Natural Language to Structured Query Language

June 29, 2021

By: Muhammad Usama Alvi (19K-0926)
Supervisor: Dr. Muhammad Rafi

National University of Computer and Emerging Sciences

Contents

List of Figures

List of Tables

1 Abstract

Relational databases are being used world wide to store massive amount of data in numerous fields. However, user faces issues while accessing data from these databases. Users need to understand structured query languages (SQL) in order to query on these databases and fetch results. An average user do not have this knowledge of these query languages. Here NL2SQL tasks play their role for providing an interface to fetch results. These tasks revolves around converting questions written in natural languages to structured query languages using deep learning approaches. This problem has received more attention recently because of large scale dataset releases such as WikiSQL. We have used RoBERTa model to achieve our task. We use RoBERTa model embeddings along with two knowledge vectors, namely, header knowledge and question knowledge vectors and pass it to LSTM based models to give us final results. We have improved both knowledge vectors which are passed to LSTM based models. The proposed method in the study has been evaluated on WikiSQL dataet. We use 6 LSTM based models for predicting individual parts of SQL. Significant improvements have been seen in each of the sub models which adds up to total accruacy. Model responsible for predicting select column has improved 1.0%, select agg 0.2%, where number 0.8%, where column 2.7%, where op 4.5%, where value 5.4%, which makes our test execution accuracy increased by 5.5%.

General Terms
Multi-label Classification, Natural language, Structured query language

Keyword and Index Terms
NL2SQL, SQL, NLP

2 Introduction

2.1 Background

This has been a long standing open problem to provide an interface on which a user, who has no idea about the syntax and working of a structured query language also known as SQL, can write queries in natural language and system converts that natural language to SQL. This SQL executes on database in order to fetch results. Many research has been done on this and still going on in order to bring up the accuracy. Although currently developed models are giving us a accuracy greater than 80%, there are some dependencies in the model which makes them not suitable to use in practical applications. We will be discussing them as well in related work.

Relational databases store a vast amount of data and are accessed through structured query languages. Many non technical persons are not familiar with these structured query languages, which creates a lot of trouble and dependencies. If we can remove this structured query language (SQL) dependency, then it would be a lot easier to fetch data from databases. This problem has been given a lot of attention after great number of large scale datasets has been made available [1] [2] [3].

Pre trained language models such as BERT [4] and RoBERTa [5] have gained a lot of success in the area of natural language processing tasks. [6] has done some amazing work using BERT on WikiSQL dataset. It has utilized the pre-trained model BERT as well as database table content in order to achieve state of the art model with highest accuracy. This comes with some limitations which we will discuss in related work.
Although there are number of models having significant accuracies on WikiSQL dataset, accuracies are still limited on Spider benchmark [3]. One of the reasons is that Spider dataset is more complex compared to WikiSQL dataset. WikiSQL focuses on rather simpler queries having single or multiple where clauses and no joins.

2.2 Motivation

Structured query languages has always been a bottleneck for researchers who are not familiar with the ways of structured query languages. This unfamiliarity creates an additional dependency for researchers in order to interact with databases for the dependency of data. If we are able to make a program, model, some kind of interface which lets researchers or any native users interact with databases without any prior knowledge of database, it will save a lot of time and resources. This has been a primarily source of motivation to decouple this dependency in order to save time and energy as well.

We explored state of the art existing models, their methodologies and how they were trying to get higher accuracy keeping in vision the privacy of the data. It is easier to achieve higher accuracy if vectors are trained using the actual data from database tables. However, that compromises the privacy of the data and that becomes the issue in real world applications. We explored in depth the understanding of models and how knowledge vectors are being created and passed to the models. We helped model gain more knowledge from vectors by incorporating more meaningful data into those vectors.

3 Related Work

In this section, we will review the past contributions that has brought natural language to structured query language task closer to real world solutions. Tabular form representation is shown at table 1.

3.1 Problems with Previous Approaches

Problem of converting plain text into a structured query language (SQL) is a problem of semantic parsing. It has been wildly in the focus of database communities. In the beginning, focus was more on rule based techniques [7], but after the release of large scale databases [3] [1], more approaches has come forward. Recent approaches are now NLP based. Some of the broader categories are - sequence to tree, sequence to sequence and SQL sketch. These approaches comes with their pros and cons. One of the issues of Seq2Seq discussed in [8] is that it faces the order matters problem. In order to convert natural language query to structured query language, we use sequence-to-sequence style model. In order to use this type of model, we have to serialize SQL queries. Here is the problematic part. Since there can be multiple correct queries, it creates different serializations for each query which is not good. Example shown in figure 1. There are some approaches which helps overcome this issue. One of the solution for this type of issue is to rely on reinforcement learning to reward the decoder when it comes with any of the correct serializations [8].

```
SELECT result                         SELECT result
WHERE score='1-0' AND goal=16         WHERE goal=16 AND score='1-0'
```

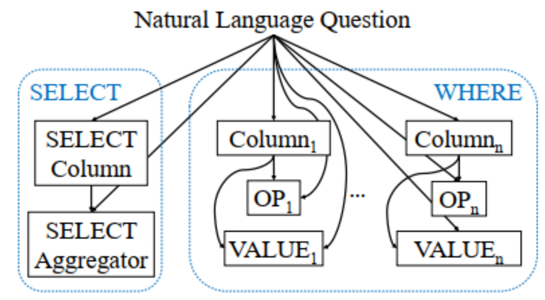
Figure 1: Order Matters

3.2 SQLNet

SQLNet [8] works differently from Seq2Seq method. It creates the sketch of important features and then these features are predicted individually as shown in figure 2. It uses column attention method and uses LSTM to encode question and headers. There are number of models which are built on top of this with some variations. For example, SQLova [9] uses the BERT encoders instead of LSTM.

```
SELECT $AGG $COLUMN
WHERE $COLUMN $OP $VALUE
(AND $COLUMN $OP $VALUE)*
```

(a) SQL Sketch



(b) Graphical illustration of the dependency in a sketch

Figure 2: Sketch syntax and the dependency in a sketch

3.3 Content Enhanced Model

NL2SQL-Rule [6] uses table content as well while predicting the SQL. It marks vectors to indicate the match of question and headers across two parts and provide it in BERT representations as shown in figure 3. However, ColloQL uses table content in BERT encoders. Another technique on top that is used frequently with these models is Execution Guided (EG) decoding. This was first introduced in [10]. In this technique, partial SQL queries are executed and their results are used as a guidance for decoding process. However, this is not very helpful in real world. In real world applications, tables have records in millions which would make this process very slow and time consuming. Moreover, EG methods change query on the basis of whether empty result is returned, which is not always due to an erroneous query, but it works on WikiSQL dataset setting.

Algorithm 1 The construction for question mark vector

```
vector = [0]*question_length
for cell in table do
  if contains(question,cell) then
    start_index = get_index(question, cell)
    vector[start_index:start_index+len(cell)] = 2
    vector[start_index] = 1
    vector[start_index+len(cell)] = 3
    break
  end if
end for
for one in header do
  if contains(question,one) then
    index = get_index(question, one)
    vector[index] = 4
  end if
end for
```

Figure 3: Knowledge vector using cell data

One of the drawbacks using content to enhance your model is the data privacy. When table content and data is provided in the training step, it's privacy is being compromised. There are some industries where data has privacy issues and can not use for the training of model. In order to address this issue, some work has been done. [11] tried to address this issue using RoBERTa model. It has suggested data agnostic methods. It has eliminated the use of table data and gives the structured query language (SQL) utilizing only table schema and the provided natural language question. Although they were able to completely eliminate table content, there was privacy v/s performance tradeoff. Using this technique, model is capable of zero shot learning and it makes it scalable as well. Headers and natural language question embeddings were created using RoBERTa model and it was provided for prediction along with two knowledge vectors, namely, Question Mark Vector and Header Mark Vector. These are the binary knowledge vectors and important of these vectors are marked with 1 which tells the model the importance and to focus more on this part. It is calculated with the matching tokens across the table schema (Headers) and the natural language question. RoBERTa embedding layer plays an important part of this problem. It is possible that there are very few matches across natural language question and the Headers, therefore, two knowledge vectors will not have useful information. Since RoBERTa is already trained on masked language modeling tasks and they have the sense of knowledge and context, it can be very helpful in solving the above problem.

3.4 IRNet

IRNet is one of the known effective model for this problem [12]. It is used for complex and cross-domain text to structured query language. The main focus of this model has been on two issues. First is to address the mismatch between the queries written in natural language and the generated SQL. Second is to increase the accuracy in generating the column for SQL. Generating the column for SQL has been difficult due to the use of out of domain words.

3.5 X-SQL

Another approach that is used is called X-SQL. It is a new network architecture. X-SQL proposes to reinforce the contextual output from BERT-style pre training model into the structural schema representation, and learn a new schema representation for downstream tasks. This is tested on WikiSQL data set and it shows better results than the state of the art models. Results were improved on development as well as test sets [13].

Name	Review	Dataset Used
Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation (Jiaqi Guo et al., 2019)	In this paper, a neural approach SemQL is used for complex and cross-domain text to structured query language (SQL)	Dataset Spider is used here. It contains large-scale human annotated and cross domain Text-To-SQL bench mark
Content Enhanced BERT-based Text-to-SQL Generation (Tong Guo et al., 2020)	In this paper, the additional input was given to Bert based models which is table content. Table contents provided to deep models resulted in increasing accuracy.	WikiSQL data is used here which is a very large human annotated dataset for natural language to SQL queries.

Name	Review	Dataset Used
Table2answer: Read the database and answer without SQL (Tong Guo et al., 2019)	In this paper, the reasoning part was injected in the deep model. Therefore, the model here takes input the question and the data, and returns the actual data which was queries instead of the SQL	WikiSQL data is used here which is a very large human annotated dataset for natural language to SQL queries.
What It Takes to Achieve 100% Condition Accuracy on WikiSQL (Semih Yavuz et al., 2018)	In this paper, a deeper analysis was performed on the WikiSQL dataset in order to find why we are not able to gain higher accuracy. Focused solutions were implemented here keeping in focus the WikiSQL dataset.	WikiSQL data is used here, which is a very large human annotated dataset for natural language to SQL queries.
X-SQL: REINFORCE CONTEXT INTO SCHEMA REPRESENTATION (Pengcheng He et al., 2018)	Another approach that is presented is called X-SQL. It is a new network architecture. XSQL proposes to reinforce the contextual output from BERTstyle pretraining model into the structural schema representation, and learn a new schema representation for downstream tasks. This is tested on WikiSQL data set and it shows better results than the state of the art models. Results were improved on development as well as test sets.	WikiSQL data is used here which is a very large human annotated dataset for natural language to SQL queries.

Name	Review	Dataset Used
A Comprehensive Exploration on WikiSQL with Table-Aware Word Contextualization (Wonseok Hwang et al., 2019)	In this paper, author demonstrate the effectiveness of table-aware word contextualization on a popular semantic parsing task, WikiSQL. We propose BERT-based table-aware encoder and three taskspecific modules with different model complexity on top of the encoder, namely SHALLOW-LAYER, DECODERLAYER, and NL2SQLLAYER.	WikiSQL data is used here which is a very large human annotated dataset for natural language to SQL queries.
SQLNet: GENERATING STRUCTURED QUERIES FROM NATURAL LANGUAGE WITHOUT REINFORCEMENT LEARNING (Xiaojun Xu et al., 2017)	In this paper, focus has been on solving the "order-matters" problem. In order to convert natural language query to structured query language, we use sequence-to-sequence style model. In order to use this type of model, we have to serialize SQL queries. Here is the problematic part. Since there can be multiple correct queries, it creates different serializations for each query which is not good. There are some approaches which helps overcome this issue. One of the solution for this type of issue is to rely on reinforcement learning to reward the decoder when it comes with any of the correct serializations.	WikiSQL data is used here which is a very large human annotated dataset for natural language to SQL queries.

Table 1: Related Work

4 Research Methodology

4.1 Dataset

WikiSQL is a large semantic parsing dataset. It contains around 24,241 tables and 80,654 natural language questions and it has corresponding structured query languages to those questions. Releasing this dataset has brought much attention to NLP2SQL task. Having this big dataset, it has enabled deep neural techniques to adopt this task. However, it contains simpler queries which do not have any joins and have at most one relation.

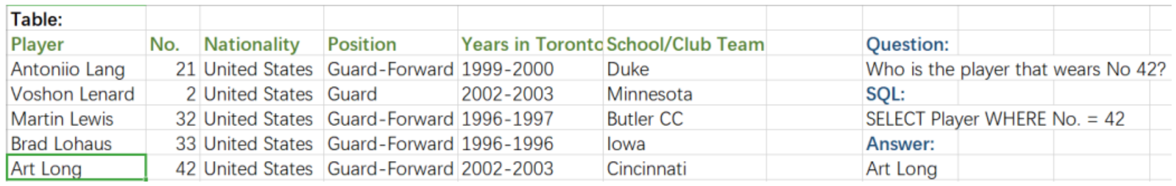


Table:							
Player	No.	Nationality	Position	Years in Toront	School/Club Team		Question:
Antoniio Lang	21	United States	Guard-Forward	1999-2000	Duke		Who is the player that wears No 42?
Voshon Lenard	2	United States	Guard	2002-2003	Minnesota		SQL:
Martin Lewis	32	United States	Guard-Forward	1996-1997	Butler CC		SELECT Player WHERE No. = 42
Brad Lohaus	33	United States	Guard-Forward	1996-1996	Iowa		Answer:
Art Long	42	United States	Guard-Forward	2002-2003	Cincinnati		Art Long

Figure 4: An example of WikiSQL dataset

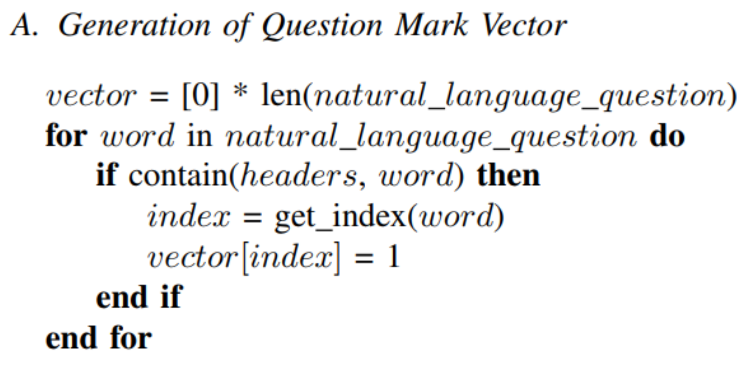
4.2 Implementation

We have implemented two based models based on BERT and RoBERTa, in order to build up our implementation on top of it. In the following section, we will be going through each implementation and their differences along with advantages and disadvantages.

4.3 RoBERTa

In RoBERTa based model, we generate two knowledge vectors to give our model more confidence [14]. These two are binary vectors and tell model which part are more important and need to be focused on by marking that specific parts. They encode the significance of headers and question tokens. Both of these vectors are then concatenated together and passed as additional features to model in order to bring more confidence in models.

4.3.1 Question Mark Vector



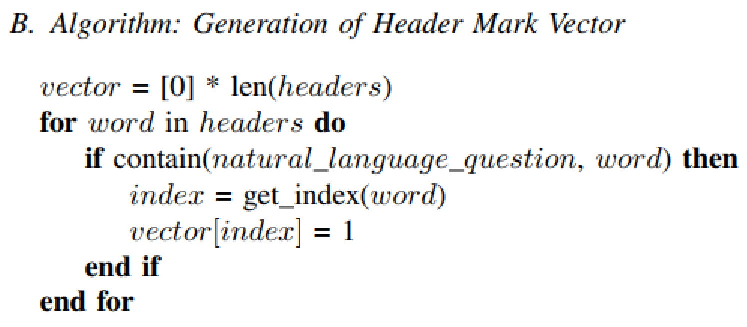
A. Generation of Question Mark Vector

```
$vector$ = [0] * len($natural\_language\_question$)
for $word$ in $natural\_language\_question$ do
    if contain($headers$, $word$) then
        $index$ = get_index($word$)
        $vector[index]$ = 1
    end if
end for
```

Figure 5: Question Mark Vector

First, vector is initialized of same length as natural language question and assigned all the zeros in Fig 2. After initialization, program loops on natural language question tokens. Headers and each token of natural language question is passed in ”contains” function which returns true if the token exists in headers, and false if it does not exist. If the token exists in header, then token index position is marked in vector with 1, otherwise, value remains 0 as per initialization.

4.3.2 Header Mark Vector



B. Algorithm: Generation of Header Mark Vector

```
$vector$ = [0] * len($headers$)
for $word$ in $headers$ do
    if contain($natural\_language\_question$, $word$) then
        $index$ = get_index($word$)
        $vector[index]$ = 1
    end if
end for
```

Figure 6: Header Mark Vector

For header mark vector, we follow same procedure. Frist, vector is initialized of same length as headers and assigned all the values to zero in Fig 3. After initialization, program loops on headers. Each header and natural language question is passed in "contains" function which returns true if the header exists in questions, and false if it does not exist. If the header exists in question, then header index position is marked in vector with 1, otherwise, value remains 0 as per initialization. One thing to notice here is that loop iteration is not over headers here, rather it is on individual words in headers. Moreover, one thing that is majorly different here from [6] is that we do not iterate through data here, hence maintaining the data integrity and privacy.

4.4 BERT

In BERT based model, we generate two knowledge vectors as well to give our model more confidence. These two are binary vectors same as in the case of RoBERTa and tell model which part are more important and need to be focused on by marking that specific parts. They encode the significance of headers and question tokens. Both of these vectors are then concatenated together and passed as additional features to model in order to bring more confidence in models. The main difference comes in the encoding. Here we encode based on the actual data which compromises the privacy of the data. Here we need the actual data for the encoding and for the learning phase of the model.

4.4.1 Question Mark Vector

For the encoding of Question mark vector, we create a vector of question's length initialized with zeros only. After initialization, we iterate through each cell in table data and if there is a match between cell data and question, then respective position of question vector is marked. One important thing to note here is that we are iterating through table data as shown in figure 7. This compromises the privacy of data. Although this increases the accuracy of the model due to training on actual data, it exposes actual data to model.

Algorithm 1 The construction for question mark vector

```
vector = [0]*question_length
for cell in table do
    if contains(question,cell) then
        start_index = get_index(question, cell)
        vector[start_index:start_index+len(cell)] = 2
        vector[start_index] = 1
        vector[start_index+len(cell)] = 3
        break
    end if
end for
for one in header do
    if contains(question,one) then
        index = get_index(question, one)
        vector[index] = 4
    end if
end for
```

Figure 7: Knowledge vector using cell data

4.4.2 Header Mark Vector

For this vector, we initialize vector with zeros of header length. We iterate though cell data as well for marking positions of great importance which can convey useful information to the model. Once again, this vector as well compromises the privacy of actual data while training.

Algorithm 2 The construction for table header mark vector

```
vector = [0]*header_length
index = 0
for one in header do
  if contains(question, one) then
    vector[index] = 1
  end if
  index = index + 1
end for
for cell in table do
  if contains(question,cell) then
    vector[get_column_index(table, cell)] = 2
  end if
end for
```

Figure 8: Header vector using cell data

4.5 Model

Sequence to sequence style model is not recommended to use here because of the ”order-matters” problem. It is possible for SQL query to have more than one correct queries. It is possible to move some conditions ordering and still get correct result. However, sequence to sequence style model choses one ordering correct and it labels other queries as wrong. In order to overcome this issue, another approach is used here suggested by SQLNet [8]. It uses sketch based approach to predict SQL queries. However, unlike sequence to sequence, it eliminates the task of predicting complete queries and reduced that task into predicting some parts of it only. Hence the order matters problem is eliminated here. SQL query sketch:

SELECT $\{aggregate\}$ $\{column\}$ **FROM** $\{table\}$
  WHERE $\{column\}$ $\{operator\}$ $\{value\}$
      AND*
  WHERE $\{column\}$ $\{operator\}$ $\{value\}$
    *(Repeating **WHERE** blocks)

Figure 9: Query Sketch

In fig 3, we can see all the parts which will be filled by models predictions instead of complete queries.

Natural language question along with table headers is given input to RoBERTa model. It gives us embeddings which are then used for downstream tasks. These embeddings along with the concatenation of question mark vector and header mark vector is passed into sub models in order to predict and fill SQL sketch. Model architecture is shown in Fig 5.

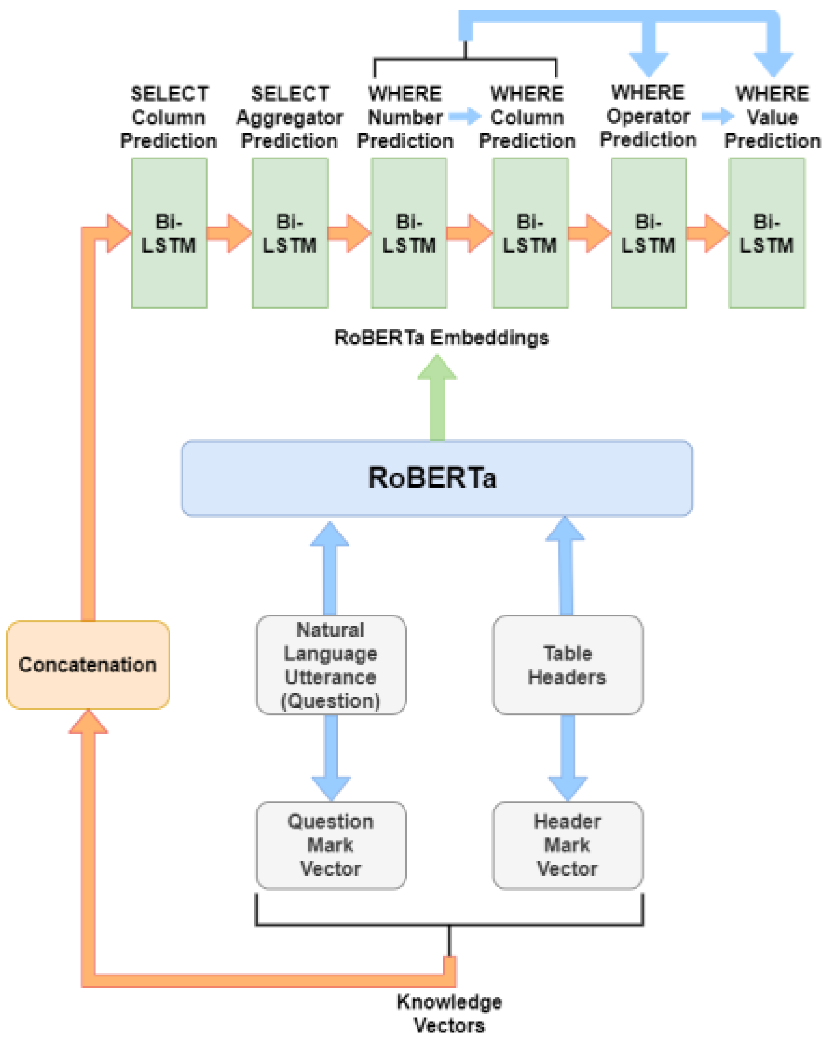


Figure 10: Model Architecture

Embedding layer in this architecture plays an important role. Since we pass two knowledge vectors as well along with RoBERTa embedding into sub models for downstream tasks, it is important to not make our model depend on these two knowledge vectors completely. It is because there is a possibility that natural language question does not match with headers token, hence, not giving much information. If there are no matches between natural language questions and header tokens, then vectors would be consisting of zeros and would not give much information to model. Therefore, embeddings is very important and that is why RoBERTa embeddings is used here.

It is important for model to extract significance amount of information from natural language question and table headers. Hence, both of these information is passed in RoBERTa model as shown in Fig 4. Similar architecture is used in [6], but vectors are generated using table content, hence, compromising data privacy. Another reason to use RoBERTa model is to have some sense of knowledge and context in model so that it can understand the meaning is same of even mismatched tokens. RoBERTa is pre trained on very large corpus of 160 GB and on masked language modeling, next sentence predictions tasks. This gives the model desired sense of knowledge and context.

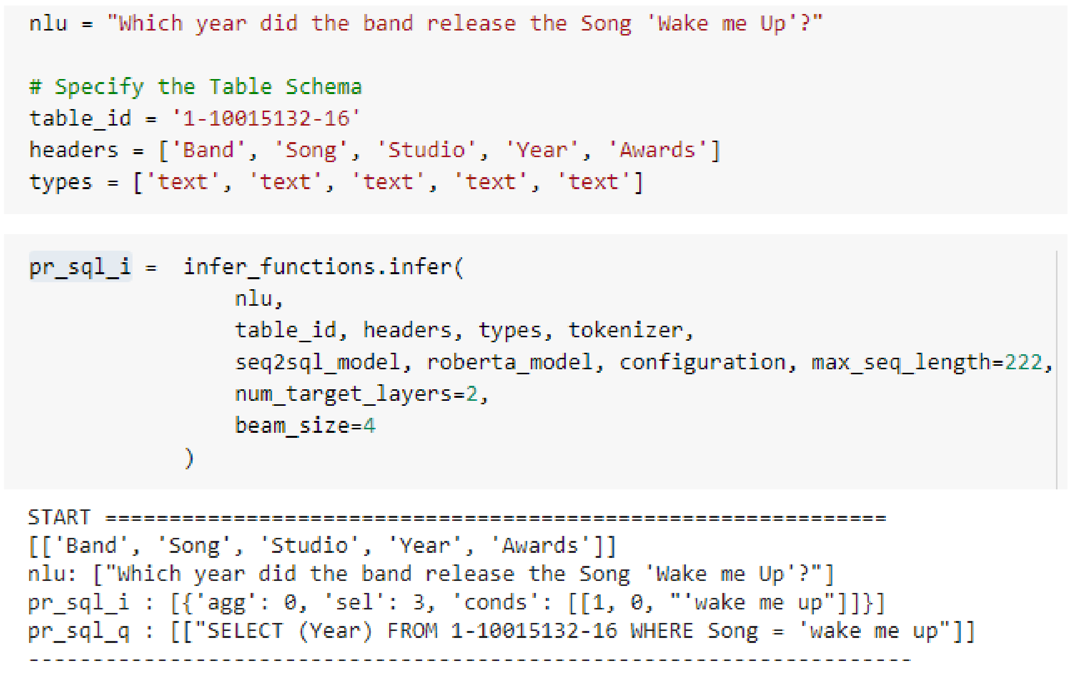
5 Reproducing State of the Art

We have implemented two state of the art models [6], [5], giving us the inside of core implementations. [6] is the basic model and [5] is built on top of this model. Therefore, understanding these two implementations and their differences is very important.

5.1 BERT and RoBERTa Model

BERT and RoBERTa are pre trained model on a very large corpus about 16 GB and 160 GB data respectively. They are trained on masked language modeling task and next sentence prediction task as well. This training on large corpus gives them the sense of "knowledge" and "context". [6] uses two knowledge vectors as well explained in fig. 2 and fig. 3. However, the major difference comes in Question Mark Vector. In [6], loop goes through the cell values where as in [11] goes through header tokens, hence respecting the data privacy. This is why we have seen [11] model has privacy v/s performance trade off. [11] has less accuracy compared to [6]. However, [11] model does not need data for training, hence, it has zero shot learning.

After training model we were able to generate similar results as mentioned in paper [11]. Below is an screenshot of inferring query from natural language question after training the model.



```
nlu = "Which year did the band release the Song 'Wake me Up'?"

# Specify the Table Schema
table_id = '1-10015132-16'
headers = ['Band', 'Song', 'Studio', 'Year', 'Awards']
types = ['text', 'text', 'text', 'text', 'text']
```

```
pr_sql_i =  infer_functions.infer(
                nlu,
                table_id, headers, types, tokenizer,
                seq2sql_model, roberta_model, configuration, max_seq_length=222,
                num_target_layers=2,
                beam_size=4
            )
```

```
START =============================================================
[['Band', 'Song', 'Studio', 'Year', 'Awards']]
nlu: ["Which year did the band release the Song 'Wake me Up'?"]
pr_sql_i : [{'agg': 0, 'sel': 3, 'conds': [[1, 0, "'wake me up"]]}]
pr_sql_q : [["SELECT (Year) FROM 1-10015132-16 WHERE Song = 'wake me up"]]
----------------------------------------------------------------------
```

Figure 11: Infering Query

6 Enrichment via Similar Sense Words

We have built our experiment on top of the RoBERTa model.

6.1 Similar Sense Words

The WordNet database contains many sorts of relationships between words which help us understand relations between different words. It can categorize words into different hierarchies and answer many interesting questions. If we have a single word ”run” and we would like to understand its sense and relations between different words which are in close proximity to it. WordNet helps us understand different senses and different relations of words with each other. Example shown in figure 12.

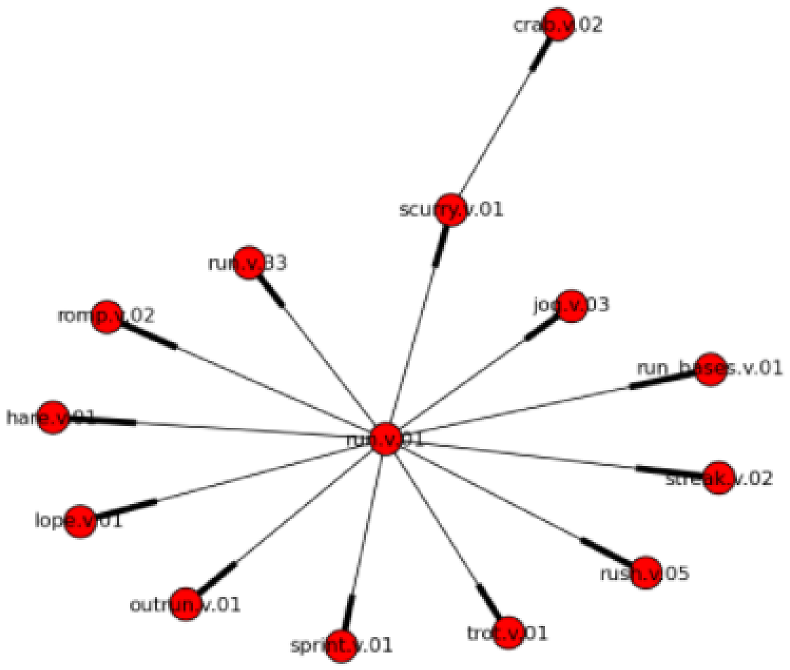


Figure 12: WordNet

6.2 Implementation

We have taken help of WordNet in creating of our two knowledge vectors, header knowledge vector, question knowledge vector. While marking these vectors in order to provide useful information to model with the areas where to focus on, we used to look for exact matches in the headers and questions. Through this method, our vectors were loosing much information which could help model gain some accuracy and show better results. In our implementation, we did not look for exact matches, rather, we created two synsets and calculated the probability of their match which helped us better understand the relations of words rather than comparing it straightforward. We marked vectors if the probability was greater than or equal to 0.7.

6.3 why?

It is quite common in natural language that we sometimes use synonyms in our sentences and not exact words which were affecting results in our problem domain. Not only synonyms, but the use of singular words and plural words were also affecting our model, since, singular and plural does not result in exact match as well.

6.4 Analysis

When we started enriching the header and knowledge vectors with the help of similar sense words, we started tracking how many words were benefiting from it. We started tracking count of words which were completely same and count of those words as well which were close to each other. This gives us the clear picture in terms of percentage that how many words are being benefitted from our implementation. Comparison can be seen in figure 13

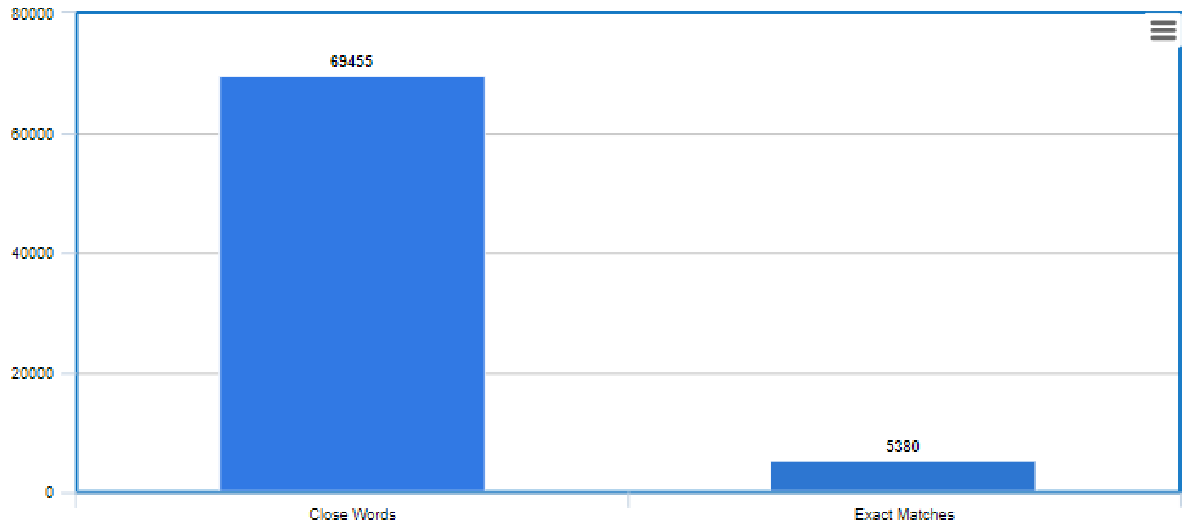


Figure 13: Word Matches

7 Analysis and Results

We have evaluated our model on the WikiSQL dataset with the logical form and execution accuracy metrics. The method of calculating the execution

Model	Dev logical form acc.	Dev exec acc.	Test logical form acc.	Test exec acc.
BERT	84.3%	90.3%	83.7%	89.2%

Table 2: Overall result of BERT Model on WikiSQL task

Model	SELECT column	SELECT agg	WHERE number	WHERE column	WHERE op	WHERE value
BERT	97.4%	90.0%	99.1%	97.9%	98.1%	97.6%

Table 3: Break down result of BERT model on WikiSQL task

accuracy does involve the model querying the database which goes against our core objective of data privacy, but this is not a necessary step and neither does it affect the model's training in any way. The only purpose of such querying is to obtain an additional evaluation metric to better understand our model's performance. The final model when deployed would not need to calculate any such metrics, thus keeping the data completely private. We have compared our model with the state of the art model as well as the baseline SQLNet Model: Note: Accuracy has been shortened to acc. and execution has been shortened to exec.

7.1 BERT with Content Enhanced Implementation

BERT model implementation with table data included in training gives promising results. However, it compromises the privacy of the data. This implementation is carried out with traditional two knowledge vectors without the additional input of WordNet, results are shown in table 2 and table 3.

7.2 RoBERTa with Traditional Knowledge Vectors

We implemented RoBERTa model without using table data. It did ensure the privacy of data, however, it did come with its trade off. We marked two knowledge vectors using only headers and natural language question. In this model, we did not use any real data for training of the model, rather, we used only schema information of the table. Results are shown in table 4 and

Model	Dev logical form acc.	Dev exec acc.	Test logical form acc.	Test exec acc.
RoBERTa	74.4%	80.9%	73.3%	80.3%

Table 4: Overall result of RoBERTa Model on WikiSQL task

Dataset	SELECT column	SELECT agg	WHERE number	WHERE column	WHERE op	WHERE value
Dev	96.2%	90.6%	98.0%	91.3%	93.4%	90.5%
Test	95.8%	90.4%	97.2%	89.8%	92.4%	89.7%

Table 5: Break down result of RoBERTa model on WikiSQL task

table 5.

7.3 RoBERTa Model with Header Knowledge Vector with WordNet

We implemented RoBERTa model without using table data. We marked two knowledge vectors using only headers and natural language question. However, the main difference here is how we did the encoding of Header knowledge vector. We encoded vector here with the help of WordNet as explained in Section 6. Results are shown in table 6 and table 7.

Model	Dev logical form acc.	Dev exec acc.	Test logical form acc.	Test exec acc.
RoBERTa	81.0%	86.0%	80.4%	85.4%

Table 6: Overall result of RoBERTa Model on WikiSQL task with WordNet Header vector

Dataset	SELECT column	SELECT agg	WHERE number	WHERE column	WHERE op	WHERE value
Dev	97.0%	90.8%	98.8%	93.8%	97.7%	95.6%
Test	96.8%	90.6%	98.3%	93.3%	97.2%	95.2%

Table 7: Break down result of RoBERTa model on WikiSQL task with WordNet Header vector

Model	Dev logical form acc.	Dev exec acc.	Test logical form acc.	Test exec acc.
RoBERTa	81.4%	86.6%	80.5%	85.8%

Table 8: Overall result of RoBERTa Model on WikiSQL task with WordNet Header & Question vectors

7.4 RoBERTa Model with Question & Header Knowledge Vectors with WordNet

We implemented RoBERTa model without using table data. We marked two knowledge vectors using only headers and natural language question. However, the main difference here is how we did the encoding of Header knowledge vector as well as Question knowledge vector. We encoded vectors here with the help of WordNet as explained in Section 6. Results are shown in table 8 and table 9.

7.5 Comparison of Final Results

Final comparison of test results are shown in table 10 and table 11.

Dataset	SELECT column	SELECT agg	WHERE number	WHERE column	WHERE op	WHERE value
Dev	97.2%	90.8%	98.8%	94.0%	97.9%	95.9%
Test	96.7%	90.7%	98.3%	93.4%	97.3%	95.3%

Table 9: Break down result of RoBERTa model on WikiSQL task with WordNet knowledge vectors

Model	Dev logical form acc.	Dev exec acc.	Test logical form acc.	Test exec acc.
RoBERTa base Model	74.4%	80.9%	73.3%	80.3%
Our Model	81.4%	86.6%	80.5%	85.8%

Table 10: Comparisons of base and our final Model.

Model	SELECT column	SELECT agg	WHERE number	WHERE column	WHERE op	WHERE value
RoBERTa base model	95.8%	90.4%	97.2%	89.8%	92.4%	89.7%
Our Model	96.7%	90.7%	98.3%	93.4%	97.3%	95.3%

Table 11: Comparisons of base and our final sub Models.

8 Conclusion

In this experiment, the focus was on building the model without using the data available in database due to obvious privacy issues. This is a data agnostic model. We have seen that existing models use table content as well as a feature when inputting the model. Moreover, if some models are not using the real data, they make use of execution guided decoding. We have not used any of these two techniques. We have developed a data blind model using only table schema and two important knowledge vectors with the help of WordNet.

Our results show promising improvements when even one vector is implemented with WordNet. Moreover, slightly more improvement is seen when both knowledge vectors where marked with the help of WordNet. Comparison between one vector, both vectors and no vectors for dev dataset is shown in figure 14 and for test dataset is shown in figure 15

9 Future Work

Our work opens a new direction to explore which can yield many promising results. There are many possibilities inside the work of finding a similar sense of words and enriching the knowledge vectors which can provide new insights of data to model. Moreover, there could be dynamically selection of probability values while finding the different senses among words. These new findings can be tried with our previous existing knowledge in different combinations which can lead to new dimensions of research.

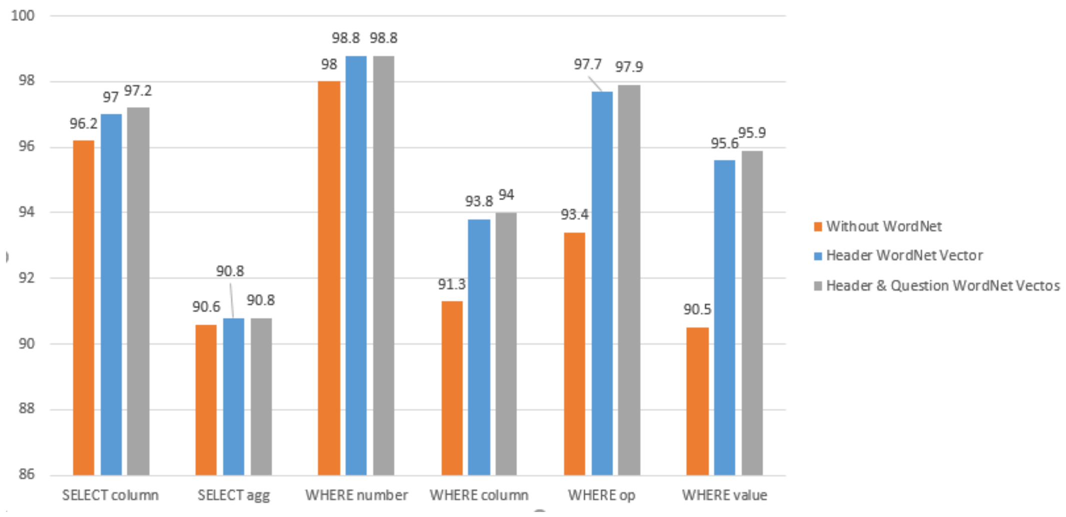


Figure 14: Comparison betwen Without WordNet, Header WordNet & Question WordNet Vectors for Dev

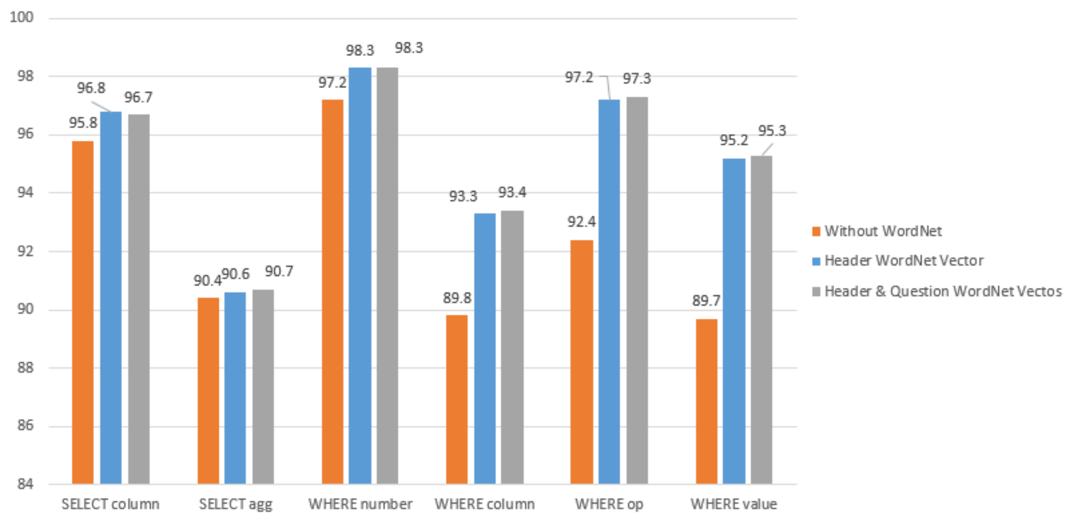


Figure 15: Comparison betwen Without WordNet, Header WordNet & Question WordNet Vectors for Test

References

[1] V. Zhong, C. Xiong, and R. Socher, "Seq2sql: Generating structured queries from natural language using reinforcement learning," *arXiv preprint arXiv:1709.00103*, 2017.

[2] V. Setlur, M. Tory, and A. Djalali, "Inferencing underspecified natural language utterances in visual analysis," in *Proceedings of the 24th International Conference on Intelligent User Interfaces*, pp. 40–51, 2019.

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

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[5] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[6] T. Guo and H. Gao, "Content enhanced bert-based text-to-sql generation," *arXiv preprint arXiv:1910.07179*, 2019.

[7] V. Setlur, S. E. Battersby, M. Tory, R. Gossweiler, and A. X. Chang, "Eviza: A natural language interface for visual analysis," in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, pp. 365–377, 2016.

[8] X. Xu, C. Liu, and D. Song, "Sqlnet: Generating structured queries from natural language without reinforcement learning," *arXiv preprint arXiv:1711.04436*, 2017.

[9] W. Hwang, J. Yim, S. Park, and M. Seo, "A comprehensive exploration on wikisql with table-aware word contextualization," *arXiv preprint arXiv:1902.01069*, 2019.

[10] C. Wang, K. Tatwawadi, M. Brockschmidt, P.-S. Huang, Y. Mao, O. Polozov, and R. Singh, "Robust text-to-sql generation with execution-guided decoding," *arXiv preprint arXiv:1807.03100*, 2018.

[11] D. Pal, H. Sharma, and K. Chaudhari, "Data agnostic roberta-based natural language to sql query generation," *arXiv preprint arXiv:2010.05243*, 2020.

[12] J. Guo, Z. Zhan, Y. Gao, Y. Xiao, J.-G. Lou, T. Liu, and D. Zhang, "Towards complex text-to-sql in cross-domain database with intermediate representation," *arXiv preprint arXiv:1905.08205*, 2019.

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

[14] D. Pal, H. Sharma, and K. Chaudhuri, "Data agnostic roberta-based natural language to sql query generation," in *2021 6th International Conference for Convergence in Technology (I2CT)*, pp. 1–5, IEEE, 2021.