To investigate how the cross attention mechanism affects the performance, we apply different combination of attention strategies in cross attention, namely attention over descendants (AOD), attention over ancestors (AOA), attention over full nodes (AOF) and no attention (None). Table 7 shows the experiment result.

Table 7. Cross attention ablation. The combination is in the format of **intermediate to leaf + leaf to intermediate** such as AOD + AOA. None symbol means no attention. For leaf nodes, AOF means attending all intermediate nodes. Similarly, AOF means attending all leaf nodes for intermediate nodes. **Bold** means the best result.

Cross Attention Combination	BLEU (%)	NIST
AOD + AOA	**28.84 ± 0.22**	**5.94 ± 0.06**
+ AOF	28.19 ± 0.37	5.82 ± 0.07
+ None	28.65 ± 0.41	5.89 ± 0.08
AOF + AOA	28.42 ± 0.66	5.83 ± 0.10
+ AOF	27.61 ± 0.91	5.77 ± 0.12
+ None	27.40 ± 0.61	5.76 ± 0.13
None + AOA	28.42 ± 0.54	5.86 ± 0.09
+ AOF	28.48 ± 0.65	5.87 ± 0.09
+ None	28.29 ± 0.39	5.82 ± 0.09

AOD + AOA works best, consistent with our expectations. We consider the cross attention is a balance problem. AOF can capture all kinds of relations, but may introduce

more noises (information from less related nodes), while None would lose some vital information. For example, AOD + None performs better than AOD + AOF, which means in this case AOF would introduce more noises. Besides, AOD + None outperforms None + None, indicating that ignoring all relations would lead to poorer performance. In this task, we choose AOD + AOA as our attention strategy, which can catch relations among different types of nodes without introducing too much noise.

4.6. Human Evaluation

We randomly select 100 samples (∼20%) from the dev set of Spider to conduct the human evaluation. For the SQL-to-text task, we should evaluate the correctness and fluency of the generation. To assess the correctness, we recruited two CS students familiar with SQL to score generations. They were first asked to select the better one for correctness from two generations. Furthermore, we asked them to objectively count the number of correct generation for aggregator (`MIN`, `MAX` and so on), column (column in SQL) and operator (+, −, `DESC`, `IN` and so on). Then, we calculated the metrics (precision, recall, and f1), respectively. Additionally, we asked three native English speakers to evaluate the fluency and grammar correctness. Our model is evaluated against the strongest baseline (transformer with relative position). The results are illustrated in Table 8. The lower part of Table 8 shows the percentage of choosing the generation as more correct (line *correctness*) or fluent (line *fluency*), and the percentage of a generation being chosen both correct and fluent (line *both*). From the evaluation result, we can conclude that our model can generate more correct sentences with a comparable fluency.

Table 8. Human evaluation for our model (RGT) and transformer with relative position (REL). **Bold** means the best result.

Aspect	REL			RGT		
	Precision	**Recall**	**F1**	**Precision**	**Recall**	**F1**
agg	0.72	0.71	0.71	**0.77**	**0.74**	**0.75**
col	0.67	0.51	0.56	**0.78**	**0.63**	**0.68**
op	0.56	0.49	0.51	**0.62**	**0.54**	**0.56**
correctness	34%			**66**%		
fluency	**53**%			47%		
both	37.25%			**62.75**%		

4.7. Case Study

We show two examples generated by our model RGT and the transformer with relative position (Figure 5). For the first example, both models can realize the *type* correctly, but the baseline fails to generate the *pet*. Our model can strengthen relations among tokens in one column, so the *pet* in the SQL would be a strong signal. For the second example, the baseline generates a more fluent sentence. There is a grammar error in the generation of our model (*teacher* is not correct), but *teacher* is matched with the SQL *teacher* in SQL. This phenomenon indicates that our model is more concerned with the relations among nodes. These two cases are consistent with our human evaluation conclusion.

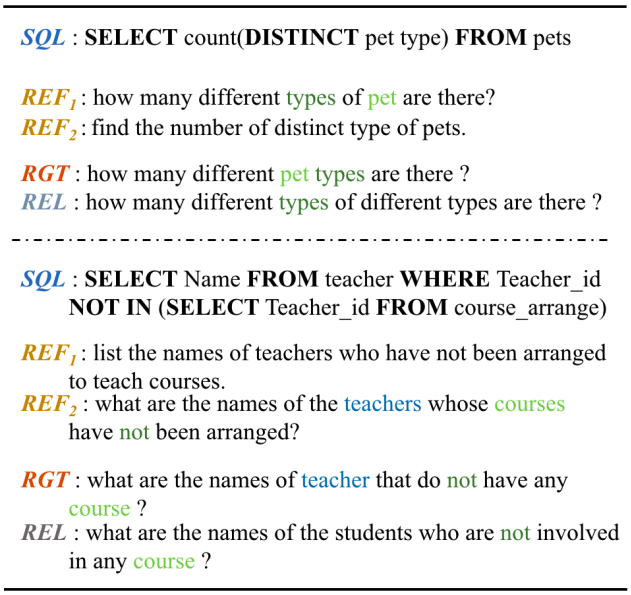

Figure 5. Case study: REF is the reference; REL is transformer with relative position; RGT is our model.

5. Conclusions

In this paper, we propose a relation-aware graph transformer (RGT) for complex SQL-to-Text generation. When learning the representation of each token in a SQL, multiple relations are considered in our model. Extensive experiments on two datasets *WikiSQL* and *Spider* show that our proposed model outperforms strong baselines including Seq2Seq models and Graph2Seq models.

There are two lines of work we can finish in the future. First, we can apply our SQL-to-text model to augment more text and SQL pairs to boost the performance of the text-to-SQL model by generating lots of SQLs automatically. In detail, we can make some SQL templates by handcrafting rules. Based on these templates, a lot of SQL queries can be generated, and then our SQL-to-text model transforms them into texts. These augmented text and SQL pairs can assist to train the text-to-SQL model. Second, we can extend our method to a more general task, e.g., code-to-text. Our model is appropriate to encode the abstract syntax tree of the programming language.

Author Contributions: Conceptualization, D.M. and L.C.; data curation, D.M.; formal analysis, Z.C.; supervision, Z.C., L.C. and K.Y.; writing—original draft, D.M., R.C. and X.C.; writing—review and editing, L.C. and K.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data supporting the conclusions of this article is avaliable at https://github.com/salesforce/WikiSQL and https://yale-lily.github.io/spider, accessed on 10 November 2020.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Liang, P. Learning executable semantic parsers for natural language understanding. *Commun. ACM* **2016**, *59*, 68–76. [CrossRef]
2. Sorokin, D.; Gurevych, I. Modeling semantics with gated graph neural networks for knowledge base question answering. *arXiv* **2018**, arXiv:1808.04126.
3. Livowsky, J.M. Natural Language Interface for Searching Database. U.S. Patent 6,598,039, 22 July 2003.
4. Shwartz, S.; Fratarcangeli, C.; Cullingford, R.E.; Aimi, G.S.; Strasburger, D.P. Database Retrieval System Having a Natural Language Interface. U.S. Patent 5,197,005, 23 March 1993.
5. Nihalani, N.; Silakari, S.; Motwani, M. Natural language interface for database: A brief review. *Int. J. Comput. Sci. Issues (IJCSI)* **2011**, *8*, 600.

6. Guo, D.; Sun, Y.; Tang, D.; Duan, N.; Yin, J.; Chi, H.; Cao, J.; Chen, P.; Zhou, M. Question Generation from SQL Queries Improves Neural Semantic Parsing. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, 31 October–4 November 2018; Association for Computational Linguistics: Stroudsburg, PA, USA, 2018; pp. 1597–1607.
7. Wu, K.; Wang, L.; Li, Z.; Zhang, A.; Xiao, X.; Wu, H.; Zhang, M.; Wang, H. Data Augmentation with Hierarchical SQL-to-Question Generation for Cross-domain Text-to-SQL Parsing. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, Online and Punta Cana, Dominican Republic, 7–11 November 2021; Association for Computational Linguistics: Stroudsburg, PA, USA, 2021; pp. 8974–8983.
8. Cao, R.; Chen, L.; Chen, Z.; Zhao, Y.; Zhu, S.; Yu, K. LGESQL: Line Graph Enhanced Text-to-SQL Model with Mixed Local and Non-Local Relations. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, Online, 2–4 August 2021; pp. 2541–2555.
9. Chen, Z.; Chen, L.; Li, H.; Cao, R.; Ma, D.; Wu, M.; Yu, K. Decoupled Dialogue Modeling and Semantic Parsing for Multi-Turn Text-to-SQL. *arXiv* **2021**, arXiv:2106.02282.
10. Cao, R.; Zhu, S.; Liu, C.; Li, J.; Yu, K. Semantic Parsing with Dual Learning. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Florence, Italy, 28 July–2 August 2019; Association for Computational Linguistics: Stroudsburg, PA, USA, 2019; pp. 51–64.
11. Chen, Z.; Chen, L.; Zhao, Y.; Cao, R.; Xu, Z.; Zhu, S.; Yu, K. ShadowGNN: Graph Projection Neural Network for Text-to-SQL Parser. In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Online, 6–11 June 2021; Association for Computational Linguistics: Stroudsburg, PA, USA, 2021; pp. 5567–5577.
12. Sutskever, I.; Vinyals, O.; Le, Q.V. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*; Curran Associates, Inc: Red Hook, NY, USA, 2014; pp. 3104–3112.
13. Iyer, S.; Konstas, I.; Cheung, A.; Zettlemoyer, L. Summarizing source code using a neural attention model. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, Berlin, Germany, 7–12 August 2016; pp. 2073–2083.
14. Xu, K.; Wu, L.; Wang, Z.; Feng, Y.; Witbrock, M.; Sheinin, V. Graph2seq: Graph to sequence learning with attention-based neural networks. *arXiv* **2018**, arXiv:1804.00823.
15. Xu, K.; Wu, L.; Wang, Z.; Feng, Y.; Sheinin, V. SQL-to-Text Generation with Graph-to-Sequence Model. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, 2–4 November 2018; Association for Computational Linguistics: Stroudsburg, PA, USA, 2018; pp. 931–936.
16. Scarselli, F.; Gori, M.; Tsoi, A.C.; Hagenbuchner, M.; Monfardini, G. The graph neural network model. *IEEE Trans. Neural Netw.* **2008**, *20*, 61–80. [CrossRef] [PubMed]
17. Zhong, V.; Xiong, C.; Socher, R. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv* **2017**, arXiv:1709.00103.
18. Yu, T.; Zhang, R.; Yang, K.; Yasunaga, M.; Wang, D.; Li, Z.; Ma, J.; Li, I.; Yao, Q.; Roman, S.; et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv* **2018**, arXiv:1809.08887.
19. Gatt, A.; Krahmer, E. Survey of the State of the Art in Natural Language Generation: Core tasks, applications and evaluation. *J. Artif. Intell. Res.* ***2018***, 61, *65–170.*
20. Gardent, C.; Shimorina, A.; Narayan, S.; Perez-Beltrachini, L. The WebNLG Challenge: Generating Text from RDF Data. In Proceedings of the 10th International Conference on Natural Language Generation, Santiago de Compostela, Spain, 4–7 September 2017; Association for Computational Linguistics: Stroudsburg, PA, USA, 2017; pp. 124–133.
21. Novikova, J.; Dušek, O.; Rieser, V. The E2E Dataset: New Challenges For End-to-End Generation. In Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue, Saarbrücken, Germany, 15–17 August 2017; Association for Computational Linguistics: Stroudsburg, PA, USA, 2017; pp. 201–206.
22. Banarescu, L.; Bonial, C.; Cai, S.; Georgescu, M.; Griffitt, K.; Hermjakob, U.; Knight, K.; Koehn, P.; Palmer, M.; Schneider, N. Abstract Meaning Representation for Sembanking. In Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse, Sofia, Bulgaria, 8–9 August 2013; Association for Computational Linguistics: Stroudsburg, PA, USA, 2013; pp. 178–186.
23. Konstas, I.; Iyer, S.; Yatskar, M.; Choi, Y.; Zettlemoyer, L. Neural AMR: Sequence-to-Sequence Models for Parsing and Generation. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Vancouver, BC, Canada, 30 July–4 August 2017; Association for Computational Linguistics: Stroudsburg, PA, USA, 2017; pp. 146–157.
24. Castro Ferreira, T.; Calixto, I.; Wubben, S.; Krahmer, E. Linguistic realisation as machine translation: Comparing different MT models for AMR-to-text generation. In Proceedings of the 10th International Conference on Natural Language Generation, Santiago de Compostela, Spain, 4–7 September 2017; Association for Computational Linguistics: Stroudsburg, PA, USA, 2017; pp. 1–10.
25. Marcheggiani, D.; Perez-Beltrachini, L. Deep Graph Convolutional Encoders for Structured Data to Text Generation. In Proceedings of the 11th International Conference on Natural Language Generation, Tilburg, The Netherlands, 5–8 November 2018; Association for Computational Linguistics: Stroudsburg, PA, USA, 2018; pp. 1–9.
26. Beck, D.; Haffari, G.; Cohn, T. Graph-to-Sequence Learning using Gated Graph Neural Networks. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Melbourne, Australia, 16–18 July 2018; Association for Computational Linguistics: Stroudsburg, PA, USA, 2018; pp. 273–283.

27. Kipf, T.N.; Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. In Proceedings of the 5th International Conference on Learning Representations, ICLR '17, Toulon, France, 24–26 April 2017.
28. Koncel-Kedziorski, R.; Bekal, D.; Luan, Y.; Lapata, M.; Hajishirzi, H. Text Generation from Knowledge Graphs with Graph Transformers. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Minneapolis, MN, USA, 3–5 June 2019; Association for Computational Linguistics: Stroudsburg, PA, USA, 2019; pp. 2284–2293.
29. Xu, X.; Liu, C.; Song, D. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv* **2017**, arXiv:1711.04436.
30. Koutrika, G.; Simitsis, A.; Ioannidis, Y.E. Explaining structured queries in natural language. In Proceedings of the 2010 IEEE 26th International Conference on Data Engineering (ICDE 2010), Long Beach, CA, USA, 1–6 March 2010; pp. 333–344.
31. Ngonga Ngomo, A.C.; Bühmann, L.; Unger, C.; Lehmann, J.; Gerber, D. Sorry, i don't speak SPARQL: Translating SPARQL queries into natural language. In Proceedings of the 22nd International Conference on World Wide Web, Riode Janeiro, Brazil, 13–17 May 2013; pp. 977–988.
32. Eriguchi, A.; Hashimoto, K.; Tsuruoka, Y. Tree-to-sequence attentional neural machine translation. *arXiv* **2016**, arXiv:1603.06075.
33. Tai, K.S.; Socher, R.; Manning, C.D. Improved semantic representations from tree-structured long short-term memory networks. *arXiv* **2015**, arXiv:1503.00075.
34. Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; Bengio, Y. Graph attention networks. *arXiv* **2017**, arXiv:1710.10903.
35. Schlichtkrull, M.; Kipf, T.N.; Bloem, P.; Van Den Berg, R.; Titov, I.; Welling, M. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*; Springer: Heraklion, Crete, Greece, 2018; pp. 593–607.
36. De Cao, N.; Aziz, W.; Titov, I. Question answering by reasoning across documents with graph convolutional networks. *arXiv* **2018**, arXiv:1808.09920.
37. Shaw, P.; Uszkoreit, J.; Vaswani, A. Self-attention with relative position representations. *arXiv* **2018**, arXiv:1803.02155.
38. Xiao, F.; Li, J.; Zhao, H.; Wang, R.; Chen, K. Lattice-based transformer encoder for neural machine translation. *arXiv* **2019**, arXiv:1906.01282.
39. Wang, B.; Shin, R.; Liu, X.; Polozov, O.; Richardson, M. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *arXiv* **2019**, arXiv:1911.04942.
40. Li, X.; Yan, H.; Qiu, X.; Huang, X. FLAT: Chinese NER Using Flat-Lattice Transformer. *arXiv* **2020**, arXiv:2004.11795.
41. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*; MIT Press: Long Beach, CA, USA, 2017; pp. 5998–6008.
42. Ba, J.L.; Kiros, J.R.; Hinton, G.E. Layer normalization. *arXiv* **2016**, arXiv:1607.06450.
43. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef]
44. See, A.; Liu, P.J.; Manning, C.D. Get to the point: Summarization with pointer-generator networks. *arXiv* **2017**, arXiv:1704.04368.
45. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*; Vancouver Convention Center: Vancouver, BC, Canada, 2019; pp. 8026–8037.
46. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
47. Pennington, J.; Socher, R.; Manning, C. GloVe: Global Vectors for Word Representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 26–28 October 2014; Association for Computational Linguistics: Stroudsburg, PA, USA, 2014 ; pp. 1532–1543.
48. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Minneapolis, MU, USA, 29–31 July 2019; Association for Computational Linguistics: Stroudsburg, PA, USA, 2019; pp. 4171–4186.
49. Papineni, K.; Roukos, S.; Ward, T.; Zhu, W.J. BLEU: A method for automatic evaluation of machine translation. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, Philadelphia, PA, USA, 8–10 July 2002; pp. 311–318.
50. Wołk, K.; Koržinek, D. Comparison and adaptation of automatic evaluation metrics for quality assessment of re-speaking. *arXiv* **2016**, arXiv:1601.02789.
51. Bahdanau, D.; Cho, K.; Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv* **2014**, arXiv:1409.0473.