Natural Language Interface for Databases Using a Dual-Encoder Model

Ionel Hosu[1], Radu Iacob[1], Florin Brad[2], Stefan Ruseti[1], Traian Rebedea[1]
[1]University Politehnica of Bucharest, Romania
[2]Bitdefender, Bucharest, Romania
{ionel.hosu, radu.iacob,stefan.ruseti,traian.rebedea}@cs.pub.ro
fbrad@bitdefender.com

Abstract

We propose a sketch-based two-step neural model for generating structured queries (SQL) based on a user's request in natural language. The sketch is obtained by using placeholders for specific entities in the SQL query, such as column names, table names, aliases and variables, in a process similar to semantic parsing. The first step is to apply a sequence-to-sequence (SEQ2SEQ) model to determine the most probable SQL sketch based on the request in natural language. Then, a second network designed as a dual-encoder SEQ2SEQ model using both the text query and the previously obtained sketch is employed to generate the final SQL query. Our approach shows improvements over previous approaches on two recent large datasets (WikiSQL and SENLIDB) suitable for data-driven solutions for natural language interfaces for databases.

1 Introduction

The quest for a simpler, more intuitive interface to query databases and other sources of structured information is one of the main practical applications of natural language processing. Developing a natural language interface for databases (NLIDB), able to process text queries directly, would be the optimal solution with regards to ease of use, requiring no additional knowledge and allowing non-technical persons to interact with the data directly. Nevertheless, even after half of decade of research, this is merely a desideratum, as no current solution is able to perform efficiently for complex databases.

From a computational linguistics perspective, natural language interfaces for databases could be treated as a special case of machine translation. Given a text in natural language to query a data source, a NLIDB system needs to output a statement in an artificial language, designed especially by computer scientists to query the database. Several such query languages exist, the most widely used being SQL (Structured Query Language) for structured databases and SPARQL for knowledge bases and ontologies.

Traditional NLIDB solutions have been using mainly a combination of rules and heuristics built upon syntactic dependencies and semantic parsing, with machine learning playing only a marginal role. The lack of large datasets suitable for training data-driven solutions is probably the main reason for the lack of complex machine learning approaches. However, a couple of large datasets (Brad et al., 2017; Zhong et al., 2017) have been recently proposed. These corpora allowed deep learning solutions to be deployed.

An additional obstacle that needs to be overcome by natural language interfaces for databases, especially by data-driven approaches, is the strong dependence of the corpora on the schema of a given database. Thus, even if one model is trained on a large dataset for a given database schema, it will not have good performance on different schemas. As it is impossible to develop a new dataset with pairs of text queries and corresponding SQL statements for each new database, the solution must be looked for in the underlying model. To this extent, we propose using a two-step approach which employs two different SEQ2SEQ models. The first one is schema agnostic and learns to output an SQL sketch (defined in Section 3) without any elements specific to a given database schema. The second model uses both the generated sketch and additional information about the database schema (information encoded by the model in its vocabulary) to generate the final SQL statement. We show that this model offers a significant improvement compared to a baseline SEQ2SEQ network with attention.

From a different standpoint, it is important to note that one of the biggest obstacles in generating code, including SQL, is that the output needs to be both syntactically and semantically valid. In previous research on code generation from natural language using neural models, this issue has been tackled by incorporating the syntax into a tree-based decoder (Yin and Neubig, 2017). To this end, our proposed two-step approach aims to learn separately the syntactic and semantic aspects of SQL generation without incorporating any human-defined grammar. The first step involves a SEQ2SEQ network trained on (text query, SQL sketch) pairs. The second step consists of a dual-encoder SEQ2SEQ architecture that generates SQL code based both on the user's request and the previously decoded sketch. Conditioning on the sketch can be seen as injecting syntactic information which helps the second network to focus more on the semantic aspects and on improving the syntax, given additional semantic elements. We show that this model offers a significant improvement compared to a baseline SEQ2SEQ model on two large datasets suitable for data-driven approaches for NLIDB.

The paper continues with an overview of existing solutions for NLIDBs, covering both incipient deep learning approaches and multi-stage conventional ones using a mix of rule-based and machine learning. In Section 3 we briefly present SENLIDB and WikiSQL, two large datasets of text queries and corresponding SQL statements which have already been used to train deep learning models for NLIDB. Section 4 presents the main contribution of our paper, introducing a two-step approach for generating SQL statements from textual input and query sketch using a novel dual-encoder neural model. Section 5 highlights the performance of the proposed two-step architecture compared to previous work. The paper ends with conclusions and future work.

2 Related Work

The first NLIDB solutions used dictionaries, grammars and dialogue to allow users to iteratively refine their query in natural language (Codd, 1974; Hendrix et al., 1978). For decades, complex multi-stage systems were designed for interacting in natural language with a database. They consisted of a mix of hand-crafted rules and heuristics, pattern matching, syntactic parsing, semantic grammar systems, and intermediate representation languages before generating the final statement in the query language (Androutsopoulos et al., 1995).

More recent conventional approaches combine syntactic parsing and semantic alignment to change the order of nodes in the parse trees in order to be correctly interpreted by a semantic analyzer (Popescu et al., 2004). To align the text with the generated SQL candidate statements, they propose using bipartite matching and dictionaries for semantic alignment. NaLIR (Li and Jagadish, 2014) also uses dependency parse trees and several hand-made rules and heuristics to generate candidate SQL statements. Given the input's dependency tree, the database schema and some manual semantic mappings, the system builds candidate query trees, as an intermediate step to SQL generation. The best query tree is computed using a scoring mechanism taking into account the similarity between the dependency and query trees and between adjacent nodes in the query tree, and an additional interaction with the user which is asked to select the best choice from a list of reformulations in natural language of top ranking candidate query trees.

Sqlizer (Yaghmazadeh et al., 2017) uses a mix of traditional rule-based and heuristics approaches combined with machine learning. Similar to our solution, a semantic parser is employed to generate a query sketch, which is then iteratively refined and repaired using rules and heuristics until the score of the generated SQL query cannot be improved. Sqlizer uses machine learning and Word2Vec (Mikolov et al., 2013) to determine the query sketch - a general form of the query, including clauses, but which does not contain any specific database schema information (e.g. table and column names). The reported results are surpassing previous solutions for several standard NLIDB datasets. While the underlying idea of Sqlizer (Yaghmazadeh et al., 2017) is similar to ours (using an intermediate sketch), there are significant differences. First, they do not employ a purely data-driven approach, as they use several hand-written heuristics. Secondly, the query sketch proposed by them is structurally different (encoding merely column and table names). Thirdly, the sketch is generated by a semantic parser instead of a neural model. However, the neural dual-encoder model employed in the second step of SQL generation is actually the most original part of our work, differentiating our paper from previous proposals employing SQL sketches. This model is novel and provides a substantial improvement over previous rule-based systems for filling the query sketch.

Deep learning approaches for code generation and semantic parsing have allowed the mapping from language to structured output to be learned directly, without the need for intermediate processing steps and hand-crafted rules and heuristics. Iyer et al. (2017) apply an interactive user-based feedback learning scheme to improve the results of a standard SEQ2SEQ model for generating SQL statements. Other approaches (Yin and Neubig, 2017; Rabinovich et al., 2017) incorporate the syntactic structure of the target structured language and decode the Abstract Syntax Tree (AST) corresponding to the surface code. Other recent solutions in semantic parsing avoid the need for ground truth logical forms inferring them automatically from (textual input, answer) pairs and also showing how this process can be generalized to multiple tables (Yin et al., 2015; Neelakantan et al., 2016).

Several large datasets have been recently introduced to facilitate training of deep learning approaches for NLIDB. Brad et al. (2017) propose the SENLIDB dataset together with a baseline SEQ2SEQ architecture for generating complex SQL statements. Zhong et al. (2017) introduce the WikiSQL dataset which contains simpler SQL-like queries involving a single table. They also suggest augmenting a standard SEQ2SEQ model with pointer networks (Vinyals et al., 2015) and highlight the fact that the order of specific clauses from the generated SQL statement affects the accuracy of a SEQ2SEQ model despite having no impact on the actual execution results. To mitigate this issue they apply reinforcement learning and reward queries that diverge from the ground truth but which return the correct results. More recently, Xu et al. (2017) report improved results on the same dataset without the use of reinforcement learning. Their solution, called SQLNet, employs a sketch-based approach designed to align well with the specific structure of SQL statements found in WikiSQL. Their SQL sketch conditions the prediction of a slot only on those particular values it may depend on, instead of all previous predictions. Our sketch-based approach is extended to support full SQL queries, while the two-step approach which uses both the sketch and the description in a dual-encoder to generate the SQL statement differentiates our model from SQLNet.

On another hand, dual-encoder SEQ2SEQ models have been successfully employed used both for machine translation and neural conversational models. In machine translation, dual-encoders have shown to improve the results when using multiple sources as input to generate the target translation (Zoph and Knight, 2016). In addition, dual-attentive decoders using features both from the source sentence and from an additional image related to it have been shown to improve multi-modal machine translation (Calixto et al., 2017). For neural conversational models, while Li et al. (2016) paved the road by introducing a persona-based decoder fed both from an encoder modelling the user input and from a persona specific embedding, only recently dual-encoders have been shown to produce better results than more complex hierarchical neural models (Lowe et al., 2017). Our approach uses a similar dual-encoder model, however we provide as its input two different representations for the desired SQL statement, the textual description supplied by the user and an automatically generated SQL sketch modelling the syntax of the final statement.

3 Two-Step Sketch-Based Model

We define a two-step solution for the problem of generating SQL statements from natural language queries[1]. First, we employ a network that learns to generate an SQL sketch given the user's request. Second, using both the aforementioned generated sketch and the corresponding natural language query, we propose a dual-encoder model to generate the final SQL statement. We consider that the SQL sketch generation is a simpler task, whose intermediate result provides important additional insights for determining the correct SQL statement.

Moreover, as the sketch preserves the SQL syntax of the final statement and only removes semantic

[1]Implementation available at `https://github.com/johnthebrave/nlidb-datasets`

information (e.g. names of tables, columns, constants and other), the proposed two-step model effectively splits the SQL generation problem into two distinct subproblems:

1. The problem of learning the syntax of the SQL query language and generating the most probable SQL sketch given a query in natural language. This refers to learning the surface structure of an SQL stamtement: the order of SQL reserved keywords in a (syntactically) correct query, the precedence of different clauses and subclauses in a the statement, the usage of different operators and so on.

2. The problem of generating semantically correct SQL statements for a given database schema, taking into account both the user's description and the most probable SQL sketch. This task mainly involves choosing the correct table and column names to fill in the sketch, generating the appropriate clauses (WHERE, GROUP BY etc.) given these names, but also correcting the sketch based on the additional semantic and schema information.

3.1 SQL Sketch Generation

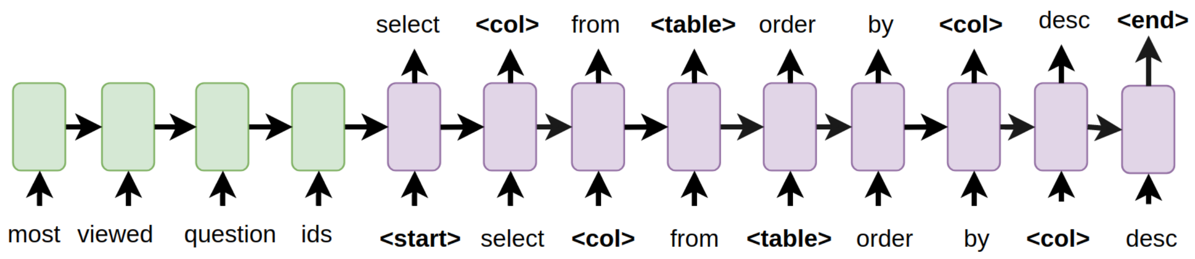

Figure 1: SQL sketch generation using a SEQ2SEQ network

In the first step, the model learns to generate an SQL sketch conditioned on the user's request. In order to obtain the sketches, we altered the dataset by replacing non-SQL keywords, reserved chars and operators within each SQL statement with placeholders (e.g. table names with <table>, similar notations for column names, aliases, variables, and constants). Thus, the SQL sketch only stores the surface form of an SQL statement and is syntactically correct, as we kept all mathematical and logical operators, SQL reserved keywords and special chars. All elements which are specific to the underlying database schema are removed from the sketch, therefore it is schema-agnostic.

To generate the SQL sketch, we have employed a standard SEQ2SEQ model with global-general attention (Luong et al., 2015) which we train on the (description, SQL sketch) pairs extracted from the dataset. An example of generating the SQL schema for a given user input is provided in Figure 1. Using placeholders for schema-related elements in the SQL sketch significantly reduces the output vocabulary for the decoder. This helps in two ways. First, it makes the model focus on the syntactic structure of the SQL statements. Second, it allows training the SEQ2SEQ network using less data - as the syntax is independent from the database schema, this data might even come from different datasets.

We used the open-source neural machine translation toolkit OpenNMT [2]. Both the encoder and the decoder are long short-term memory (LSTM) cells with two hidden layers and 500 neurons. The word embedding layer has 500 neurons. We used batches of maximum size 64. We trained the models with Stochastic Gradient Descent (SGD) for 25 epochs with a learning rate of 1.0 and a learning decay of 0.5 if perplexity did not decrease on the validation set. We generated the SQL sketch using a beam search of size 5.

3.2 Dual-Encoder for SQL Statement Generation

The second stage of the process consists of generating the full SQL statement using a dual-encoder SEQ2SEQ model, as shown in Figure 2. This model receives the natural language description of the query, as well as the SQL sketch obtained in the previous step, which are processed separately by each encoder. The last hidden states of each encoder are concatenated and fed to the decoder to generate the

[2]`https://github.com/OpenNMT/OpenNMT-tf`

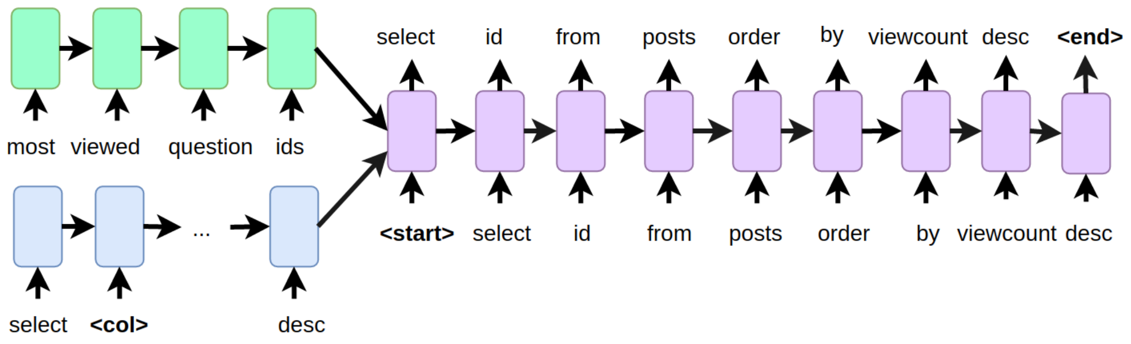

Figure 2: Query generation based on natural language and query sketch

final SQL statement. The decoder uses two attention mechanisms to attend both to the textual description and to the SQL sketch. By relying on the syntactical information provided by the sketch, the dual-decoder can focus more on schema specific entities (e.g. filling in the correct table and column names in the final statement) and on semantic aspects (e.g. generating the correct columns from a selected table). We also expect the dual-encoder model to be able to fix SQL sketches which contain some errors (when compared to the ground truth) after the first step as adding schema and semantic information might improve the syntax as well.

The encoders and the decoder share no weights with the first SEQ2SEQ network used for sketch generation. We used long short-term memory (LSTM) cells with two hidden layers and 500 neurons. The dual-encoder architecture was trained for 25 epochs using batches of 64 samples and a learning rate of 1.0 with learning decay of 0.5. We employed grid search for establishing the number of LSTM cells in the hidden layers and the size of the word embeddings (size 500). Additional hyperparameters that we used are dropout rate (0.3).

4 Results

We evaluated the proposed architecture on the SENLIDB and WikiSQL datasets. We compare our two-step dual-encoder model with a SEQ2SEQ baseline with global-general attention and unknown replacement (Luong et al., 2015). We also perform ablation tests by removing the attention mechanism, either on the textual description encoder or on the SQL sketch encoder, for the dual decoder that generates the final SQL statement.

Model	Dataset	BLEU	Accuracy
Dual-Encoder Seq2Seq (dual attention)	Dev	**83.89**	**39.50**
	Test	**83.2**	**38.99**
Dual-Encoder Seq2Seq (attention on description)	Dev	83.55	38.89
	Test	83.17	38.45
Dual-Encoder Seq2Seq (attention on SQL sketch)	Dev	83.86	38.54
	Test	83.21	38.47
Seq2Seq	Dev	78.90	32.60
	Test	78.50	32.00

Table 1: BLEU scores and accuracy for the generated SQL statements on the WikiSQL dataset

As it can be seen in Table 1 and Table 2, the proposed dual-encoder SEQ2SEQ model greatly outperforms the SEQ2SEQ baseline on both datasets. As expected, the BLEU and accuracy scores on the SENLIDB dataset are significantly smaller than on the WikiSQL dataset, due to the complexity of the SQL statements and database schema from the former dataset. In all experiments, accuracy is computed by checking whether the generated SQL statement matches the ground truth. This process does not take into account that different queries may actually provide the same results. This is especially true for the

Model	Dataset	BLEU	Accuracy
Dual-Encoder SEQ2SEQ	Validation	**21.70**	**3.52**
(dual attention)	Test-annotated	**22.92**	**3.97**
Dual-Encoder SEQ2SEQ	Validation	20.94	3.22
(attention on description)	Test-annotated	21.13	3.33
Dual-Encoder SEQ2SEQ	Validation	21.32	3.47
(attention on SQL sketch)	Test-annotated	21.80	3.60
SEQ2SEQ	Validation	16.90	2.80
	Test-annotated	18.20	2.44

Table 2: BLEU scores and accuracy for the generated SQL statements on SENLIDB dataset

more complex queries in the SENLIDB dataset (e.g. changing the order of the selected columns does not influence the results, but it influences the query match accuracy).

We also notice that removing any of the two attention mechanisms reduces the overall performance of the model on both datasets. This suggests that contextual information from both the SQL sketch and textual description is important to generate the final SQL statement.

Ultimately, all the dual-encoder models outperform the single encoder SEQ2SEQ model on the two datasets, which proves that the proposed two-step model benefits from the previously generated sketch. The added performance of our dual-encoder model comes from the initial task of SQL sketch generation. This demonstrates that even an incorrect, but approximate, syntactic surface form provides important additional information to the dual-encoder which corroborated with the textual description increases the performance of the model.

For the WikiSQL dataset, we also performed experiments with pretrained embeddings (GloVe) (Pennington et al., 2014) and a copying mechanism (See et al., 2017). Using these improvements, the proposed dual encoder model achieves an accuracy of 55.07%, outperforming most previous results.

4.1 Dataset Statistics

We have evaluated our proposed two-step neural architecture on two large datasets: WikiSQL (Zhong et al., 2017) and SENLIDB (Brad et al., 2017). Both datasets consist of several tens of thousand pairs of questions and corresponding SQL statements. However, they are different both with regard to the complexity of the database schemas and to the nature of the corresponding SQL statements. In this section, we briefly explain the characteristics of each dataset to highlight their different nature.

The content of the database tables used in WikiSQL (Zhong et al., 2017) was generated starting from $24,241$ Wikipedia tables. Schema-wise, the database is simplistic as all tables are independent from each other: there are no foreign keys or any other relationships between tables. The natural language queries and corresponding SQL statements were generated in several steps. First, SQL statements were randomly produced using a fixed template. Second, a corresponding interrogation in natural language was then generated automatically for each statement. Afterwards, human annotators from Amazon Mechanical Turk were employed to suggest several paraphrases for the automatically generated interrogations. Other human annotators were further employed to ensure the validity and quality of these paraphrases.

The fixed templates used to generate the SQL statements enforced them to have a specific, simple structure. Each statement contains a single $'select'$ clause requesting a unique column from only one table. An aggregation operator (e.g. $count()$) may be applied on the selected column. The only other component of the SQL statement is the $'where'$ clause, which also follows a very restricted form. Each corresponding subclause is actually a binary operation (e.g. equality operator, '$=$') applied on a column name and a substring which can always be found in the associated natural language query.

On the other hand, SENLIDB (Brad et al., 2017) consists of real-life SQL statements issued by users on the Data StackExchange Explorer website [3]. Unlike the ones from WikiSQL, these statements have no predefined syntactic limitations and are not generated automatically starting from a specific pattern.

[3] `https://data.stackexchange.com/`

Sample Statistics \ Datasets	SENLIDB		WikiSQL		
	Train	Test	Train	Dev	Test
# Samples	24890	780	61297	9145	17284
# Tables	29	15	18471	2704	5200
# Columns	204	98	50207	10144	17642
Avg. Text Length	7.88	10.44	11.73	11.81	11.78
Avg. SQL Length	71.38	7.46	11.50	11.48	11.55
Avg. SQL AST Height	26.11	20.27	19.62	19.63	19.63

Table 3: Brief comparison between SENLIDB and WikiSQL datasets

Moreover, the statements are issued against the StackExchange database which has a complex schema with several relationships and constraints between the various tables. Each SQL statement has a corresponding textual description added by the user who created it. This process resembles a crowd-sourced dataset, with each pair of (textual description, SQL statement) created by a user. To ensure the quality of the dataset, Brad et al. (2017) have preprocessed the data to remove incorrect or uninformative queries. Moreover, a small subset of queries (SENLIDB Test) has been manually annotated by developers with corresponding queries in natural language.

Table 3 provides a brief comparison between the two datasets. Although the size of the training data is similar for both datasets, the test set for SENLIDB is significantly smaller than the corresponding one for WikiSQL. This can be explained by the fact that this data has been manually annotated by developers. Related to the number of tables, although WikiSQL references a significantly larger number of tables than SENLIDB, there is no relation between them and each SQL statement only queries one table. In contrast, the SQL statements in SENLIDB may require data from several tables (thus requiring several $'join'$ operations) or use subqueries or even data extracted from temporary tables created within the statement. Finally, the Abstract Syntax Tree (AST) for the statements in SENLIDB are more complex not only to the number and type of SQL subclauses, but also to the height of the AST.

4.2 Qualitative results

input: what is terrence ross ’ nationality **generated sketch:** select <col>from table where <col> = <col> **ground truth sketch:** select <col> from table where <col> = <ct> **generated SQL:** select [nationality] from table where [player] = [terrence ross] **ground truth SQL:** select [nationality] from table where [player] = [terrence ross]
input: which podiums did the williams team have with a margin of defeat of 2 **generated sketch:** select <col> from table where <col> = <ct> **ground truth sketch:** select <col> from table where <col> = <ct> and <col> = <ct> **generated SQL:** select [podiums] from table where [margin of error] = [2] and [team] = [williams] **ground truth SQL:** select [podiums] from table where [team] = [williams] and [margin of defeat] = [2]

Table 4: Examples of SQL sketches and statements generated for the WikiSQL dataset

We list some decoded examples in Table 4 and Table 5. In the first example in Table 4, both the sketch and the final SQL statement are generated correctly by the dual-encoder model. The generated sketch in the second example is incorrect, but the final SQL statement is correct, which means the dual-encoder model actually rectified the surface syntactic form provided by the sketch. This shows that the dual-encoder not only adds specific schema elements and semantic information, but actually is able to modify some incorrect SQL sketches generated in the first step. This result is important as it means that additional semantic information is successfully used by the model to correct the proposed syntax.

In Table 5, we notice that for both examples, the generated sketch is very different from the ground

input: show top 50000 posts which have more than 10000 views
generated sketch: select top <ct> <alias> . <col> as <alias> , <alias> . <table> , <alias> . <col> , <alias> . <col> from <table> as <alias> inner join <table> as <alias> on <alias> . <col> = <alias> . <col> where <alias> . <col> = <ct> and <alias> . <col> > = <ct> order by <alias> . <col> desc
ground truth sketch: select top <ct> * from <table> where <table> . <col> > <ct> order by <table> . <col> desc
generated SQL: select top 10000 posts . id , posts . body from posts where posts . viewcount > 10000
ground truth SQL: select top 50000 * from posts where posts . viewcount > 10000 order by posts . viewcount desc

input: list id , body of posts that have the same postid as the id of posttags and tagid of posttags = 17 and id of posts < 1000
generated sketch: select <table> . <col> , <col> (*) as <alias> from <table> inner join <table> on <table> . <col> = <table> . <col> inner join <table> on <table> . <col> = <table> . <col> inner join <table> on <table> . <col> = <table> . <col> and <col> = <ct> group by <table> . <col> order by <col> desc
ground truth sketch: select <alias> . <col> , <alias> . <col> from <table> <alias> inner join <table> <alias> on <alias> . <col> = <alias> . <col> where <alias> . <col> = <ct> and <alias> . <col> < <ct>
generated SQL: select count (*) from posts inner join posttags on posttags . postid = posts . id
ground truth SQL: select p . id , p . body from posts p inner join posttags pt on p . id = pt . postid where pt . tagid = 17 and p . id < 1000

Table 5: Examples of SQL sketches and statements generated for the SENLIDB dataset

truth sketch. However, in the first example the generated SQL statement resembles the target SQL ($'top'$ clause, $'viewcount'$ column and $'posts'$ table are correctly identified). In the second example, the network correctly generates a more complex SQL query that contains a join operation between tables $'posts'$ and $'posttags'$. Again, the dissimilarity between sketches and similarity between final SQL statements suggests that the dual-encoder can recover from a incorrect generated sketch.

One of the reasons for which the SQL sketches on the SENLIDB dataset are more complex is related to the fact that the training data contains slightly more complex queries than the test set. This leads the sketch network to have a bias for more complex sketches.

4.3 Sketch accuracy

Dataset	BLEU	Accuracy
WikiSQL test	94.06	82.38
SENLIDB test	28.36	5.12

Table 6: BLEU scores and accuracy of the predicted SQL sketches

We have also investigated the performance of each of the two steps in our proposed model. In this subsection we measure the performance of the first step, the generation of the SQL sketch. In Table 6 we measure how many sketches are correctly predicted by the first SEQ2SEQ model. The BLEU score and accuracy of the generated sketches for the WikiSQL dataset are very high, as expected due to the much simpler nature of the SQL queries featured in this dataset. The accuracy of the final SQL statements is very low compared to the sketch accuracy, which means that the main difficulty with the WikiSQL dataset is correctly instantiating the SQL sketch with the appropriate columns names and constants thus resembling a slot filling task.

On the other hand, the sketch accuracy for SENLIDB is significantly lower compared to WikiSQL.

This is unsurprising, as the SQL queries are more complex. However, the difference between the sketch accuracy and the SQL accuracy for SENLIDB is small, which suggests that getting the right sketch is crucial for these queries and this step is actually more difficult than instantiating them.

4.4 Sketch feeding

We also evaluated the performance of our model when the correct sketch is being provided along with the textual description. To this extent, the dual-encoder in the second step receives the ground truth sketch at test time instead of the decoded sketch from the previous network.

Model	Dataset	BLEU	Accuracy
Dual-Encoder Seq2Seq	WikiSQL test	92.77	39.04
	SENLIDB test	68.20	25.89

Table 7: BLEU scores and accuracy of the generated SQL statements using the ground truth SQL sketch

We observe in Table 7 that the accuracy improves significantly on the SENLIDB corpus (21.92% improvement), which shows again the importance of generating the correct sketch for these difficult SQL statements. The accuracy boost on WikiSQL is small (1.15% improvement), reconfirming our earlier observation that WikiSQL is more challenging as a slot filling task.

4.5 Tables and Column Identification

Model	Task	Precision	Recall	F1 score
Dual-Encoder SEQ2SEQ	Table prediction	0.85	0.74	0.78
	Column prediction	0.59	0.53	0.55
SEQ2SEQ (Brad et al., 2017)	Table prediction	0.82	0.72	0.76
	Column prediction	0.55	0.47	0.50

Table 8: Results (using macro-averaging) for table and column prediction on the SENLIDB dataset

A simpler, but important subtask for the generation of SQL statements is the correct instantiation of table and column names in a query. In Table 8, we present different metrics for this task treated as a classification problem. As it can be observed, the dual-encoder model consistently outperforms the SEQ2SEQ baseline by merely using the SQL query sketch as input.

5 Conclusions

In this paper, we presented a two-step architecture for generating SQL statements starting from a user request expressed in natural language. The novelty resides in generating the underlying SQL sketch first, followed by the generation of the full SQL statement using a dual-encoder model conditioned by both the sketch and the query in natural language. The two-step approach actually separates the problem of generating an SQL statement into first constructing the correct syntactic surface form, which is later used to embed semantic and schema specific elements in the second step.

We demonstrated the validity of this approach by comparing the two-step model with a SEQ2SEQ baseline on two large datasets. The results show that the model significantly improves the results of the baseline. In addition, the best results are obtained when the dual-encoder model uses a dual attention mechanism on both textual description and SQL sketch. Future research should consider improving the two attention mechanisms and how they are combined. Other improvements can be obtained by training the sketch generation network on several datasets, considering that the sketch is schema-agnostic and might benefit from transfer learning between datasets.

A last insight of the paper relates to an important difference between the two datasets. As WikiSQL contains data from a simplistic database schema with artificial constraints for the SQL statements, it resembles a slot filling task. On the other hand, the main challenge for solving the more complex queries in SENLIDB is the generation of the correct SQL sketch corresponding to a query.

References

I. Androutsopoulos, G.D. Ritchie, and P. Thanisch. 1995. Natural language interfaces to databases – an introduction. *Natural Language Engineering*, 1(1):29–81.

Florin Brad, Radu Iacob, Ionel Hosu, and Traian Rebedea. 2017. Dataset for a neural natural language interface for databases (nnlidb). *arXiv preprint arXiv:1707.03172*.

Iacer Calixto, Qun Liu, and Nick Campbell. 2017. Doubly-attentive decoder for multi-modal neural machine translation. *arXiv preprint arXiv:1702.01287*.

E. F. Codd. 1974. Seven steps to rendezvous with the casual user. In *IFIP Working Conference Data Base Management*, pages 179–200, January. IBM Research Report RJ 1333, San Jose, California.

Gary G. Hendrix, Earl D. Sacerdoti, Daniel Sagalowicz, and Jonathan Slocum. 1978. Developing a natural language interface to complex data. *ACM Trans. Database Syst.*, 3(2):105–147, June.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback.

Fei Li and H. V. Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *VLDB*, 8(1):73–84, September.

Jiwei Li, Michel Galley, Chris Brockett, Georgios P Spithourakis, Jianfeng Gao, and Bill Dolan. 2016. A persona-based neural conversation model. *arXiv preprint arXiv:1603.06155*.

Ryan Thomas Lowe, Nissan Pow, Iulian Vlad Serban, Laurent Charlin, Chia-Wei Liu, and Joelle Pineau. 2017. Training end-to-end dialogue systems with the ubuntu dialogue corpus. *Dialogue & Discourse*, 8(1):31–65.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems*, NIPS13, pages 3111–3119, USA. Curran Associates Inc.

Arvind Neelakantan, Quoc V. Le, Martin Abadi, Andrew McCallum, and Dario Amodei. 2016. Learning a Natural Language Interface with Neural Programmer. pages 1–13.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. 2004. Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *Proceedings of the 20th International Conference on Computational Linguistics*, COLING '04, Stroudsburg, PA, USA. Association for Computational Linguistics.

Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract syntax networks for code generation and semantic parsing. *CoRR*, abs/1704.07535.

Abigail See, Peter J Liu, and Christopher D Manning. 2017. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700.

Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*.

Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. Type and content-driven synthesis of sql queries from natural language. *CoRR*, abs/1702.01168.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation.

Pengcheng Yin, Zhengdong Lu, Hang Li, and Ben Kao. 2015. Neural enquirer: Learning to query tables. *CoRR*, abs/1512.00965.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.

Barret Zoph and Kevin Knight. 2016. Multi-source neural translation. *arXiv preprint arXiv:1601.00710*.