applied sciences

MDPI

Article

Columns Occurrences Graph to Improve Column Prediction in Deep Learning Nlidb

Shanza Abbas [1], Muhammad Umair Khan [1], Scott Uk-Jin Lee [1,*] and Asad Abbas [2]

[1] Department of Computer Science and Engineering, Hanyang University, Ansan 15588, Korea; shanza92@hanyang.ac.kr (S.A.); mumairkhan@hanyang.ac.kr (M.U.K.)
[2] Faculty of Information Technology, University of Central Punjab, Lahore 54000, Pakistan; asadabbas.grw@ucp.edu.pk
* Correspondence: scottlee@hanynag.ac.kr

Abstract: Natural language interfaces to databases (NLIDB) has been a research topic for a decade. Significant data collections are available in the form of databases. To utilize them for research purposes, a system that can translate a natural language query into a structured one can make a huge difference. Efforts toward such systems have been made with pipelining methods for more than a decade. Natural language processing techniques integrated with data science methods are researched as pipelining NLIDB systems. With significant advancements in machine learning and natural language processing, NLIDB with deep learning has emerged as a new research trend in this area. Deep learning has shown potential for rapid growth and improvement in text-to-SQL tasks. In deep learning NLIDB, closing the semantic gap in predicting users' intended columns has arisen as one of the critical and fundamental problems in this research field. Contributions toward this issue have consisted of preprocessed feature inputs and encoding schema elements afore of and more impactful to the targeted model. Various significant work contributed towards this problem notwithstanding, this has been shown to be one of the critical issues for the task of developing NLIDB. Working towards closing the semantic gap between user intention and predicted columns, we present an approach for deep learning text-to-SQL tasks that includes previous columns' occurrences scores as an additional input feature. Overall exact match accuracy can also be improved by emphasizing the improvement of columns' prediction accuracy, which depends significantly on column prediction itself. For this purpose, we extract the query fragments from previous queries' data and obtain the columns' occurrences and co-occurrences scores. Column occurrences and co-occurrences scores are processed as input features for the encoder–decoder-based text to the SQL model. These scores contribute, as a factor, the probability of having already used columns and tables together in the query history. We experimented with our approach on the currently popular text-to-SQL dataset Spider. Spider is a complex data set containing multiple databases. This dataset includes query–question pairs along with schema information. We compared our exact match accuracy performance with a base model using their test and training data splits. It outperformed the base model's accuracy, and accuracy was further boosted in experiments with the pretrained language model BERT.

Keywords: deep learning; text-to-SQL; natural language processing; database; machine learning; machine translation

Citation: Abbas, S.; Khan, M.U.; Lee, S.U.-J.; Abbas, A. Columns Occurrences Graph to Improve Column Prediction in Deep Learning Nlidb. *Appl. Sci.* **2021**, *11*, 12116. https://doi.org/10.3390/app112412116

Academic Editors: Kamran Shaukat and Suhuai Luo

Received: 10 November 2021
Accepted: 7 December 2021
Published: 20 December 2021

1. Introduction

Recently, enormous databases have come to contain substantial knowledge about an organization because of the digital storage of data. These vast data repositories can contribute to research in data analysis and finding trends and patterns therein, according to any particular research goal. Increasingly, data repositories concerning medical health, movies and employee data require direct access by users according to their questions and queries. Traditionally, users have needed to learn structured query languages such as SQL to get precise results from relational databases. All experts of a particular domain,

for example, medicine, do not necessarily know structured query languages, limiting the access of organizational knowledge to a limited number of users. Therefore, existing data storage is not being utilized to its maximum potential.

Compulsory knowledge of structured query language for comprehensive access to all aspects of data has become a hurdle. Translating natural language questions into structured query language would provide maximum user access to relational databases. Asking questions in English-language text provides constraint-free access to databases, liberating the user from careful selection and click-based interfaces. Solving text-to-SQL tasks can expose the whole relational database for users to utilize and analyze according to their needs and choices. Seeking a solution for this issue leads us toward the field of natural language processing. However, natural language processing techniques, alone, cannot solve this problem, as the database, itself, is a critical part of the stated problem in this context, leading us to data-science methods as well.

Combining natural language processing and data science methods may yield the solution to building a natural language interface for databases. Thus, translating natural language queries into structured-language queries is a struggling area for merging natural language processing and data science. The goal is to translate natural language queries into SQL that can be executed on a system to access its data for all kinds of users. Translating natural language questions into programmed language has been a long-standing problem [1,2]. In this work, we have focused on a natural language interface to databases by the generation of executable SQL queries. Figure 1 elaborates the text-to-SQL task briefly.

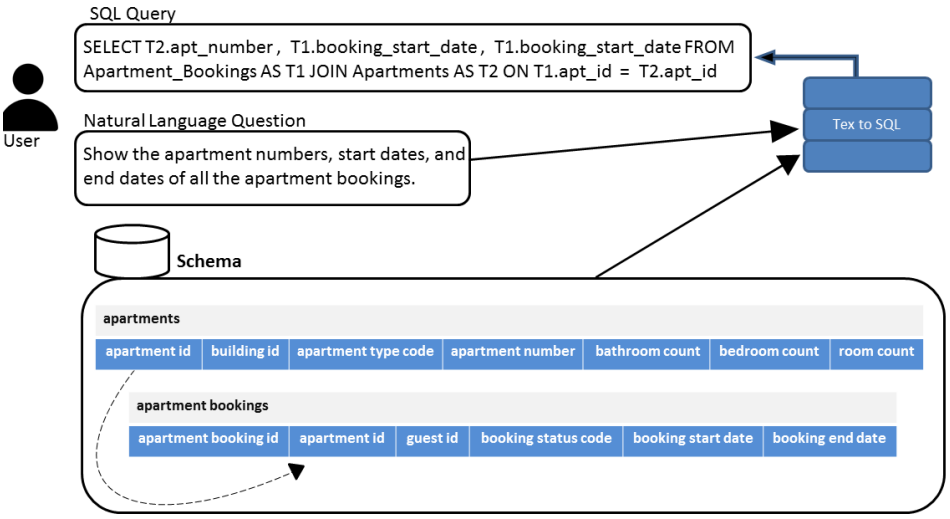

Figure 1. text-to-SQL task.

Deep learning has been an emerging tool in various research areas like communication, machine translation, and networking. Deep learning methods have arrived at competitive performances, compared with the traditional techniques, in a short time. They have been shown to be a potential tool for growth, even for mature fields with a higher bar of entry for new approaches, such as communication [3,4]. Deep learning has exhibited its problem-solving potential and growth in mobile traffic classification, as well. Though port information is not critical when working with deep learning traffic classifiers, mobile classifiers using deep learning methods can identify the application sources from which information is coming. Feature representations of direct-input data from training can be a potential path for improving traffic classifiers [5].

Deep learning has also shown promising results in the area of encrypted mobile traffic classification. However, deep learning, here, has some limitations because it is less mature in this area than are traditional methods, besides which, its black-box nature allows the least human intervention but contributes to resolving some of the complex matters in

improving performance [6,7]. Deep learning has recently been adopted for text-to-SQL tasks; keeping in mind deep learning's success stories in neighbouring research areas, deep learning shows potential for rapid improvement in this task as well.

The initial work in adapting deep learning to text-to-SQL tasks was primarily based on neural network sequence-to-sequence learning [8], adopting copying and attention mechanisms and the sequence-to-sequence RNN model to improve translation accuracy. These approaches improved the basic framework and their contributions were pioneering in NLIDB systems with deep learning concepts.

Recent work in the text-to-SQL research area mainly focuses on improving two significant aspects, syntactic accuracy and semantic accuracy. Semantic accuracy is to interpret the user's intention correctly and map it to a given database schema. The semantic gap between mapping user's preferences from natural language query into schema nomenclature is also known as a mismatch [9,10] or lexical problem [11,12]. Often, exact column names are not mentioned in the text query. Instead, one of the synonyms of column names are mentioned, or the intended column's value is mentioned. This scenario makes mapping the intended column to the actual column name in the database vague. Figure 2 elaborates the issue more precisely with an example. The natural language question in Figure 2 does not have the exact intended column name in the SQL query, which makes the column prediction vague in the given example.

```
Natural Language Question:
        What is the average number of rooms of studio apartments?
SQL Query:
        SELECT avg(room_count) FROM Apartments WHERE apt_type_code = "Studio"
```

Figure 2. Example of no column mention in NL Question.

Bridging the gap between user intentions and data can improve column predictions and, ultimately, impact overall accuracy [13]. Primarily, this issue of semantic gap occurs during the keyword mapping component of the whole process. Keyword mapping is part of the text-to-SQL translation process in which words from a natural language query are mapped to column names and values from the database schema. Formulating a query with the user's intended columns and values from the database is the fundamental semantic issue of this task.

This semantic issue has been an intricate part of the text-to-SQL task because NLIDB systems are intended for everyday users who do not necessarily have explicit knowledge of database schemas. Therefore, they cannot specify the schema items in their natural language questions precisely. Finding a pattern within previous user queries and extracting co-occurrences between columns and between columns and tables would resolve the issue of mapping the intended schema elements as closely as possible to the user's choice in a structured query. The calculated pattern of previous query histories can provide a concept of typical users' preferences regarding the database and columns in question. Therefore, in this work, we have focused on filling the semantic gap between database schemas and users' intentions with the help of patterns established from previous query data. In this work, we have captured such patterns in a co-occurrences graph of various columns. The graph is explained in detail in Section 4.1. Co-occurrences graph scores can be utilized in two main steps; (1) capturing the occurrences and co-occurrences of columns and tables in previous query data; and (2) integrating that data as feature input and other input vectors in a deep learning NLIDB model.

A similarity measure has been described to capture scores from the column co-occurrences graph effectively. We have performed experiments with a text-to-SQL task on the Spider dataset. The Spider dataset and its difficulty categorization and criteria are explained in Section 5. Our experiments show that our approach improves exact match accuracy. We have compared our results with a base model of SyntaxSQLNet and two

other contributions that have implemented preprocessing on input feature vectors and integrated with SyntaxSQLNet [14–16]. We show that our work improved accuracy up to 10% in an experiment with BERT embedding. The rest of the paper is constructed as shown in Figure 3 below. Table 1 in the following contains all the acronyms used in the article.

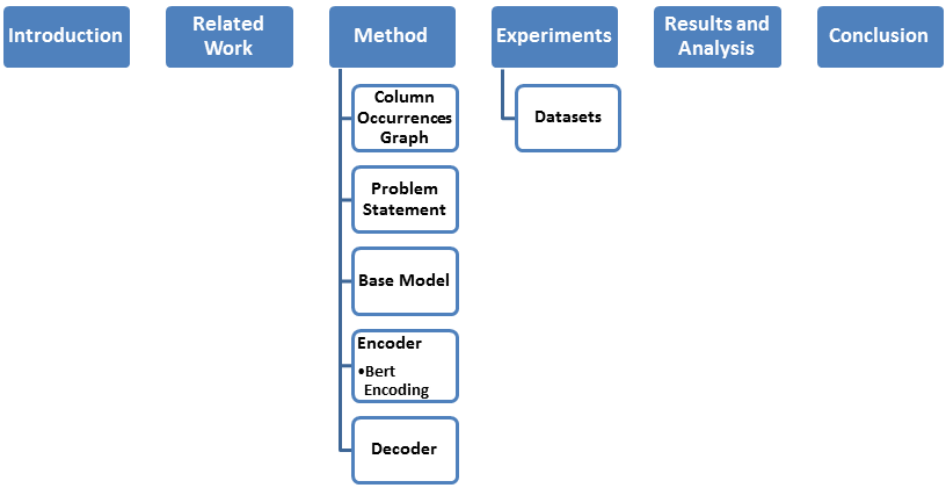

Figure 3. Paper Structure.

Table 1. Acronyms used in the paper.

Acronyms	Expansions
NLIDB	natural language interface to database
NL	natural language
NLP	natural language processing
DB	database
DL	deep learning
BERT	bidirectional encoder representations from transformation
GloVe embeddings	global vectors representation
SQL	structured query language

2. Background Study

2.1. Word Embedding

Word embedding is a vector representation of words learned in an architecture similar to neural networks. Similar words have similar and so closer representations in a predefined vector space. This vector representation is learned from a predefined, fixed-size library. The three most common techniques used for word embedding are an embedding layer, Word2Vec and GloVe. Word2Vec is a statistical technique for capturing the local meaning and context of a stand-alone text corpus [17,18].

On the other hand, gloVe extends the Word2Vec technique, combining its local context capturing and global matrix-factorization statistics. It uses statistics from the entire corpus of text to build a matrix of word co-occurrence. GloVe performs relatively better than Word2Vec.

2.2. Encoder–Decoder Structure

Encoder–decoder architectures for machine translation have been emerging since [19,20] used all the models based on an encoder that encoded a variable-length text sentence into a fixed-length vector. The decoder decoded the variable-length text from the same fixed-length vector to generate the targeted output translation. In the context of a text-to-SQL task, the LSTM encoder converts variable-length text sentences into a hidden state of the

encoder, which is a fixed-length vector. RNN layers are then stacked upon each other to build the final form of the encoder. These RNN layers structures contribute to capturing the context of the words and temporal dependencies in the sequence. The last step of the RNN acts as the hidden state from the encoder that is passed to the decoder [21]; this context vector encapsulates the whole meaning and the context of the sequence for the decoder to translate.

The decoder is the other half of the structure, which receives the hidden state from the encoder, encapsulating all the possible information of the sequence [19]. The decoder is another LSTM consisting of the stack of RNN layers along with the prediction layer. It converts the hidden state vector into an SQL query based on the information stored in the hidden state vector. The encoder–decoder architecture consists of two LSTMs that allow the variable-length input-output.

2.3. Attention Mechanism

The encoder–decoder architecture does not perform very well when sentences sequence grow longer [19] because of a built-in behavior of LSTM by which it can only remember that which that it has just seen. This makes it difficult for the encoder LSTM to encapsulate all the required information in one context vector, especially for longer sentences. The attention mechanism proposed by the [22] deals with these issues by structuring a method to embed all the words of a sequence in the context vector. For this purpose, they calculated a weighted sum of the hidden states to form a final hidden-state vector. The following equation has been proposed to calculate such weights:

$$C_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \quad (1)$$

3. Related Work

An NLIDB (natural language interface to database) system aims to cover the language hurdle between users and database systems. With NLIDB systems, non-technical users can also interact with the system without a knowledge of structured query languages. Therefore, NLIDB systems provide friendlier and greater access to relational databases for a broader range of users. Users query an NLIDB system in plain- or natural language text. The natural language question is then translated into a structured query language (SQL) query to be executed by the database engine and provide the user-intended results. Work by [1] is one of the early contributions to NLIDB [1], in which they presented an NLIDB task with brief examples describing problem statements and emphasized the importance of separating the linguistic task from the database information in their NLIDB task. To this end, syntax tree-based and semantic-based systems have been in the leading position, alongside intermediate representations of text and SQL languages. Until this point, the format of text query had been fixed, such as with fixed-syntax queries and menu-based systems, to limit associated semantic problems.

Subsequently, Ref. [2] proposed a method based on knowledge representation to separate the exact match tokens of natural language questions in the context to database elements. Moving further along the NLIDB research timeline, Ref. [23] introduced the tree kernels concept in NLIDB systems. They integrated tree kernels to rank candidate queries. Further work in this area has belonged to two categories; pipeline-method NLIDBs [24] and neural-network NLIDBs. Deep learning is used most in research on neural-network NLIDBs. Deep learning NLIDBs can be categorized further into sequence-to-sequence learning [8] and sequence-to-set learning. For our work, we have adopted sequence-to-set learning.

Sequence-to-set learning for text-to-SQL tasks was initially proposed to solve the 'order matters' issue [25]. The reward is calculated based on the whole ground-truth sequence compared to the gold-standard query for sequence-to-sequence structures. Sequence-to-set learning has been a fundamental approach for much work in NLIDB systems, in parallel

with sequence-to-sequence learning, as the order of the columns does not matter in the context of execution results [8]. Later, however, in the timeline of NLIDB systems, semantic accuracy became one of the significant issues in processing text-to-SQL tasks, as it has a substantial contribution in overall accuracy. Researchers have attempted to resolve this issue in various ways, such as with slot-filling approaches [26–28], which separate the syntactic and semantic issues by dealing with the former via building syntax grammatically before predicting the schema elements to populate slots. This allowed models to focus more on the semantic issue while predicting columns and tables. Then, Ref. [12] proposed global reasoning by a gating GCN to encode the DB schema for improving the prediction of its elements. Schema linking is another way to tackle mismatch issues, and was recently used in IRNet [10,29].

IRNet was extended to a sequence-based approach by [8]. They incorporated schema linking and intermediate representation into the baseline method. Schema linking, in IRNet, is done by simple string matching to identify the columns and tables mentioned in the question, labelling them with the column's information type. Although IRNet improved upon the baseline method, it is still not clear how schema linking impacts performance [10]. They separated the schema-linking task from the overall text-to-SQL task to analyze the impact of schema linking, showing that separating the schema entities from the questions and using placeholders instead allows the model to focus more on the syntactic component and improve overall query accuracy. Therefore, it is impactful on overall accuracy, but schema linking is not a perfect solution for semantic issues in text-to-SQL tasks. Additionally, using table content in the process has also improved prediction accuracy [30]. Despite this accuracy improvement, table contents may not be available in all cases, due to confidentiality. "Human in the loop" is another method of improving the output, in the context of capturing the user's intention, by taking feedback from them and revising it accordingly. DialSQL performs post-processing over the output of a prediction model. They take users' feedback in the form of multiple-choice questions. Various options related to defined error categories are provided for the user to select from. Then, taking the user's input into account, they improve the predicted query according to the user's provided information. This process requires the user to be trained and to explain the defined user categories beforehand. Ref. [31] advanced the "human in the loop" method by collecting user feedback in natural language and improving the interface's usability, as compared with DialSQL. With a "human in the loop", recruiting experts who know the database schema well enough to point to semantic errors is an additional effort compared with the other approaches. Both of these methods perform post-processing over generated output queries.

Various research contributions have proposed the preprocessing of input features to map DB schema elements to the user's question. Such work, by [14], implemented data anonymization over the DB schema and text queries before encoding the input features. Data anonymization consists of identifying the DB elements in the question via probability scores. Placeholders then replace the identified DB elements for the training phase. After anonymizing the text question from the DB elements, many training examples become similar and thus can share in the training process. Column/cell binding is the final step in generating probability distributions for the DB elements. The authors integrated this preprocessed anonymized data with an existing model, SyntaxSqlNet, for their experiments.

Another example of preprocessing feature vectors to impact a learning model is shown by [16]. They used column value polarities, generated beforehand, and integrated them with SyntaxSQLNet to better predict data elements. In the present work, we have followed a similar pattern of extending the baseline model by adding a preprocessed-input feature, i.e., column occurrences scores. Our contribution is unique in that we have utilized the data from previous queries to find patterns of users' intentions regarding the columns required of their questions. After a co-occurrences score is calculated it is integrated with SyntaxSQLNet as an additional feature vector. We have made the required changes to the

encoder to adjust our other features with respect to the model. Those variations and further details of the model are explained in the Methods section.

4. Methods

4.1. Column Occurrences Graph

First, we introduce the column occurrences graph. Following the idea of [32], the column occurrences graph is built from the query log or, in the Spider dataset case, from the previous SQL queries of a particular database in the training data. Example of such set of queries is shown in Figure 4. Figure 5 shows the columns occurrence frequency in the set of queries. Figure 6 shows the columns occurrences graph, where nodes represent the frequency of individual column occurrence and edges represent the co-occurrences of the comuns. The intuition of this graph is to capture the user's intention for clearer column prediction. Primarily in cases when the exact column name is not mentioned in the text question, prediction of one column can assist in predicting the other columns as well. Our method is an extended version of [32], in that we compute occurrences and co-occurrences of columns specifically, explicitly excluding the "from" part of queries to avoid noise and repetition. Instead, tables are concatenated with their column names, column types and relations, following the encoding method of [27]. Graph G contains edges, e, representing the column occurrences and vertices, v, representing co-occurrences of the involved schema elements in a particular database's query sets to capture this intuition. Following the idea of [32], the Dice similarity coefficient is used to reflect columns co-occurrences as follows:

$$Dice_{(C_1,C_2)} = \frac{2 \times n_e(C_1, C_2)}{n_v(C_1) + n_v(C_2)} \tag{2}$$

C_1 and C_2 are pairs of columns at a given time, and n_v and n_c are co-occurrences and occurrences of the respective column elements. Finally, the accumulate Dice coefficient for all column pairs is calculated with the following equation.

$$Score_{COG}(\phi) = [\prod_{(C_1,C_2)\in\phi T^2} Dice_{(C_1,C_2)}]^{\frac{1}{|\phi|}} \tag{3}$$

```
17x- SELECT apt_number, bedroom_count FROM Apartments
3x-SELECT max(room_count), max(bedroom_count) FROM Apartments WHERE T2.apt_type_code ="Flat"
4x-SELECT T2.apt_number, T1.booking_start_date FROM Apartment_Bookings AS T1 JOIN Apartments AS
    T2 ON T1.apt_id = T2.apt_id and T2.apt_type_code = "Flat"
```

Figure 4. Example of a set of queries.

21x: Apartments:apt_number	**3x**: Apartments:room_count
3x: Apartments:bedroom_count	**3x**: Apartments:building_id ?op ?var
4x: Apartment_Bookings:booking_start_date	**4x**: Apartments:apt_type_code ?op ?var

Figure 5. Columns Occurrences.

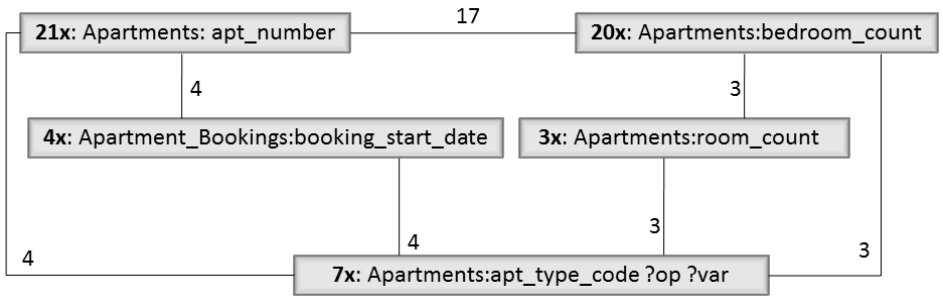

Figure 6. Column Occurrences Graph.

This section elaborates on our work and its implications for the issue of minimizing the ambiguities in the column prediction phase of the text-to-SQL task. First of all, we describe a problem statement for the semantic component of the text-to-SQL task. After that, the workings and components of the base model are explained to understand our extended work. The rest of the section addresses the implications of the column occurrences score in this work.

4.2. Problem Statement

Given a natural language text query Q and database schema primarily consisting of column names concatenated with table names and type information, labelled COL, our goal is to improve column prediction accuracy and ultimately enhance the model's overall accuracy. Utilizing previously used queries in a particular database can minimize the gap between the predicted columns and the user's intended output columns. Given that precalculated column-occurrences graph score, COG, along with Q and COL, PVALCOL is generated, ultimately generating the corresponding SQL, is the final goal. Text queries or natural language questions are treated as series of tokens to feed into the encoder–decoder model.

4.3. Base Model

Our base model, SyntaxSQLNet [27], is an encoder–decoder grammar-based slot-filling approach. The encoder encodes columns, the natural language question and history tokens as inputs, applying an attention mechanism to embed the most relevant question tokens in a columnular context. The weighted sum is calculated to bring the hidden state of the question token to the columns' attention. Decoders of the model predict the SQL syntax via a grammar to call modules based on history tokens. For semantic slot filling, the model has nine separate modules consisting of independent and respective biLSTM decoders. The column-predicting module is one of these nine modules and is trained separately.

$$P_{COL}^{num} = p(W_1^{num} {H_{Q/COL}^{num}}^T + W_2^{num} {H_{HS/COL}^{num}}^T) \quad (4)$$

The equation above reflects the column module's intuition. It is formulated in two parts; finding the total number of columns in the query, followed by the prediction of the column's values. The first equation computes the number of columns in the query, where $H_{Q/COL}$ is the hidden state of the question-to-column word embedding and $H_{HS/COL}$ represents the hidden state of the history of the last decoded element to the columns' attention mechanism. W_1 and W_2 are trainable parameters. Softmax was used for the probability distribution.

$$P_{COL}^{val} = p(W_1^{val} {H_{Q/COL}^{val}}^T + W_2^{val} {H_{HS/COL}^{val}}^T + W_3^{val} {H_{COL}}^T) \quad (5)$$

4.4. Encoder

The word embedding of question tokens, the schema, and history tokens is obtained from a pretrained GloVe [18] embedding. The current decoding history is further encoded with a biLSTM, denoted by HS [27]. We adopted the idea from [33] for schema encoding to capture self-attention of the schema elements. Schema encoding starts from obtaining the embedded column names, whererafter table names are embedded, as are column types. These initial embeddings are concatenated together. Self-attention is used between columns to capture the internal structure of the schema more effectively, where table names are also integrated. In the self-attention layer, another layer of biLSTM is applied to connect them and denoted as H_{COL}. The table schema encoding process is portrayed in Figure 7.

Following [27], after tokenization and GloVe embedding, the natural language question is encoded with biLSTM. To effectively capture the meaning and context of natural language questions fully with respect to the available column and tables in the database schema, an attention mechanism layer is applied, with the outputs of the hidden state of

the question tokens' biLSTMs and the columns' biLSTMs, generating the $H_{Q/COL}$. Figure 8 shows the NL question tokens encoding process.

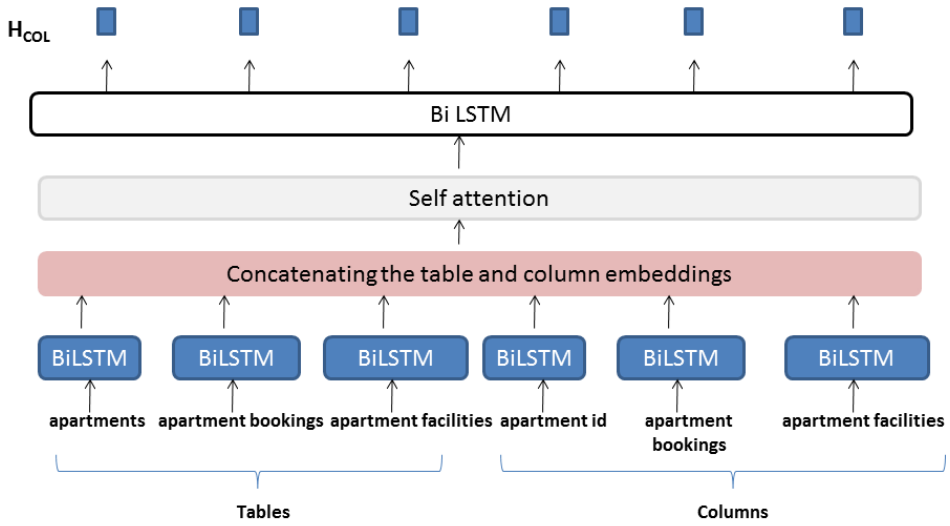

Figure 7. Table Schema Encoder.

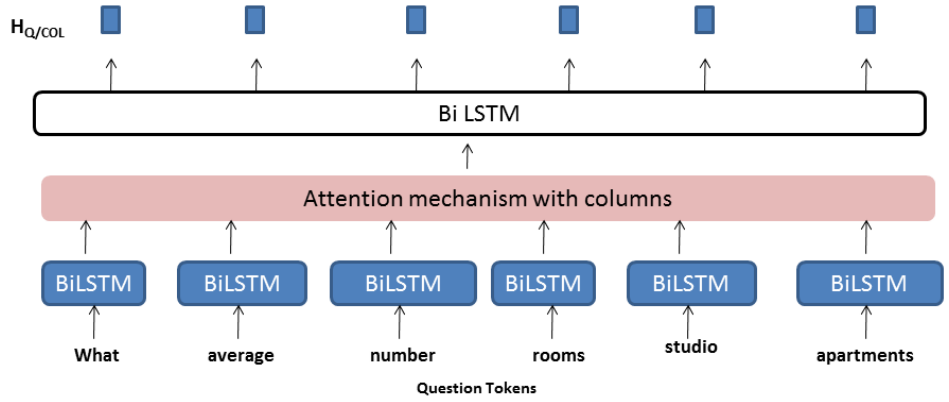

Figure 8. Question Encoder.

Columns occurrences scores have a graphical structure, with nodes as columns and edges as their co-occurrences. The embedded column names and occurrence scores are integrated. Along with other input features, column occurrence scores are also embedded via GloVe embedding, and then a biLSTM is applied. The hidden-state input of this biLSTM is further processed with the columns' hidden states to implement an attention mechanism between the question embedding and column occurrences embedding to find the relevant pairs of co-occurrences and their scores, denoted by $H_{COG/Q}$.

4.5. BERT Embedding of Input Features

Another option is to encode the input features using a pretrained BERT model. Question tokens and schema elements are fed into one sequence with a separator token. That sequence of input tokens is then provided to the BERT model. The final hidden state of the BERT is used as embedded input for the decoder. Previously, BERT has been shown to improve the overall accuracy of many models. We conducted one experiment with BERT, as well, to exhibit its compatibility.

4.6. Decoder

The decoder used in SyntaxSQLNet [27] is a recursively activating, sketch-based sequence-to-set approach. Nine separate decoder modules for each task are trained inde-

pendently from each other. Grammar is used to structure the syntax of a query and the calling of modules for each prediction. The decoder history path is encoded as part of the input to decide the next module with which to predict the next token. The sequence-to-set approach is used to avoid the 'order matters' problem caused in sequence-to-sequence approached [8], as identified by [27]. For example, "select id, name" is equivalent to "select name, id"; however, in the traditional sequence-to-sequence approach, the model penalizes these over even correct queries if the order of the sets is changed. The attention mechanism in [27] is generalized for all embeddings, as follows:

$$H_{\frac{1}{2}} = softmax(H_1 W {H_2}^T) H_1 \tag{6}$$

where W is a trainable parameter and H_1 and H_2 are the last hidden states of the biLSTM. Softmax was used for the calculation of probability distribution. The output of each module is computed in a sketch mode, independently of each other and whenever required, according to the syntax grammar [27].

The column module first computes the number of columns and then the values of the columns of the whole query altogether. We extended the column module by adding column occurrences scores as an input feature and using self-attention between columns instead of simple column embeddings, as follows:

$$P^{val}_{COL} = p(W^{val}_1 {H^{val}_{Q/COL}}^T + W^{val}_2 {H^{val}_{HS/COL}}^T + W^{val}_3 {H_{COL}}^T + W^{val}_4 {H^{val}_{COL/Q}}^T) \tag{7}$$

Here, W_1, W_2, W_3 and W_4 are trainable weights. Similarly, other modules are called according to the decoding history path and syntax grammar. For further details of the whole decoder and other modules, we refer the reader to the [25].

5. Experiments

This model was implemented in PyTorch [34]. Input questions and columns were tokenized using the Stanford toolkit [35] to process the sentence. A GloVe word-embedding vector was fed the one-hot vector output from the Stanford toolkit. A pretrained GloVe word-embedding model was used for all input features, such as columns, text questions, history tokens and COSs (column occurrence scores). All the word embeddings, here, were fixed-length vectors. Fixed-length vectors from the word-embedding model were then used as the input for the biLSTM model in the encoder. The dimension of the layers was 124, and the dropout rate was 0.2. An Adam optimizer was used to train the model.

Datasets

Despite the discussed works' successes, generalizing to new datasets was not an important factor in their models. Most of these models were built on traditional datasets, such as GeoQuery and ATIS. The task definitions of these datasets were comparatively simple and insufficient for practical use. The maximum complexity of these datasets was 500 SQL labels, which we expanded by paraphrasing approximately ten questions for each structured query. The test and training sets of queries contained overlapping queries, reducing the task's complexity. Another dataset used in this field is WikiSQL. It includes separate databases for training and testing purposes. Despite being a comparatively complex dataset, in terms of its larger size and multiple databases, it nonetheless has more straightforward queries. Such simple questions are inadequate to addressing the practical issues in semantic parsing. Yu et al. (2018b) have recently developed a complex dataset called the Spider dataset to cope with these issues. The Spider dataset contains around 6000 complex queries, along with 200 databases and multiple tables. This dataset defines the task of text-to-SQL with more complexity and can train cross-domain models.

Spider is the latest large-scale human-annotated dataset for text-to-SQL tasks [36]. It consists of 146 databases, along with their schemas, and 8659 query and question pairs. It has a training split of 752 queries and 1659 questions from previously established datasets, such as Scholar [37], IMDB & Yelp [38], GeoQuery [39], Restaurants [40], and Academic [41].

We used the Spider dataset primarily to evaluate our model with exact match accuracy. We used a split of query–question pairs of 130 for training, 36 for development, and 40 for the test, with a random distribution. Evaluation was performed according to the Spider evaluation script [36]. Beyond exact match accuracy, there are options for execution accuracy and logical form accuracy, as well. We chose exact match accuracy because execution accuracy can give false positives in some scenarios, wherein outputs may be similar, despite the schema columns from which they need to select being different [36]. Logical form accuracy could be more beneficial for syntactic improvement-oriented work. As this work focuses on the prediction of user-intended columns, exact match accuracy better represents our desired performance improvement.

This cross-domain and multitable dataset also introduced difficulty levels for model evaluation. Difficulty criteria contain a set of rules. The presence of particular rules decides the difficulty of the query from among easy, medium, hard and extra hard [36]. Figure 9 below shows the examples of easy, medium, complex and extra-hard queries.

Easy:
SELECT avg(room_count) FROM Apartments WHERE apt_type_code = "Studio"

Medium:
SELECT T1.booking_start_date FROM Apartment_Bookings AS T1 JOIN Apartments AS T2 ON T1.apt_id = T2.apt_id WHERE T2.apt_type_code = "Duplex"

Hard:
SELECT max(T2.bathroom_count), max(T2.bedroom_count) FROM Apartment_Bookings AS T1 JOIN Apartments AS T2 ON T1.apt_id = T2.apt_id WHERE T1.booking_status_code ="Provisional" and T2.apartment_type_code="Studio"

Extra Hard:
SELECT T2.apt_number FROM Apartment_Bookings AS T1 JOIN Apartments AS T2 ON T1.apt_id = T2.apt_id WHERE T1.booking_status_code = "Confirmed" INTERSECT SELECT T2.apt_number FROM Apartment_Bookings AS T1 JOIN Apartments AS T2 ON T1.apt_id = T2.apt_id WHERE T1.booking_status_code = " Confirmed"

Figure 9. Query Difficulty Examples.

Easy queries can contain only one keyword among "where", "join", "like", "having", "or", "limit", "order by" and "group by". Medium queries may contain two of these keywords. Queries categorized as "easy" do not have more than one entry in a select column, such as a "where" condition, aggregate function, or "order by" clause. Medium queries can have any two such clauses with more than one entry. Hard queries are those having at least three clauses with more than one entry, as shown in the example. The hard query in Figure 7 has two aggregate functions, two select column entries and two "where" conditions. Hard queries also contain two keywords (where, join, or, limit, order by, group by). All queries that do not fall within these three categorize are extra hard.

6. Results and Analysis

Although our decoder is similar to the base model SyntaxsqlNet, our columns occurrences score, in the encoder, allows the model to include user intentions regarding column prediction from the database. In addition, overall accuracy increased with our model's greater focus on column prediction. Our model achieved a prominent increase in exact match accuracy. Table 2 shows the experimental results in the form of exact match accuracy compared with the base model and two other methods from [14,16]. Our model's performance was higher than the base model in terms of exact match accuracy. Efforts by [14,16] are similar to our work in preprocessing the input features to impact overall model performance. Our integration of column occurrences scores outperformed the other two models, contributing to "understanding accurate intention regarding column predic-

tion from the natural language questions". Column prediction by column occurrences score enhanced the overall accuracy and outperformed the previous similar works [16] by, at our model's highest accuracy, 9.7%. Work by [16] extended the SyntaxSQLNet model by integrating the column value polarities as a feature vector [14], wherein they anonymized their input utterances to conceal lexical problems, minimizing semantic issues before data encoding. Column occurrences score and the self-attention mechanism between columns also contributed to the improved results.

We also experimented, on our model as well as the base model, with a pretrained language model, BERT. It improved the exact match accuracy even better for both models. Table 3 shows the partial matching accuracy, in terms of F1 scores, for "select", "where", "group by", "order by" and "keywords" separately. As shown in the table, accuracy improvement was less in the "group by", "order by" and "keywords" than the "select" and "where" clauses. The reason behind this is that group-by and order-by data are less represented in the training data. Therefore, there was less margin for improvement. Besides this observation, our overall exact match improvement shows our approach's potential for further work improvements in this area.

Table 2. Exact Match Accuracy Comparison.

Method	Easy	Medium	Hard	Extra Hard	All
SyntaxSQLNET	43.3%	22.8%	22.3%	4.2%	25.3%
DAE [14]	45.2%	30.5%	25.7%	7.9%	29.8%
SyntaxSQLNET + BERT	56.1%	31.7%	29.5%	8.9%	33.7%
Adjective-Noun Phrasing Knowledge [16]	67.5%	48.2%	41.7%	14.7%	45.4%
SyntaxSQLNet + COS (Ours)	84.2%	59.9%	59.5%	18.6%	55.1%
Ours + BERT	97.6%	70.2%	67.7%	24.8%	64.8%

Table 3. Clause-Wise F1 score accuracy.

Method	Select	Where	Group by	Order by	Keywords
SyntaxSQLNET	62.4%	34.1%	34.9%	56.8%	85.9%
DAE	59.8%	40.1%	38.4%	57.6%	91.2%
SyntaxSQLNET + BERT	71.0%	47.7%	38.5%	61.2%	89.3%
Adjective-Noun Phrasing Knowledge	70.1%	44.8%	63.2%	67.8%	76.8%
SyntaxSQLNet + COS (Ours)	71.5%	50.2%	65.0%	68.5%	77.9%
Ours + BERT	80.2%	61.9%	77.4%	80.0%	86.2%

7. Conclusions

In the field of NLIDB, bridging the gap between user-intended columns in a NL query and the predicted columns in the SQL query has been an ongoing issue. To resolve this semantic issue, various methods have been proposed, including preprocessing the data before its entrance to the model to reduce the semantic complexity. We have proposed an approach to include column occurrences scores extracted from previously executed user queries. We extended the base SyntaxSQLNet approach to emphasize column prediction accuracy with the help of a column occurrences scores graph. Column occurrences scores, in a previously executed queries database, shows the commonly used columns of that database by users with respect to the other columns, bridging the gap between predicted SQL query and user intended columns for a particular query. Our experiment shows that

bridging the column-prediction and user-intention gap has the potential to enhance the overall accuracy of NLIDB systems. We extended the column module in SyntaxSQLNet, adjusting its encoder–decoder accordingly. We predicted the columns that were nearest to the user's intentions by adding column occurrences scores as an additional input feature to the model. This study shows that predicting user-intended columns can enhance overall accuracy.

Author Contributions: S.A. formulated the theoratical formalism and investigation. M.U.K. and A.A. contributed to the development and experimenting. S.U.-J.L. supervised the process and contributed with the funding aquisition. All authors discussed and analyzed the results and formulated the final menuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data that support the findings of this study are openly available at github: https://yale-lily.github.io/spider, Reference number [36].

Conflicts of Interest: All authors declare that they have no conflict of interest.

References

1. Androutsopoulos, I.; Ritchie, G.D.; Thanisch, P. Natural Language Interfaces to Databases—An Introduction. *Nat. Lang. Eng.* **1995**, *1*, 29–81. [CrossRef]
2. Popescu, A.-M.; Etzioni, O.; Kautz, H. Towards a theory of natural language interfaces to databases. In Proceedings of the 8th International Conference on Intelligent User Interfaces, Miami, FL, USA, 12–15 January 2003; pp. 149–157.
3. Alam, T.M.; Mushtaq, M.; Shaukat, K.; Hameed, I.A.; Sarwar, M.U.; Luo, S. A Novel Method for Performance Measurement of Public Educational Institutions Using Machine Learning Models. *Appl. Sci.* **2021**, *11*, 9296. [CrossRef]
4. O'Shea, T.; Hoydis, J. An Introduction to Deep Learning for the Physical Layer. *IEEE Trans. Cogn. Commun. Netw.* **2017**, *3*, 563–575. [CrossRef]
5. Aceto, G.; Ciuonzo, D.; Montieri, A.; Pescapé, A. Mobile Encrypted Traffic Classification Using Deep Learning: Experimental Evaluation, Lessons Learned, and Challenges. *IEEE Trans. Netw. Serv. Manag.* **2019**, *16*, 445–458. [CrossRef]
6. Alam, T.M.; Shaukat, K.; Mahboob, H.; Sarwar, M.U.; Iqbal, F.; Nasir, A.; Luo, S. A Machine Learning Approach for Identification of Malignant Mesothelioma Etiological Factors in an Imbalanced Dataset. *Comput. J.* **2021**. [CrossRef]
7. Aceto, G.; Ciuonzo, D.; Montieri, A.; Pescapé, A. Toward effective mobile encrypted traffic classification through deep learning. *Neurocomputing* **2020**, *409*, 306–315. [CrossRef]
8. Zhong, V.; Xiong, C.; Socher, R. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv* **2017**, arXiv:1709.00103.
9. Naseem, U.; Khushi, M.; Khan, S.K.; Shaukat, K.; Moni, M.A. A comparative analysis of active learning for biomedical text mining. *Appl. Syst. Innov.* **2021**, *4*, 23. [CrossRef]
10. Guo, J.; Zhan, Z.; Gao, Y.; Xiao, Y.; Lou, J.G.; Liu, T.; Zhang, D. Towards complex text-to-sql in cross-domain database with intermediate representation. *arXiv* **2019**, arXiv:1905.08205.
11. Latif, M.Z.; Shaukat, K.; Luo, S.; Hameed, I.A.; Iqbal, F.; Alam, T.M. Risk factors identification of malignant mesothelioma: A data mining based approach. In Proceedings of the 2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE), Istanbul, Turkey, 12–13 June 2020; pp. 1–6.
12. Bogin, B.; Gardner, M.; Berant, J. Global reasoning over database structures for text-to-sql parsing. *arXiv* **2019**, arXiv:1908.11214.
13. Shaukat, K.; Luo, S.; Varadharajan, V.; Hameed, I.A.; Xu, M. A survey on machine learning techniques for cyber security in the last decade. *IEEE Access* **2020**, *8*, 222310–222354. [CrossRef]
14. Dong, Z.; Sun, S.; Liu, H.; Lou, J.G.; Zhang, D. Data-anonymous encoding for text-to-SQL generation. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing, Hong Kong, China, 3–7 November 2019; pp. 5405–5414.
15. Javed, U.; Shaukat, K.; Hameed, I.A.; Iqbal, F.; Alam, T.M.; Luo, S. A review of content-based and context-based recommendation systems. *Int. J. Emerg. Technol. Learn.* **2021**, *16*, 274–306. [CrossRef]
16. Liu, H.; Fang, L.; Liu, Q.; Chen, B.; Lou, J.G.; Li, Z. Leveraging adjective-noun phrasing knowledge for comparison relation prediction in text-to-sql. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing, Hong Kong, China, 3–7 November 2019; pp. 3515–3520.
17. Khushi, M.; Shaukat, K.; Alam, T.M.; Hameed, I.A.; Uddin, S.; Luo, S.; Reyes, M.C. A comparative performance analysis of data resampling methods on imbalance medical data. *IEEE Access* **2021**, *9*, 109960–109975. [CrossRef]
18. Pennington, J.; Socher, R.; Manning, C.D. Glove: Global vectors for word representation. In Proceedings of the EMNLP 2014: Conference on Empirical Methods in Natural Language Processing, Doha, Qatar, 25–29 October 2014; pp. 1532–1543.

19. Cho, K.; Merriënboer, B.V.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv* **2014**, arXiv:1406.1078.
20. Sutskever, I.; Vinyals, O.; Le, Q.V. Sequence to sequence learning with neural networks. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 8–13 December 2014; pp. 3104–3112.
21. Cho, K.; Merriënboer, B.V.; Bahdanau, D.; Bengio, Y. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv* **2014**, arXiv:1409.1259.
22. Bahdanau, D.; Cho, K.; Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv* **2014**, arXiv:1409.0473.
23. Giordani, A.; Moschitti, A. Translating Questions to SQL Queries with Generative Parsers Discriminatively Reranked. 2012. Available online: https://aclanthology.org/C12-2040.pdf (accessed on 10 November 2021).
24. Saha, D.; Floratou, A.; Sankaranarayanan, K.; Minhas, U.F.; Mittal, A.R.; Özcan, F. ATHENA: An ontology-driven system for natural language querying over relational data stores. *Proc. VLDB Endow.* **2016**, *9*, 1209–1220. [CrossRef]
25. Xu, X.; Liu, C.; Song, D. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv* **2017**, arXiv:1711.04436.
26. Lee, D. Clause-wise and recursive decoding for complex and cross-domain text-to-SQL generation. *arXiv* **2019**, arXiv:1904.08835.
27. Yu, T.; Yasunaga, M.; Yang, K.; Zhang, R.; Wang, D.; Li, Z.; Radev, D. Syntaxsqlnet: Syntax tree networks for complex and cross-domaintext-to-sql task. *arXiv* **2018**, arXiv:1810.05237.
28. Lin, K.; Bogin, B.; Neumann, M.; Berant, J.; Gardner, M. Grammar-based neural text-to-sql generation. *arXiv* **2019**, arXiv:1905.13326.
29. Lei, W.; Wang, W.; Ma, Z.; Gan, T.; Lu, W.; Kan, M.Y.; Chua, T.S. Re-Examining the Role of Schema Linking in Text-to-SQL. 2020. Available online: https://aclanthology.org/2020.emnlp-main.564.pdf (accessed on 10 November 2021).
30. Chen, Y.; Guo, X.; Wang, C.; Qiu, J.; Qi, G.; Wang, M.; Li, H. Leveraging Table Content for Zero-shot Text-to-SQL with Meta-Learning. In Proceedings of the AAAI Conference on Artificial Intelligence, Online, 2–9 February 2021; Volume 35, pp. 3992–4000.
31. Elgohary, A.; Hosseini, S.; Awadallah, A.H. Speak to your parser: Interactive text-to-SQL with natural language feedback. *arXiv* **2020**, arXiv:2005.02539.
32. Baik, C.; Jagadish, H.V.; Li, Y. Bridging the semantic gap with SQL query logs in natural language interfaces to databases. In Proceedings of the IEEE 35th International Conference on Data Engineering (ICDE), Macao, China, 8–11 April 2019; pp. 374–385.
33. Zhang, R.; Yu, T.; Er, H.Y.; Shim, S.; Xue, E.; Lin, X.V.; Radev, D. Editing-based SQL query generation for cross-domain context-dependent questions. *arXiv* **2019**, arXiv:1909.00786.
34. Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; Lerer, A. Automatic differentiation in pytorch. In Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA, 4–9 December 2017.
35. Manning, C.D.; Surdeanu, M.; Bauer, J.; Finkel, J.R.; Bethard, S.; McClosky, D. The Stanford CoreNLP natural language processing toolkit. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014), Baltimore, MD, USA, 23–24 June 2014; pp. 55–60.
36. Yu, T.; Zhang, R.; Yang, K.; Yasunaga, M.; Wang, D.; Li, Z.; Radev, D. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv* **2018**, arXiv:1809.08887.
37. Iyer, S.; Konstas, I.; Cheung, A.; Krishnamurthy, J.; Zettlemoyer, L. Learning a neural semantic parser from user feedback. *arXiv* **2017**, arXiv:1704.08760.
38. Yaghmazadeh, N.; Wang, Y.; Dillig, I.; Dillig, T. SQLizer: Query synthesis from natural language. In Proceedings of the ACM on Programming Languages, Paris, France, 15–21 January 2017; pp. 1–26.
39. Zelle, J.M.; Mooney, R.J. Learning to parse database queries using inductive logic programming. In Proceedings of the National Conference on Artificial Intelligence, Portland, OH, USA, 4–8 August 1996; pp. 1050–1055.
40. Tang, L.R.; Mooney, R.J. Using multiple clause constructors in inductive logic programming for semantic parsing. In *European Conference on Machine Learning*; Springer: Berlin/Heidelberg, Germany, 2001; pp. 466–477.
41. Li, F.; Jagadish, H.V. Constructing an interactive natural language interface for relational databases. *Proc. VLDB Endow.* **2014**, *8*, 73–84. [CrossRef]