ACL-SQL: Generating SQL Queries from Natural Language

Ronak Kaoshik*, Prakash R*, Rohit Patil*, Shaurya Agarawal*, Naman Jain
Undergraduate, Indian Institute of Technology Gandhinagar, Gujarat, India
{ronak.kaoshik,prakash.r,rohit.patil, shaurya.agarawal, naman.j}@iitgn.ac.in

Abstract

A considerable amount of data in the world exists in the form of a relational database. Database System exists everywhere in the world from a small restaurant to financial markets. Due to inadequate knowledge of how SQL works, most people aren't able to retrieve information from the database. We try to solve this challenge by developing a deep learning-based model that translates natural language queries to SQL queries. Most of the existing model works on a database with one table. In this implementation, we propose a model for generating SQL queries from a natural language where we use the ACL anthology database, which consists of 13 tables. This model has better accuracy than the current state of the art model on a multi-table. Our model leverages the structure of SQL queries to reduce the output space of generated queries significantly. Along with this paper, we are publishing an annotated dataset of 3100 instances, each with natural language questions and particular SQL query.

Keywords: POS Tagging, XGboost, Graph Traversal, BFS, NER, Dependency Tree, Fine-tuned BERT, Dataset Creation, Dataset Augmentation, Syntax SQL

1. Introduction

A significant amount of the data generated in the world, such as medical records, financial market stats, etc. is stored in the form of relational databases. To learn a database management language is not a trivial task for most of the people. In recent years, a lot of plausible solutions based on deep learning have been proposed for this problem statement wherein the user gets the SQL query by giving a purely natural Language sentence describing the need. However, most of the works are focused towards databases with single tables and the real-world databases mostly contain multi-tables. Our solution is a combination of multiple Natural Language Processing methods from traditional heuristic-based approaches up to the most recent breakthrough, namely BERT.

The above problem statement has two aspects; human interaction and language processing, which constitutes a Natural language interface. Natural language interfaces, a research area at the intersection of NLP and interactions between humans and computers, seeks to provide people with ways of communicating with machines by using natural language. We investigate one particular aspect of NLI applied to relational databases: translating natural language questions to SQL queries. This particular problem statement deals with a sub-domain of Natural Language Processing namely Question Answering.

The existing approaches tackling the problem generally involve training over a large dataset(Yu et al., 2018b; Yu et al., 2018a; Wang et al., 2019). We try to solve the problem using the training dataset of multi relation database with much smaller size as compared to the previous ones. We have created Antho-SQL based on the ACL-anthology database. We generated around 3100 natural language-query pair with 310 uniquely intended queries each being upto 25 words long and depending on up to 5 tables.

Our solution tackles the problem by dividing it into three major sub-tasks, namely, the Table-column pair prediction, link prediction, and the condition prediction. For the first subtask, we used a decision tree that considers a combination of POS tagging and bigram model-based prediction at the first layer, and if this fails, it makes use of fine-tuned BERT embeddings based predictor for the Table-column pair. We used a similar approach for the third sub-task wherein a relation needs to be established between the specific columns and the named entities. For getting the named entity, we used a named entity recognition model that was fine-tuned for our database, whereas a BERT based model is used for the condition prediction. The link is prepared by running a BFS over the graph of table linkages, which returns the shortest path in SQL format with almost 100% accuracy. Our solution advocates a novel architecture that has the flexibility for implementation on databases with varied schemas and can be fine-tuned for the databases to achieve quite accurate results.

2. Related Work

The task of converting a natural language query into SQL query is a mapping from plain simple text to a more structured and formal language which is used worldwide for database related operations. The problem is mostly presented in the literature as slot filling or semantic parsing task(Wang et al., 2019).The semantic and syntactic information in the question and the table schema is used to generate a predefined SQL format (Dong and Lapata, 2018; Wang et al., 2019).Another approach is using Language generation methods. Language generation models are directly applied to the input question and the table schema. These approaches proposed in the database community tend to involve user interaction with the system(Yu et al., 2018a). SyntaxSQLNet presents a SQL-specific grammar which enables direct prediction of SQL clauses(Yu et al., 2018a). WikiSQL is the dataset used in SyntaxSQLNet.

*Equal Contribution

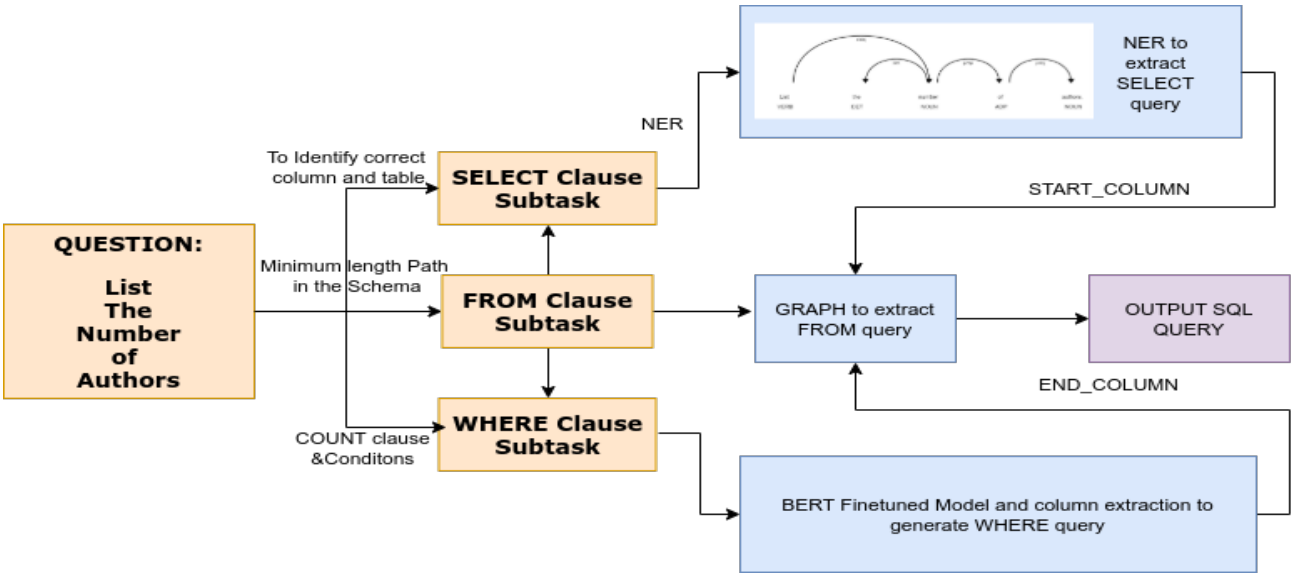

Figure 1: Model Flow

We primarily focus on techniques which rely less on dataset for training. We use our own dataset, ACL-SQL which is much smaller than existing multi relational datasets like Spyder(Yu et al., 2018b) for training. SyntaxSQLNet uses vector representation of the question as input to the model, we are also using the same in some parts of our model. They also encode the table schema along with the encoded question as an input to the SQL query generation model. The Seq2sql model(Zhong et al., 2017) uses a deep neural network for generation of SQL query. The three component model uses policy based Reinforcement learning for generating the clauses, the generated query is executed on the database and the execution result is mapped as input to the model, thereby causing a feedback loop in its architecture to improve the accuracy(Zhong et al., 2017).

In our model, as a part of preprocessing, we encoded the schema (natural numbers mapping) and embedded it as a part of the model. We also converted schema to a graph which has tables as nodes and columns as information embedded in them to perform operations such as path traversals for our architecture. We faced several domain specific challenges related to dataset. For example, consider the question, 'List the authors of the paper $"Paper_name"$ ', in these type of questions, we want the model to answer the SQL query regarding corresponding author names, but after analysing and preprocessing the question, the model tend to output the query which will list Author IDs not the particular Author names. Also the datasets used for training (Wang et al., 2019; Yu et al., 2018a) are huge, but in the case of a comparatively small dataset, applying neural-network based approaches won't help gaining the accuracy on the test data, as the model would have been trained on a small dataset and may be biased towards generating some type of queries. These challenges have motivated us to develop an architecture which tries to come to grips with them and to test our model, we have developed the ACL-SQL dataset consisting of question and SQL query pairs from an existing ACL Anthology database(Singh et al., 2018a).

3. Existing Dataset

3.1. WikiSQL

WikiSQL (Zhong et al., 2017) is a standard dataset used by many paper working on this field. The number of instances in the dataset are 80654 in which all are labelled by hand. It is distributed across 24241 tables from wikipedia. It is significant larger in magnitude than any other dataset for SQL Generation. It also contains an optional annotation script that uses Stanford CoreNLP. This script annotates query, question, and SQL table, input and output sequence creation for ease with using Seq2Seq models. But, it does not have queries which contain more than one table. Each Natural language question is framed for only one table.

3.2. ATIS,Geo

Each of them contain only limited number of SQL queries, and has exact same SQL queries in train and test split. They are also outdated in terms of SQL clause used by annotators.(Price, 1990)(Tang and Mooney, 2001)

3.3. Spider

Spider is the only one text-to-SQL dataset that contains both databases with multiple tables in different domains and complex SQL queries which contains more than one table.(Yu et al., 2018b). The average question length and SQL length are about 13 and 21 respectively.

4. Dataset Curation

4.1. Database used to generate SQL queries:

There is a need to create Domain specific Dataset because it leads to better performance than an model trained on general Domain Dataset. We used the ACL-Anthology dataset (Singh et al., 2018b) whose schema is given in 5. It consists of 13 tables and 6 foreign key columns. On average a table contains 2.15 columns. AuthID, FieldID, PaperID,

KeywordID,ConfID and AffiliationID are the foreign keys. Each of the table contains around 5,000 rows.

4.2. ACL-SQL Dataset Description

Annotation took 4 people 60 hrs in total. The detailed process of annotation is described in 2.

We started with generating 310 unique SQL query and then paraphrased the natural language queries to generate 620 pair of Natural language and SQL queries. We used placeholder instead of a entity. Entity here means a valid entry of each column. Placeholders where replaced and augmented with respective entities.

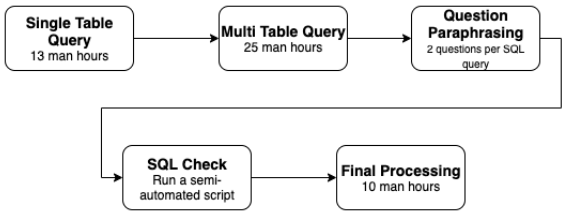

Figure 2: The Annotation Process

4.2.1. SQL clauses

We ensure that our dataset contains multiple entries of major SQL clauses. It contains 310 unique SQL entries with SELECT , FROM , WHERE , GROUP BY , ORDER BY , JOIN , HAVING , COUNT and DISTINCT clauses.

4.3. Dataset Augmentation

Placeholder are column name enclosed within $$. One such placeholder is $AuthID$ and it's representative replacement would be any entity of AuthID column. We randomly choose some entity to replace the placeholders. Below is one such augmented example. We also took of the relative scaling of the augmentation. That is, if the number of entity for a column is less than the other then the number of augmented entries of that column will be less than the other.

```
Natural Language Query : What are the
papers written by '\$AuthID\$'?
Augmented Natural Language Query: What
are the papers written by 'A-2966'?
```

4.4. SQL Review

We made a script which generate 5 random row output for a given SQL query. Then we ran it over the entire dataset and checks it matches our expectations.

4.5. Annotation tools

We open our csv data on a terminal interface powered by python library querycsv. It allows the annotators to see the schema and content of each table, execute SQL queries, and check the returned results. This library was extremely helpful for the annotation of complex SQL queries and was also used in the semi-automated SQL checking script.

	Number of Queries
Single Table	145
Two Table	23
Three Table	118
Four Table	18
Five Table	6

Table 1: Statistics on number of query consisting of multi tables.

4.6. Dataset Statistics

The distribution of query over number of tables involved in a query is given in 4. On average there are 3 tables involved in each query. The dataset was later divided into Non-complex and complex dataset. If the query contains multi table select column or contains complex SQL clauses like HAVING BY , GROUP BY , ORDER BY then it was considered complex. The proposed model was crafted keeping non-complex dataset in mind. In future we plan to enhance the model for Complex dataset as well. Once such non complex dataset entries is given below,

Natural Language Query : What is the Name of the Author with ID A-18390 ?
SQL Query: SELECT Name FROM AuthID_Name WHERE AuthID='A-18390';

5. Model

For multi-tables, there were very few natural language sentences and SQL query pairs available online. So we proposed our own architecture. Our solution tackles the problem by dividing it into three major sub-tasks namely, the Table-column pair prediction, link prediction, and the condition prediction as shown in the figure.

For the first subtask, we used a decision tree that considers a combination of POS tagging and bigram model-based prediction at the first layer and if this fails, it makes use of fine-tuned BERT embeddings based predictor for the Table-column pair. A similar approach has been used for the third sub-task wherein a relation needs to be established between the specific columns and the named entities. For getting the named entity, we used certain combination of NER methods and POS tagging for our database, whereas a BERT based model is used for the condition prediction. The link is prepared by a running a BFS over the graph of table linkages or the database schema which returns the shortest path in SQL format. Our solution advocates a novel architecture which has the flexibility for implementation on databases with varied schemas and can be fine-tuned for the databases to achieve quite accurate results.

5.1. NER

The natural language sentences given by the users will be containing special entities such as names of places, authors, years, etc. In order to identify these entities we used the

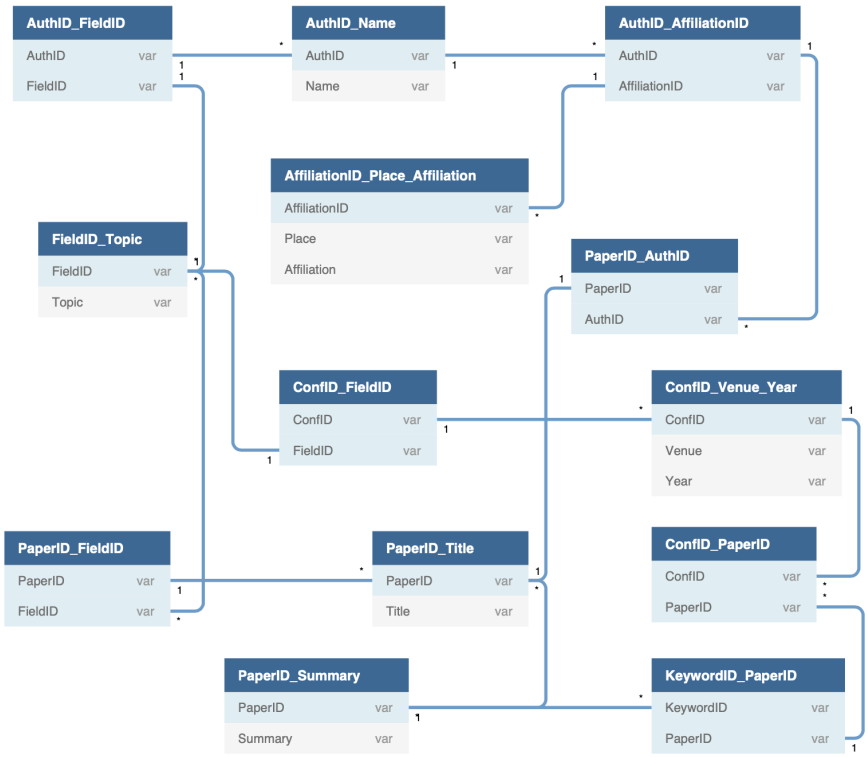

Figure 3: Schema of ACL Anthology Dataset

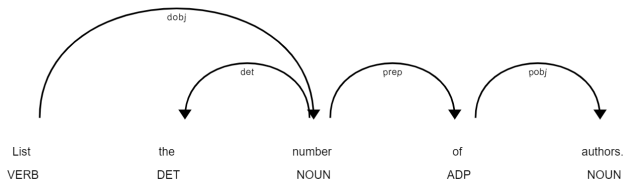

Figure 4: POS Tagging

POS tagging approach. The input natural language sentence was tokenized, with the tokens being tagged with the type of POS. The named entity contained at least a single Proper Noun. In other cases it appeared within special punctuation marks such as quotes or double quotes. Considering both these cases, we identified the named entity and compared it with the database in order to obtain the column values. Hence, this subtask gave the output as the named entity along with the column name which will further be used in the where condition space. Also, it returns the natural language sentence without the named entity.

5.2. SELECT-COLUMN Prediction

Once the NER sub-task returns a natural language sentence without the named entity, this sentence is then processed for identifying the column name that appears between the SELECT and FROM parts of the SQL query.

For the column name identification, various BERT based methods were tried but the accuracy was below satisfactory levels. Due to this, we prepared our own algorithm, that simplistically identified the column names from the sentences. Firstly, we tokenized the natural language sentence and processed the POS tags corresponding to the tokens. Then we tried to find an underlying relation between the column name required and the POS tags. We found that the column names usually exist in some or the other word forms of the column names in the sentence itself. These terms will be usually tagged as either nouns, proper nouns or verbs. Once these specific POS tagged terms were separated from the natural language sentence, each of the term was made to match with the column names using the Levenshtein distance.

We had tried using simple edit distance for the task but the former proved to be much more accurate.

The terms having the Levenshtein ratio within a certain threshold value were identified. The threshold value of the ratio was identified with the help of multiple iterations of manual checking. The threshold value of 0.55 turned out to hold true for most of the iterations.

After this, the column names were coupled with the ratio values and were sorted in descending order of the magnitude of the values. The complete list of these columns in the above order was then sent as the output of this sub-task.

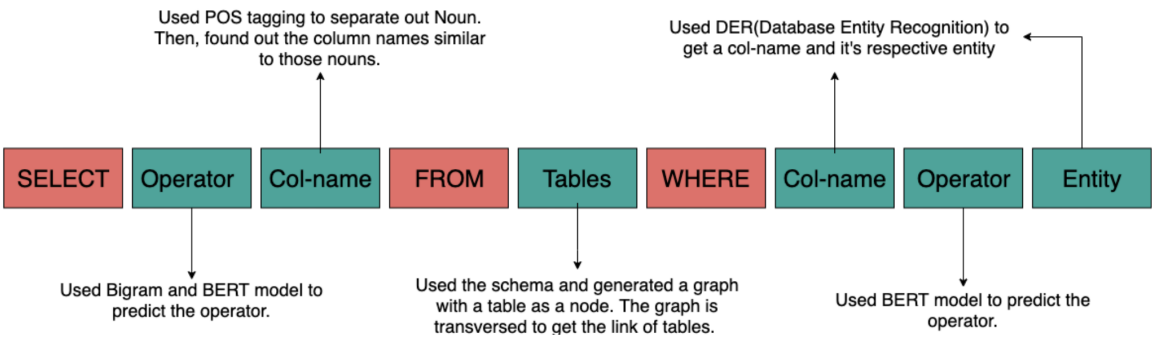

Figure 5: Model Summary

5.3. Link Prediction

Many of the questions deal with data involving two or more tables. The SQL query requires all these linkages between the table names in order to process the query for the given information. If the link is not mentioned correctly, the query falls short of enough information to look for the required condition. Hence, the link prediction task plays a vital role in the SQL query structuring task and it needs to be as accurate as possible.

Before carrying out link prediction, the list obtained from the previous sub-task was re-evaluated. If there were any column names similar between the ones just obtained and the one obtained from the NER sub-task, then only the uncommon ones were left for further processing. For the link, the first column name from the above list and the NER sub-task column name were used.

The link was prepared based on the database schema. The schema was represented in the form of a graph with the nodes as the table names and the edges denoting whether a particular table node has some column common with another table node. This link prediction task is required to have inner joins on the table names in the SQL query.

The graph currently needs to be manually made from the table names but there's a way to have this task automated as well. Considering for now that the graph is made manually. Now, two table names are found corresponding to the two column names from the previous steps. This is done using simple matching algorithms between the column and the table name. Now, one of the table node just obtained is considered as the root and a BFS is run over the graph for the other table node. As soon as the other node is found, a set of all tables is returned which lie on the link between the root node and the leaf final node. These are then formatted as per the SQL format and given as the output. Also, the table names corresponding to the column names are given as output because in the SQL query all the sub-parts containing column names require a corresponding table name for proper functioning of the query.

5.4. Condition prediction:

The only part of the SQL query left to be completed is the condition of the WHERE clause. There are many operators that appear here such as $<, >, >=, <=, ==$ etc. Obtaining the knowledge of these conditions directly from the sentence wasn't straightforward. Hence we took the deep learning approach in order to find a correspondence function between the natural language sentences and the conditions.

For this subtask we directly made use of the BERT sequence classifier. The operators were considered as the classes and the input are the BERT embeddings of the natural language sentences. The classifier was fine tuned for our task. Once, trained the classifier was able to predict the condition from the sentence quite accurately. Also, a similar classifier was fine tuned for predicting whether the query would contain 'count' or the column name in between 'select' and 'from' terms of the query.

6. Experimentation

6.1. Dataset

As mentioned above, we generated our own dataset (containing Natural Language Queries with their SQL counterparts) based on ACL Anthology database (Singh et al., 2018b) for testing our architecture. The dataset consists of 310 unique queries which were further categorised into complex and non-complex queries based on the number of clauses. For implementation purposes our model was tested on non-complex queries because the complex queries have clauses like AND, HAVING, ORDER BY etc. which would not be predicted by our current model. Non-complex ones consisted of 220 queries while complex ones have the remaining 90.

	Complex	Non-Complex
No. of Queries	220	90
Max No. of tables	5	4
Max No. of clauses	12	7

Table 2: Statistics of complex and non-complex queries.

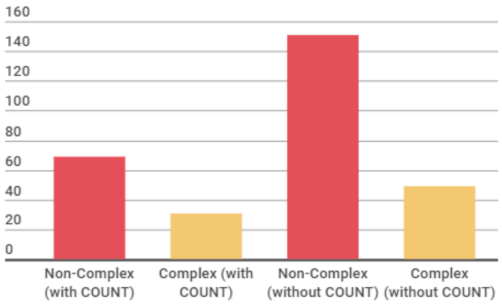

Figure 6: Dataset Distribution

6.2. SELECT-COLUMN Prediction

During the process of generating dataset, we changed our structure of the 'basic' SQL query to account that there is no repetition in the answers to the SQL queries. We added a 'DISTINCT' in front of all queries as it does not affect the queries which did not have the repetition in the first place while making sure that it does not happen in the queries in which repetition was happening in the first place.

We also had to deal with the issue of whether to use brackets in the SQL queries to separate out different clauses and maintain their order of execution. This issue was important in terms of execution and testing of our model as the model could produce the wrong number of parenthesis, or they might be invalid in some cases causing the SQL queries to not work which might have worked otherwise. We checked and tested every query with and without the use of parenthesis, and we concluded that there was no difference in answers for both of them, hence we decided to stick with no use of parenthesis rule when unnecessary while writing the queries.

'COUNT' clause may take columns inside the parenthesis whose production is not in the scope of our current model. So we decided to put queries with 'COUNT' having column names inside them in the complex division. We focused on production of 'COUNT(*)' in the current model.

The implementation issue of placeholders was taken care of our predefined convention to write the column names inside the $$ which also eased out our augmentation process of the dataset.

6.3. Model

6.3.1. Sub Task 1

This component of our model aims to produce the column that would be used in the where condition. We wasted no time in recognizing this as a problem of the Name Entity Recognition or NER. For implementing it, we tried to closely match the tokens of the natural language query against all the words present in our corpus. This sub-task was a success as it could also give correct answer to natural language queries which would have two or more columns in the "WHERE" clause.

6.3.2. Sub Task 2

This unit produces the columns that we need for the SQL query of the corresponding natural language query. Firstly, we made BERT-word Embeddings for our dataset using bert-as-service(Xiao, 2018). We used these to train a Random Forest to produce the columns but the accuracy as it can ve considered a task of classification. But the accuracy was low. XGBoost model also gave low accuracy.

We know tried to use heuristic based POS-tagging model for preicting columns as columns could only be used as common nouns for the ACL Dataset(Singh et al., 2018b). Hence we used spacy's model to create dependency tree which would give us common nouns which then passed to difflib's get_close_match() function gives the required solution. To improve accuracy we used Levenshtein distance, and thus a quite an correct model was developed.

Models	Accuracy
Random Forest	57
XGBoost	65
POS Tagging with close match	80
POS Tagging with Lev. Dsitance	98

Table 3: Different Trials for subtask 2

6.3.3. Sub Task 3

For this sub-task we needed to give output the link of tables we need for the SQL query of the corresponding natural language query, so we designed one of its kind model of treating the schema of ACL Database as a graph for our model. This technique is so good that it could give correct answer for the more than three tables. The tables were linked using "JOIN" clause applied on Foreign and Primary keys of the tables. The accuracy of this sub-task was 95%.

6.3.4. Sub Task 4

This component of the model produces the 'COUNT' clause for the SQL Query. It also takes care of the condition to be used in the 'WHERE' clause if any.

For taking care of 'COUNT' clause, we first used the Random Forest on the Natural Language Queries. The Random Forest didn't work because dataset has high number of queries without 'COUNT'. This problem which can be treated as a simple binary classifier, could not be solved using Random Forests because of the high bias. The next technique used was XGBoost whose answer was highly dependent on 'train_test_split' of the scikit-learn library. The accuracy achieved was around 75%.

To improve the accuracy of this simple problem, we thought that intent extraction could be great as it is heuristic model which would not be affected by the bias of our dataset. We used the synsets of WordNet(Miller, 1995) present in nltk library for taking out different meanings

of the tokens. Then we used Lesk algorithm to match different meanings of tokens with 'count' or 'number' to check whether the token represents quantification, hence the use of 'COUNT' in the SQL query. But due to our surprise this method didn't even surpass the accuracy of the Random Forest model.

We used the bigram model of language modelling to predict 'COUNT'. The accuracy of this model was by far the highest. For the final trial, we used pre-trained BERT word embeddings by google. We fine-tuned the BERT embeddings on our dataset by running 10 epochs for training, and it produced extra-ordinary results on test data, achieving an accuracy of 99%.

The dataset is highly biased towards having "==" in "WHERE" clause or not having the clause itself. Hence, we knew from the above experience that Random Forests and XGBoost models won't work. So we directly applied fine-tuning of BERT-Embeddings on the dataset producing an accuracy of 89% on test data.

Models	Accuracy (in %)
Random Forest	68
Intent Extraction	54
XGBoost	75
Bigram Model	94
Bert Fine-Tuned	98

Table 4: Different Trials for 'COUNT' prediction

7. Evaluation Metric

7.1. Exact Match

The percentage of test set in which the predicted SQL query is same as the actual SQL query.

$$Acc_{ex} = \frac{N_{if}}{N}$$

where
N : Total examples in the dataset
Nif: Number of queries with exact string match with ground truth query

7.2. Semantic Match

The percentage of test set in which the predicted SQL query is semantically same as the actual SQL query. If the generated output of both the predicted and actual are the same even though the query are not the same then we call it to be semantically same.

$$Acc_{ex} = \frac{N_{ex}}{N}$$

Nex: Number of queries obtaining correct result when executed

7.3. Query Distance

The exact match and the semantic match accuracy reveal only the percentage of exacts (i.e) binary. It does not tell us how close is our predicted. Here we have defined a new evaluation metrics which allows us to comment on the closeness of match between actual and predicted SQL queries.
Query Distance of a query is the distance between the predicted select and where columns and the actual select and where column. The distance is number of edges between the give columns in the BFS Tree of the Schema Graph.QD=1 if the actual and predicted columns are same and QD¿1 if there is a mismatch. QD can take a max value of 3.

$$SELDist = dist(SEL\_COL_{pred}, SEL\_COL_{act})$$

$$WHRDist = dist(WHR\_COL_{pred}, WHR\_COL_{act})$$

$$QD_i = \frac{1}{\sqrt{2}} * \sqrt{SELDist^2 + WHRDist^2}$$

$$QD = (\prod_{i=1}^{N} QD_i)^{\frac{1}{N}})$$

8. Results and Conclusion

	QD	Exact Match	Semantic Match
Model 1	1.142	86.67%	95 %
Model 2	1.132	86.67%	96.67%

Table 5: Performance of the models on test set

• We are able to generate the SQL queries quite accurately for multi-table databases in comparison to those generated by various existing architectures which are mostly deep-learning based and require heavy training.
• With the accuracy achieved, our model can easily be implemented in real-world scenarios involving database querying for low-level and medium-level of queries.
• In comparison to the existing architectures, our approach is rather more customized for a specific database and if properly fine-tuned will be able to out-perform all the existing architectures for that particular database.
• The above holds true because of the fact that most of the sub-tasks in our problem are resolved with the help of algorithms which have been designed using the knowledge of the structure of natural language sentences.
• However, unlike the existing architectures our approach cannot be directly implemented for a large types of databases as it requires fine tuning for them. However, if the tasks such as the fine-tuning of the classifier, the creation of the database schema if automated then there are high chances that our architecture can outperform the existing ones.
• Also, the dataset generated by us has queries ranging upto 5 table linkages, will be helpful for the other works around the ACL-Anthology database as all such existing datasets are mostly for single tables.

9. Future Works

• Our current model predicts queries with upto five different conditional operators which includes $<, >, >=, <=, ==$. So, we will be working on integrating other conditional operators such as 'AND', 'LIKE', 'HAVING'.
• Various parameters tuning such as database structure retrieval, which would help our model to be easy to implement on databases with varying schemas.
• We need to work on the named entity recognition accuracy and need to develop a flexible architecture which will help to adapt to new databases contents.

10. References

Dong, L. and Lapata, M. (2018). Coarse-to-fine decoding for neural semantic parsing. *arXiv preprint arXiv:1805.04793*.

Miller, G. A. (1995). Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, November.

Price, P. J. (1990). Evaluation of spoken language systems: The atis domain. In *Proceedings of the Workshop on Speech and Natural Language*, HLT '90, pages 91–95, Stroudsburg, PA, USA. Association for Computational Linguistics.

Singh, M., Dogga, P., Patro, S., Barnwal, D., Dutt, R., Haldar, R., Goyal, P., and Mukherjee, A. (2018a). Cl scholar: The acl anthology knowledge graph miner. *arXiv preprint arXiv:1804.05514*.

Singh, M., Dogga, P., Patro, S., Barnwal, D., Dutt, R., Haldar, R., Goyal, P., and Mukherjee, A. (2018b). CL scholar: The ACL anthology knowledge graph miner. *CoRR*, abs/1804.05514.

Tang, L. R. and Mooney, R. J. (2001). Using multiple clause constructors in inductive logic programming for semantic parsing. In *Proceedings of the 12th European Conference on Machine Learning*, ECML'01, pages 466–477, Berlin, Heidelberg. Springer-Verlag.

Wang, P., Shi, T., and Reddy, C. K. (2019). A translate-edit model for natural language question to sql query generation on multi-relational healthcare data. *arXiv preprint arXiv:1908.01839*.

Xiao, H. (2018). bert-as-service. `https://github.com/hanxiao/bert-as-service`.

Yu, T., Yasunaga, M., Yang, K., Zhang, R., Wang, D., Li, Z., and Radev, D. (2018a). Syntaxsqlnet: Syntax tree networks for complex and cross-domaintext-to-sql task. *arXiv preprint arXiv:1810.05237*.

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., Zhang, Z., and Radev, D. R. (2018b). Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *EMNLP*.

Zhong, V., Xiong, C., and Socher, R. (2017). Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.