BERT Model-based Natural Language to NoSQL Query Conversion using Deep Learning Approach

Kazi Mojammel Hossen[1], Mohammed Nasir Uddin[2], Minhazul Arefin[3], Md Ashraf Uddin[4]
Department of CSE, Jagannath University
Dhaka, Bangladesh[1,2,3,4]

***Abstract*—Databases are commonly used to store complex and distinct information. With the advancement of the database system, non-relational databases have been used to store a vast amount of data as traditional databases are not sufficient for making queries on a wide range of massive data. However, storing data in a database is always challenging for non-expert users. We propose a conversion technique that enables non-expert users to access and filter data as close to human language as possible from the NoSQL database. Researchers have already explored a variety of technologies in order to develop more precise conversion procedures. This paper proposed a generic NoSQL query conversion learning method to generate a Non-Structured Query Language from natural language. The proposed system includes natural language processing-based text preprocessing and the Levenshtein distance algorithm to extract the collection and attributes if there were any spelling errors. The analysis of the result shows that our suggested approach is more efficient and accurate than other state-of-the-art methods in terms of bilingual understudy scoring with the WikiSQL dataset. Additionally, the proposed method outperforms the existing approaches because our method utilizes a bidirectional encoder representation from a transformer multi-text classifier. The classifier process extracts database operations that might increase the accuracy. The model achieves state-of-the-art performance on WikiSQL, obtaining 88.76% average accuracy.**

Keywords*—*Natural language processing; NoSQL query; BERT model; Levenshtein distance algorithm; artificial neural network

I. Introduction

In today's digital age, non-relational databases are utilized in almost every industry to store information. Non-Structured Query Language (NoSQL) databases [1], [2] are increasingly being used for large-scale data sets, search engines, and real-time web applications [3]. Nowadays, NoSQL databases work as an alternative to relational databases [4] and other conventional databases [5].

With the growth of technology, NoSQL databases stores a large amount of data in document stores, key-value data stores, wide-column stores, and Graph stores. As opposed to relational databases, MongoDB, CouchDB, Cassandra, etc are designed on the architecture of distributed systems to store massive date [6]. Many organizations are gradually looking into approaches to understand and analyze this enormous unstructured data. The current approaches to data management, organization, and storage are being changed by "Big Data" [7]. In particular, "Big Data," an open source framework used to store vast amounts of structured, unstructured, and semi-structured data [8]. So, Normal users require knowledge of the query syntax and table schema to access and store a large amount of data. However, finding a reliable approach to generate the NoSQL query from Natural Language (English) is challenging. Using NoSQL approach, amateur users can interact with the database system. The model facilitates communication between humans and computers without recalling the query syntax method for the non-relational databases. Natural Language Processing (NLP) [9], [10], [11] is a branch of linguistics, information engineering, computer science, and artificial intelligence that studies how computers and humans interact with Natural Language [12]. Traditional machine translation is applied to translate the text from one language to another by NLP [13].

This research aims to develop a feasible tool for searching databases where natural language can be used without needing complex database queries that are developed by expertise. Generating NoSQL from natural language has wide range of applications. Tools with AI knowledge [14] such as Google Assistant or Alexa use the NLIDB system for non-technical users. Filling out a lengthy online form can be tedious and users might need to navigate through the screen, scroll, look up values in the scroll box, and so on whereas with NLIDB, the users need to type a question similar to a sentence. Consequently, such a tool has a wide range of usage and applications. NoSQL approach has been researched both in academia as well as in industry [15]. In this paper, we implement a Neural Machine translation model which consists of four steps. First, we have used a Natural Language Tool-Kit for performing text preprocessing. Secondly, attributes are collected and extracted using Levenshtein Distance (LD) [16], [17] algorithm. Thirdly, we have used a bidirectional encoder representations from BERT Transformers Model-based multi-text classification [18] to extract the operations including find, insert, update and remove. The last step of the proposed approach is generating query.

Many research works have used WIKISQL dataset for conversation Natural Language to Structured Query Language. The BERT Model generates the NoSQL operational command from the WIKISQL task. The contribution of this research paper can be summarized as follows:

- Designing several algorithms to come up with a standard machine translation model for converting Natural Language into NoSQL queries.
- To resolve the syntax errors for primitive users using Levenshtein Distance algorithm that can extract the collection and attributes from the text even if any users make spelling mistakes or utilize synonyms.
- To employ the latest contextual word representation BERT transformer model to extract the operations with a higher accuracy rate.

The remainder of the paper is organized as follows: Related works conducting with the same and different technologies by other researchers are illustrates in Section II. Section III describes the proposed methodology and work flow. Section IV shows experiment evaluation and result of the proposed system. Conclusions with the future expansion are detailed in Section V.

II. Related Work

Research in Natural Language for non-relational databases has started as far back as the twenty century. Since the interest in Natural Language Processing has continued tremendously. In the early 1970's LUNAR [19], the first Natural Language Interface for the relational database (NLIDB) has introduced to the researcher. LUNAR was a Question Answering (QA) system connected with the moon rock sample database. The information of rock samples brought back from the moon was used to make the LUNAR database. NLP to NoSQL query conversion field has very little research on it. This section discusses various works on Natural Language to query conversion.

In 2021, Minhazul et al. [20] suggested a machine learning-based NLP2SQL translation system. They used the Naive Bayes algorithm for command extraction and decision tree regression for condition extraction. Their proposed method lack accuracy because of using the bag of words technique in the derivation of condition from SQL. An advance deep learning solution can mitigate this problem. On the other hand, they can use the neural translation technique for this machine translation approach. The system can use the statistical translation method also.

Mallikarjun et al. [21] proposed an automated NLP-based text processing approach. Their approach can successfully convert an excel datasheet into a DBMS. Their system has a user authentication system that prevents unwanted users. The system has a limitation of 16,384 columns and 1,048,576 rows for an excel worksheet. This data may be massive for average purposes but not enough for big data.

An Intelligent processing system in a document-based NoSQL database had proposed by Benymol et al. [22] in 2021. They used state-of-the-art algorithms and technologies to convert text into NoSQL. They used different types of TF-IDF schemes for information retrieval, machine learning algorithm for modeling, and hyper parameter tuning for model selection. The system may have vulnerability in stream and batch data on the Big Data processing platform. The proposed model also has a problem with dynamic processing strategies. In this stage, the system fails to find any possible solution.

Fatma et al. [23] proposed an automatic UML/OCL model for the NoSQL database converter. Their system mainly focuses on the big data platform. Because there is wide use of NoSQL database in the big data platform. After creating the NoSQL database, the system automatically checks the OCL constraints of the model. There are different types of NoSQL databases and a maximum of them have a problem with integrity constraint checking. For this, it is the most challenging task in the system.

In [24] M. T. Majeed et al. have designed a fully automated framework that, using an AI technique, can recognize keywords, symbols, attributes, values, and relationships among various types of quiries. Additionally, they proposed a graphical user interface where users could enter NL queries and have a NoSQL query created from those queries. For complex queries, the proposed framework didn't offer a solution.

S. Mondal et al. [25] introduced a query-response model that can respond to a variety of queries, including assertive, interrogative, imperative, compound, and complicated forms. This NoSQL system's primary task is to retrieve knowledge data from the default MongoDB database. This paper didn't give any solution of time-related query such as "What is the age of x after 10 years".

T. Pradeep et al. [26] presented a Deep Learning based approach that converts English questions to MongoDB queries. They applied an encoder-Decoder machine-translation method for this conversion. The encoder turns the NLQ text input into a vector and sends it to the decoder. The decoder uses a deep neural network to predict NoSQL queries. Their system uses ten different deep learning models to handle ten types of MongoDB queries. One solution is the best possible answer for this problem.

Sebastian Blank et al. [27] suggested an end-to-end Question Answering (QA) system. It allows a user to ask a question in natural language on the Elasticsearch database. They solve the homogeneous operation problem of the database by using policy-based reinforcement learning. For that, they used Facebook's bAbI Movie Dialog dataset. They also design a KBQueryBot, an agent of translating a natural language query into the domain-specific query language based on a sequence-to-sequence model [28]. It gives every single answer with the help of an external knowledge base.

Some classic NLIDB systems can solve the spelling corrections of misspelled words automatically [29]. The module gives the interface between computer and user by the database query language. Consequently, they discuss the overall system architecture of the NLIDB, some implementation details, and experimental results. The proposed work only focuses on automatic spelling and grammar correction.

Z. Farooqui et al. [30] recommended the conversion of English to SQL. For example, their system converts English questions or text queries into SQL queries. Later it will be operated on databases. Their suggested technique and method are generic and smooth. It can handle both small and large applications for generic NLIDB systems. There are four types of input NLQ text Normal, Linear Disjoint, Linear Coincident, and Non-Linear Model. It focuses on simple SQL query clauses such as SELECT, FROM, WHERE, and JOIN. Their system can handle complex queries resulting from ambiguous NL queries.

Tanzim Mahmud et al. [31] proposed a system based on Context-Free-Grammar (CFG). Any input token of appropriate terminals found in the input NLQ will replace the corresponding attribute in the relational table or applicable operators of SQL. The interface can configure easily and automatically by the user. It relies on the Metadata set and Semantic sets for tables and attributes. It can handle ambiguities in the input NLQ. For example, the system can solve the same attribute name clashing problem within a table. The limitation of the proposed CFG system can only convert limited queries. Other than that, the system is dependent on a specific language.

Xiaojun Xu et al. proposed SQLNet [32], an NLP to SQL conversion approach which is order-independent and alternative to the traditional sketch based program synthesis approaches [33]. Failing an order input NLQ text is not a problem in that case. It uses a sequence–to-set model, which is a column attention mechanism that generates SQL queries. It represents pseudo-tasks with the help of a function of relevance and works on the WikiSQL task.

Victor Zhong et al. [34] thought about the availability of the query ground truth (intermediate labels) and database response. They proposed Seq2SQL, a modular approach that translates NLQ into SQL queries. Their suggested system also generalizes across different table schemas. There are three modules in Seq2SQL. The first module tries to identify an aggregator function like MIN() or MAX(). The second module extracts column names from NLQ and uses them as a select operator. Both modules worked on question-answer pairs. The third module extracts condition or where-clause from NLQ. There is a possibility to swap between arguments in the WHERE clause. This ambiguity problem could solve by policy-based reinforcement learning [35] in question-answer pairs.

III. Proposed Methodology

The main concept behind this method is to transform Natural Language (NL) into Non-Structured Query Language (NSQL) using Natural Language Tool Kit (NLTK) and Deep Learning Model. The concept and its description are formalized in the following sections. The proposed architecture is shown in Fig. 1.

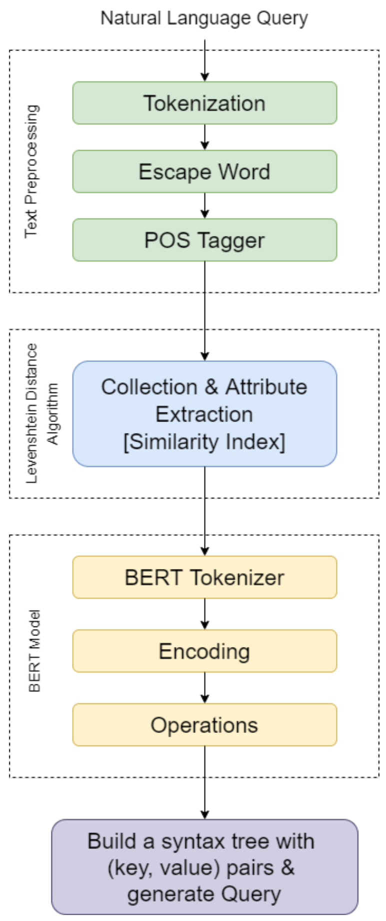
Natural Language Query
Text Preprocessing
Tokenization
Escape Word
POS Tagger
Levenshtein Distance Algorithm
Collection & Attribute Extraction [Similarity Index]
BERT Model
BERT Tokenizer
Encoding
Operations
Build a syntax tree with (key, value) pairs & generate Query

Fig. 1. Proposed methodology

A. Input Natural Language Query (NLQ)

NLQ consists of only the normal terms of user's language, without any special format or syntax. Natural language query (NLQ) in English is given as input. This input text will be processed for getting information and later converted into NoSQL queries.

1) Text preprocessing: Since the inventory of individual words, text can take many forms, ranging from sentences to many paragraphs with special letters. In NLP the text preprocessing is an important task and the first step in the preprocessing to building a model. It is a data mining technique that transforms plain text into a machine-readable format. Real-world data is frequently inadequate, inconsistent, or deficient in specific behaviors and is likely to contain various errors. This step is needed for transferring input text from human language to machine-readable format for further processing.

In this paper, we have used NLTK for text preprocessing. The NLTK is the most widely used and well-known of the NLP libraries in the Python ecosystem. It is used for all sorts of tasks from lowercase conversion to tokenization, removing escape words, part of speech tagging, and beyond. Input text will be processed for getting information from Natural language Query input. From the processed text, the system will extract collection, attribute, and operation for making a NoSQL query.

2) Lowercase conversion: Lowercase conversion is the first step of text preprocessing. In this step, the input NLQ is converted into a lower case format. Although the uppercase or lowercase forms of words are supposed to have no difference, all the uppercase characters usually changed into lowercase forms before the classification.

3) Tokenization: Tokenization splits the natural language query, phrase, string, or entire text document into smaller units such as individual words or tokens. The former Sentence Boundary Disambiguation (SBD) is often used to form a list of individual sentences. It depends on a pre-trained, language-specific algorithm similar to the Punkt Models from NLTK. The text divides into a list of words using an unsupervised algorithm to form a model for abbreviated words. For the English language, a pre-trained Punkt tokenizer includes in the NLTK data package.

4) Removing escape words: Escape or extra words are the words that are frequently appeared within the text without having more information or content. So, the escape words are removed because they are not needed in the analysis of the query. For the purpose of building queries, several sets of escape words have been developed. In this paper, we proposed a new set of escape words. Auxiliary verbs and prepositions are mainly used in this context as escape words such as 'a', 'an', 'the', 'is', 'of', 'with', 'to', 'for', 'and', 'all', etc.

TABLE I. ESCAPE WORDS

Escape Words	With Escape Words	Without Escape Words
all, the	Find all the students	Find students
is, the, of, all	What is the name of all student?	What name student
the, and	Insert the student name x and age 20	Insert student name x age 20

Beside Table I, this step eliminates punctuation from the input natural query. The detailed process of eliminating escape words is illustrated in Algorithm 1.

```
Algorithm 1: Removing Extra Words
Input: I = Input words and; E = List of Extra Words
Output: L: List of words after removing extra words
$c_w$ = CountWord($I$)
for $c_w \in C_W$ do
    $I$ = I[c] TOKENS = Tokenization($I$)
    for $l \in TOKENS$ do
        TOKEN = EMPTY
        if $l \notin E$ then
            PUSH($TOKEN, t$)
        end
    end
    PUSH($L, TOKEN$)
end
return L
```

Removing escape words is a simple but essential aspect of many text mining applications cause it reduces memory overhead. It can reduce noise and false positives. This method can potentially improve the power of prediction in any text mining application.

5) Parts of Speech (PoS) tagger: PoS tagging helps in text-to-speech conversion, information retrieval, and word sense disambiguation. It's used for the classification of words in their PoSs and labeling them according to the tagset. The collection of tags used for PoS tagging is tagset. PoS tagging is also referred to as word classes or lexical categories. However, all PoS tags aren't necessary to analyze. All PoS tagging attributes are provided by the NLTK toolkit. The PoSs must be defined as the following:

- Noun Tags = ['NN', 'NNS', 'NNP', 'NNPS']
- Adjective Tags = ['JJ', 'JJS', 'JJR']
- Verb Tags = ['VB', 'VBP', 'VBD', 'VBG', 'VBZ']
- Adverb Tags = ['RB', 'RBR', 'RBS']

Adverb and adjective tags do not have much significance in generating NoSQL queries. Only noun and verb tags are considered for the next steps of PoS tagging. Because verb & noun tags may indicate command and attributes or table name respectively. Algorithm 2 illustrates the process.

```
Algorithm 2: Keeping Necessary Tags
Input: W = All the words after removing stop words;
Output: T = Necessary Tag's with appropriate word
$c$ = CountWord($W$)
for $c \in C$ do
    $w$ = W[c] TAGS = Tagging($w$)
    for $t \in TAGS$ do
        TAG = EMPTY
        if $t \in VERB$ then
            PUSH($TAG, t$)
        end
        if $t \in NOUN$ then
            PUSH($TAG, t$)
        end
    end
    PUSH(T, TAG)
end
return T
```

B. Collection and Attribute Extraction

Levenshtein distance (LD) algorithm is used in a specific solution to extract collections and attributes from natural language queries. The approach starts by counting how many words in the list are similar to one another. Afterward, it compares every single similar word with every attribute from the WordNet by the LD and synonym list plays a crucial role in extracting attribute and collection names. This method keeps a list of words that are synonyms for each noun tag. Using WordNet, a list of noun tag synonyms is generated. The aim of making a synonym list is to find a specific collection and attribute from an input query. Every user formulates their query in a different way. They also use different words to describe the attribute or collection names. So, this approach checks synonyms of the words from the user query in the WordNet library. we give some analogies in Table II:

TABLE II. ANALOGY BETWEEN TEXT AND INTRUSION DETECTION WHEN APPLYING THE LD ALGORITHM

Text	Intrusion Detection
Find the name of all **student**	Collection(**all_student**)
What is the **accommodation** of student id 01	Attribute(**address**)
Find the **Fastname** of the students	Attribute(**name**)

In this paper, the LD algorithm works as a threshold. Sentence word is compared with the collection name if the value is greater than the threshold then it saves the collection and attributes with the appropriate name in a list. Finally, this approach gives an output figure of the match collection and attribute. The designed algorithm for collection and attribute extraction is described in Algorithm 3

Levenshtein Distance formula is used to measure the distance between the two strings a and b with length $|a|$ and $|b|$, respectively.

$$LD_{a,b}(m,n) = \begin{cases} max(m,n) \\ min \begin{cases} LD_{a,b}(m-1,n)+1 \\ LD_{a,b}(m,n-1)+1 \\ LD_{a,b}(m-1,n-1)+1_{(a_m \neq b_n)} \end{cases} \end{cases}$$

Here $(a_m \neq b_n)$ is the indicator function that is equal to 0 when $(a_m \neq b_n)$, otherwise 1, and $LD_{a,b}(m,n)$ is the distance

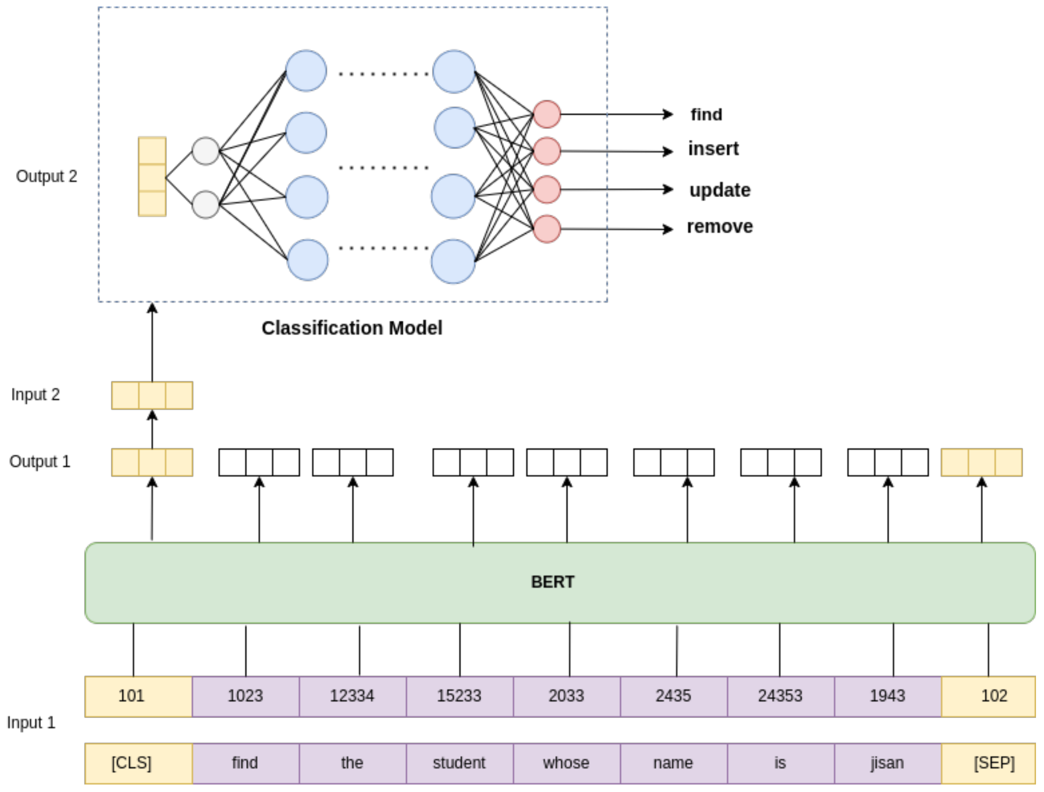

Fig. 2. Operation extraction using BERT model

```
Algorithm 3: Attribute Extraction
  Input: W = List of Attributes from Database; C = List of Collection name from Database; S = Set of Similar Words
  Output: A = Attributes Name; B = Table Name
  $t$ = CountWord($T$)
  for $i \leftarrow 1$ to $t$ do
      for $j \in S$ do
          $LD-THRESHOLD = 1$
          THRESHOLD = LD-Algorithm($S[j], W[j]$)
          if $LD-THRESHOLD > THRESHOLD$ then
              PUSH($A[i], W[j]$)
              PUSH($B[i], C[i]$)
          end
      end
  end
```

between the first m characters of a and the first n characters of b.

C. Operation Extraction

Operation extraction is a particular solution that uses BERT Model to extract operations from natural language queries. In this approach, we use BERT Model for classifying the specific operation. In machine learning, classification is the set of categories that analysis belongs to the basis of a training set of data containing (or instances) whose categorical membership is known [36]. A classification model tries to make some inferences from the observed data. To predict one or more outcomes from the dataset, provide one or more data as inputs to the categorization model.

In the dataset, BERT employs a novel technique known as Masked Language Model (MLM), in which it masks words in the sentence at random and then attempts to predict them. It doesn't use common sequence left-to-right or right-to-left language models. Instead, it uses the bidirectionally trained sequence with a deeper sense of language context and the model. The pre-train BERT applying two unsupervised tasks:

- Pre-training the BERT to understand language.
- Fine-tuning the BERT to learn specific task.

BERT depends on a Transformer (the self-attention mechanism to learns contextual relationships between words in a text). A simple Transformer consists of an encoder that reads text input and a decoder to generates a task prediction. Since the BERT model only requires the encoder part for generating a language representation model. There are two main models of BERT:

- **BERT base** has 12 transformer blocks, 768 hidden layers, 12 attention heads, and 110M parameters.
- **BERT large** has 24 transformer blocks, 1024 hidden layers, 16 attention heads, and 340M parameters.

In this paper, we used the BERT base model that has enough pre-trained data to help bridge the gap in data. The model for operation extraction shows in Fig. 2. Given the input text, the Model that tokenizes the text using BERT tokenizer then generates the input masks with input IDs of the sentence. The input mask uses WordPiece [37] for tokenizing that splits the token like “going” to “go” and “ing.” It is mainly to cover a broad spectrum of Out-Of-Vocabulary (OOV) words. After tokenization, the output class goes as input in the classification model. we used a neural network for classification to get the highest accuracy. After classifying, we get the output of the operation. Here we work on four types of operations, in consideration- **FIND, INSERT, UPDATE, REMOVE**.

D. Build Syntax Tree & Generate Query

After Tokenization, collection, attribute, and operation are extracted from the sentence, we map the syntax tree with key-value pairs to build the query sequentially with the logical expression. If there are no logical expression in the sentence, it will be Nulled. Fig. 3 shows the syntax tree.

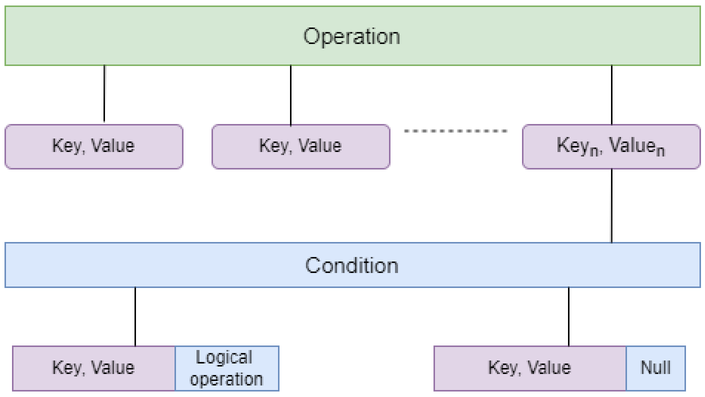

Fig. 3. Mapping syntax tree

Finally, we concatenate the whole step part-by-part and generate a NoSQL query. Fig. 4 shows the architecture of NoSQL query and given the output of the result.

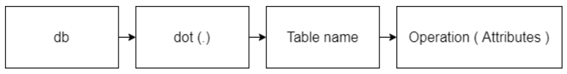

Fig. 4. Architecture of query generation

IV. Experimental Analysis and Result

In this section, we evaluate our proposed model with the dataset. Firstly, we present the analysis of our dataset, then set up the evaluation. In the end, we compare our proposed model with the existing works and mention the differences, weak and strong points of our proposed model.

A. Dataset

We reshuffle the WIKISQL dataset for a better understanding of our model performance. WIKISQL is a massive crowd sourced dataset for creating NLIDB. The model is retrained periodically by reflecting the latest dataset. Our proposed model has used two types of data: (1) Natural Language Query column which represent the natural language query and (2) Operations column. The description of the datasets is illustrated in Table III.

TABLE III. Description of Dataset

Dataset	WIKISQL
Language	NLQ
Total number of cases	80,654
Length of the text (average)	61.09
Word count of the text (average)	11.66
Granualarity of text description	line
Number of validation text	8,421
Number of test cases (total)	15,878
Number of train cases (total)	56,355

To avoid overfitting, we split the dataset into the training set and the testing set. we train our model on 70:30, 60:40, and 80:20 ratios and get the optimal result from the 80:20 ratio on our dataset. The data fields are the same among all splits. WikiSQL is a collection of hand-annotated SQL table, question, and query examples from Amazon Mechanical Turk crowd workers. It is orders of magnitude larger than current datasets, with 87000 samples as of this writing. The number of validation queries is 8,421. We build queries for the table and then ask crowd workers to paraphrase them. Each paraphrase is then double-checked by independent personnel to ensure that it does not alter the meaning of the original inquiry. We anticipate that making WikiSQL available will aid the community in developing the next generation of natural language interfaces (Fig. 5).

The Fig. 6 illustrates a blue histogram which shows the word and text distribution of dataset. It is hand-annotated semantic parsing dataset that contains logical and normal forms, respectively. In the dataset, the data is extracted from the web.

B. Text Pre-Processing

Text pre-processing is the first step of our proposed system. This step involves removing noise from our dataset. we apply several pre-processing steps to the data to convert words into numerical features. An example of tokenization is:

Input: ‘find the name of all student’
Output: [‘find’, ‘the’, ‘name’, ‘of’, ‘all’, ‘student’]

C. Collection and Attribute Extraction from WordNet

We used NLTK WordNet to find find synonyms and antonyms of words. A WordNet is a lexical database that contains semantic relationships between words and their meanings. Our proposed model can successfully extract collection and attributes from WordNet library if there were any spelling errors occur or synonyms used. The bar diagram 5 shows how extract collection and attribute from WordNet using Levenshtein distance. For example:

Collection extraction: ‘all_student’:[‘student’,‘students’]
Attribute extraction: ‘name’: [‘name’, ‘title’, ‘label’]

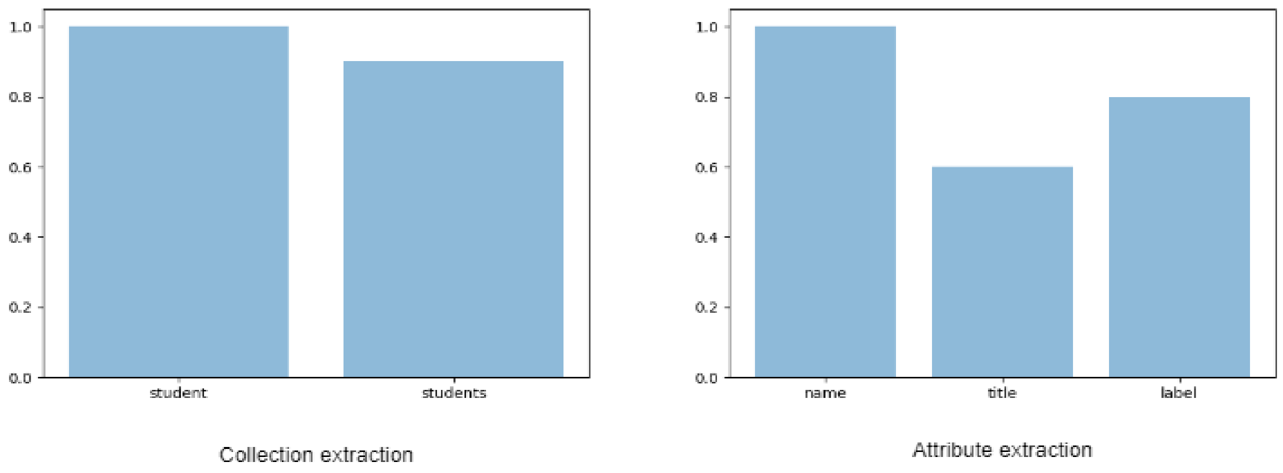

Fig. 5. Collection and attribute extraction

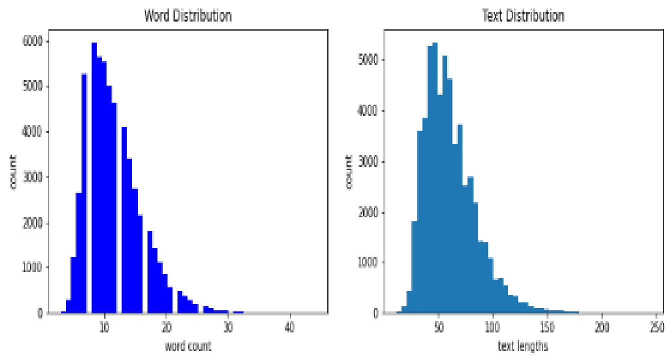

Fig. 6. Word and text distribution

D. BERT Tokenizer

In operation extraction our proposed system starts with BERT Tokenizer step. It gives sinusoidal positional encoding, the model itself learns the positional embedding during the training phase. Using the word-piece tokenizer concept that break some words into sub-words.

It helps many times to break unknown words into some known words and tokenize our text into tokens that correspond to BERT's vocabulary. An example of BERT Tokenization is:

Input: 'find the name of all student'
Output 1: [101, 1023, 12334, 15233, 2033, 2435, 24353, 102]
Output 2: ['[CLS]', 'find', 'the', 'name', 'of', 'all', 'student', '[SEP]']

Output 1 is indices of the input tokens from the vocab file and output 2 is the reverse, a human-readable token of the input_ids. Apart from the input tokens we also got 2 special tokens '[CLS]' and '[SEP]'. BERT model is designed in such a way that the sentence has to start with the [CLS] token and end with the [SEP] token.

E. Split Data for Training and Testing

The training phase is the first step for the BERT Model. This model is a transformer design based on an encoder stack. We trained the WIKISQL dataset using this model. The Model uses the Semi-Supervised Learning approach for translating natural language query into operation. The training sub-dataset contains all of the features required to turn a natural language query into an operational query. To partition the WIKISQL dataset into two sub-datasets, we use the scikit-learn library's "train test split" method. The suggested system is built using the dataset's training sub-dataset. The training dataset is a fraction (80%) of the whole data set. The rest (20%) is considered as test data. This information is imported as a.csv file. Table IV shows a portion of the training data set.

TABLE IV. A SAMPLE OF TRAINING DATASET

Line No.	Natural Language Query	Operations
1	What's Dorain Anneck's pick number?	find
2	Find the student whose name is x.	find
3	Insert the arrival time of greenbat.	insert
4	Put the status of the trains at location Museum	insert
5	Update the record for september 15, 1985.	update
6	Re-equip the student	update
7	Remove the brighton cast for jerry cruncher	remove
8	Delete the all student	remove

F. Model Building

BERT is an architecture that uses a transformer encoder to process each token of input text in the context of all other tokens. After splitting the dataset, we start with the pre-trained BERT Model to classify the find, insert, update and remove operations. In our model we use 12 layers of Transformer encoder. After run the operation we get two variables: First

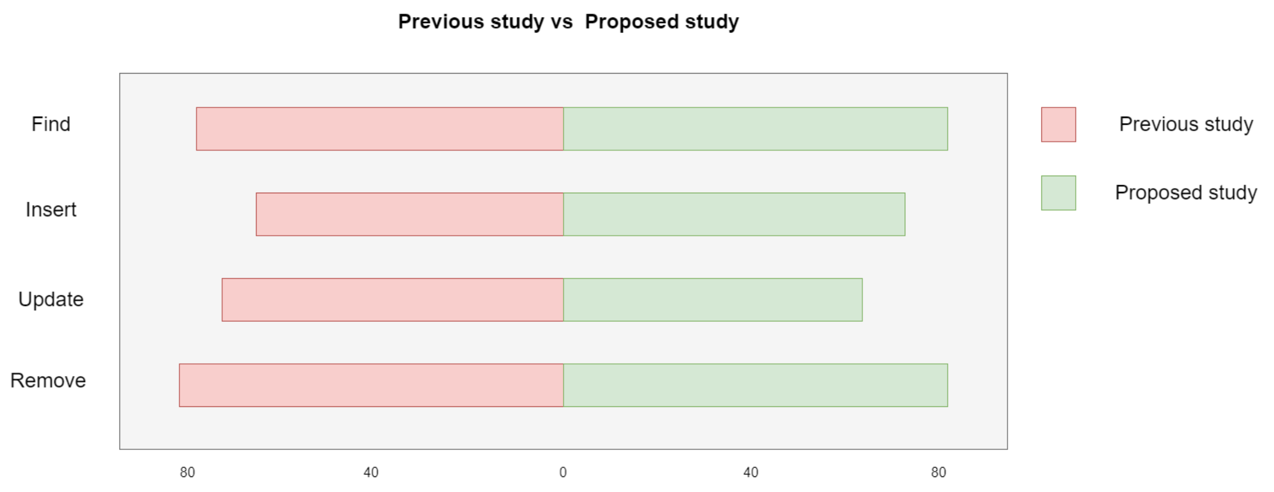

Fig. 7. Comparison of four types operations

variable contains the embedding vectors of all of the tokens in a sequence and second variable contains the embedding vector of [CLS] token. We then pass the variable into a linear layer with ReLU activation function. We have a vector of size 4 at the end of the linear layer, each of which corresponds to a category of our labels (find, insert, update, and remove). We use Adam as the optimizer and train the model for 10 epochs. Because we're dealing with multi-class classification, we'll need to use categorical cross entropy as our loss function. Fig. 8 depicted the operation.

For example:

Input: 'find the name of all student'
Output: 'find'

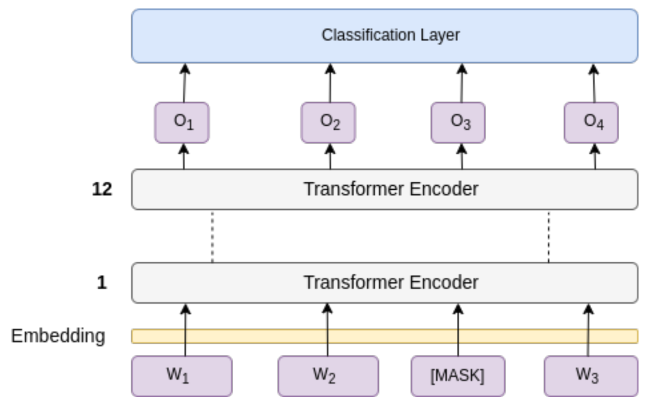

Fig. 8. Model building

The model enhances the accuracy rate for classification than the previous model. For the classification task, the model can classify 81.45% average class detection from previous research. One of the reasons is BERT uses a pre-trained model which is based on transfer learning. It can tune the data on a specific NoSQL language. Fig. 7 illustrates the accuracy rate of four types of operations separately.

G. Model Accuracy

Accuracy evaluates how well our model forecasts compare them with the original values. With a low rigor yet a high blunder, the model would make huge mistakes in the data. Both blunder and rigor lowness indicates that with most data, the model produces smaller errors. However, it produces huge mistakes in some systems if they are both high. The ideal scenario of any model would be high rigor and little blunder. Fig. 9 illustrate the accuracy of the proposed model.

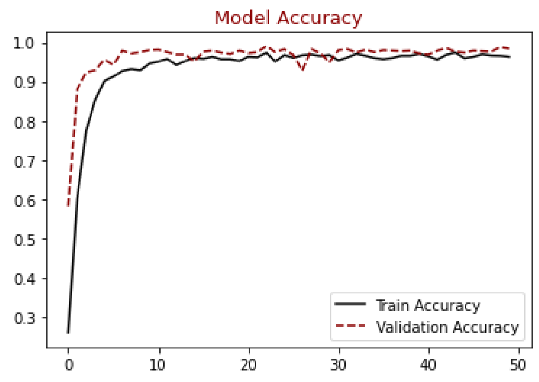

Fig. 9. Model accuracy

H. Model Loss

Loss is the total of our model errors. It evaluates how well our model does (or how badly it does). When there are a lot of mistakes, the loss is high and the model doesn't work properly. The better our model works, the lower it is. However, the greatest conclusion we can make from it is whether the loss is big or low. If we plot losses over time, we can evaluate if and how quickly our model is learning. This is because the loss function is utilized by the model for learning. This takes the shape of approaches like gradient descent, which modify parameters of the model using information on the loss outcome. Fig. 10 illustrate the loss of the proposed model.

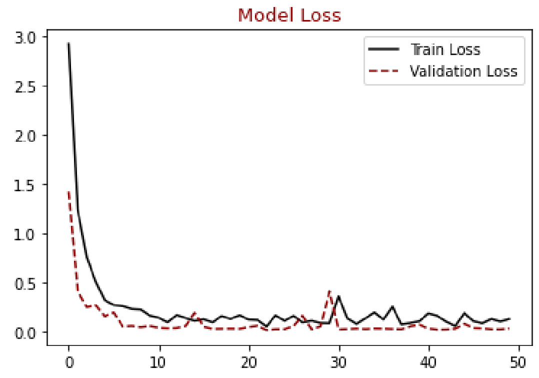

Fig. 10. Model loss

I. Output

In output, we get the collection and attribute name, such as **all_student** and **name**. From the operation extraction, get the **find** operation, then concatenate all the extractions output part-by-part to generate a NoSQL query. For example:

```
db.all_student.find({}, {'name':True})
```

We have classified the wrong output into two categories: (a)sometimes, the query contained incomplete logical expression in condition part (b) the query is incorrect. Analysis of the conversion results reveals the following:

- Observing all the NoSQL output, we can notice suggested model can work with natural language queries of different lengths. After a successful NoSQL query output, the number of input and output tokens might be distinct. The accuracy of the proposed model did not depend on the length of the query.
- The BERT Model successfully predicts the operation using a pre-trained model. It also tunes the NoSQL command from a distinctive size of input text.
- The BERT model can process a large amount of data. The WIKISQL dataset covered different types of query statements. So there is no problem for the BERT model to work with the WIKISQL dataset.
- The Bert model understands the semantic relationship between natural language and NoSQL queries. As a result, the decoder output is logically correct for the maximum query.
- The model can generate “contextualized” word embeddings but it is compute-intensive at inference time and need to calculate compute vectors every time.
- In collection and attribute extraction, we use the Levenshtein Distance algorithm. The algorithm can extract attributes from natural language queries furthermore check the spelling error. The run time complexity of this algorithm is lower than $O(n^2)$.

Test results show in Table V that have been translated into the NoSQL syntax. The test data contains the natural language query as well as appropriate. The output contains each converted NoSQL query with original query and test query, along with the percentage of converted NoSQL query.

J. Evaluation Setup

In this dissertation, we evaluate the result on our dataset that have three notation to evaluate the query synthesis accuracy.

- **Normal form accuracy** is the form of a NoSQL query that has no attribute. We analyze the synthesized NoSQL query with the ground truth to verify whether they match each other.
- **Logical form accuracy** is the accuracy of a NoSQL query that has attributes or any logical expression of the query.
- **Query match** is the comparison accuracy with the original query match for find, insert, update and remove operations query. We use a canonical representation of the synthesized NoSQL query and the ground truth to determine whether two NoSQL queries are identical.

We also find out the F1 score for operation extraction that measures the precision and recall value. Finally, we present the comparison of our model with previous work on NoSQL conversion tasks. The implementation of our model using python [38].

The F1-score measures the accuracy of the operation (find, insert, update, remove) by applying the precision and recall values of the test. This test looks at whether the system can process the sentences entered by the user so that it can be measured the operation accurately with the F1-score method. Table VI shows the accuracy values. The equation of the F1-score, precision, recall, and accuracy have given below:

- **Precision:** It is the true positive relevance rate that defined as the ratio $\frac{tp}{tp+fp}$, where fp indicates the number of false positives;
- **Recall:** It is the true positive rate that defined as the ratio $\frac{tp}{tp+fn}$, where tp and fn are the number of true positives and false negatives, respectively;
- **F1-score:** F1-score is a function of Precision and Recall that is the harmonic mean between Precision and Recall, defined the ratio as $\frac{2*(precision*recall)}{precision+recall}$;

Next, we find out the accuracy of normal and logical forms. Let X is the total number of queries in our dataset and X_ex is the execution query. we evaluate the every clause (find, insert, update and remove) query using accuracy metric for normal form Acc_nf = $\frac{X\_ex}{X}$ and for logical form Acc_lf = $\frac{X\_ex}{X}$. Table VII shows the accuracy of normal and logical queries. After that the overall result is evaluated by the BLEU (Bilingual Evaluation Understudy) that was developed to evaluate the machine translation system.

K. Result

The article presents an efficient approach to transform the natural language query into a NoSQL query effectively.

TABLE V. The Accuracy for Converting Natural Language into Non Structured Query Language

Input Text	**Original query**	**Test query**	**Accuracy(%)**
Find all the students	db.all_student.find()	db.all_student.find()	100
What is the name of all student?	db.all_student.find({name:True})	db.all_student.find({name:True})	100
Find the student whose age greater than 70	db.all_student.find({ age: { $gt: 70 })	db.all_student.find({ age: { 70 })	75.0
Insert the student whose name is x	db.all_student.insert ({name:'x'})	db.all_student.insert ({name:'x'})	100
Insert student whose name is x, age 22	db.all_student.insert ({name:'x'},{age:22})	db.all_student.insert ({name:'x'},{age:22})	83.33
Update the name y who is x in student table	db.all_student.update ({name:'x'},{$set:{name:'y'}})	db.all_student.update ({name:'x'},{$set:{name:'y'}})	100
Update name z and age 40, whoes name is x	db.all_student.update ({name:'x'},{$set:{name:'y'}})	db.all_student.update ({name:'x'},{$set:{name:'x'}})	65.5
Remove all the students	db.all_student.drop()	db.all_student.drop()	100
Delete student whose name is x and age 20	db.all_student.removeMany ({name:'x', age: 20})	db.all_student.remove ({name:'x', age: 20})	75.0

This model achieves a competitive result on our dataset. The following tables represent the experiment result of each classifier.

TABLE VI. Experimental Results of each Classifier

F1-score	0.808
Precision	0.892
Recall	0.74

Bilingual Evaluation Understudy (BLEU) is a score for comparing a candidate translation of the NoSQL query to one or more reference translations. To predict the accuracy of automatic machine translation systems, Kishore Papineni, et al. [39] proposed this score in 2002. We used the BLEU score to determine the output.

BLEU is not entirely effective but offers several interesting benefits like quick, easy to calculate, language-independent, highly interactive with human interpretation, and widely used.

$$P = \frac{m}{w_t} \tag{1}$$

where, m is the estimate of tokens from the candidate source code that are found in the reference text, and w_t is the total estimate of words in the candidate query. The accuracy is calculated using the equation 2.

$$Accuracy = P \times 100\% \tag{2}$$

The performance analysis of our model is given in Table VII.

TABLE VII. Performance Analysis of our Model. Acc$_{nf}$ and Acc$_{lf}$ Indicate the **Normal form** and **Logical form** Query Accuracy, and Acc$_{qm}$ Indicates the Accuracy of **Query Match**

Operation clause	**Acc$_{nf}$(%)**	**Acc$_{lf}$(%)**
Find	100	87.5
Insert	-	91.67
Update	-	82.75
Remove	100	75.0

Accounting to Concepts Identification errors and domain dictionary errors, the average accuracy achieved by our system is 88.76% respectively. We define the error rate as:

$$ErrorRate = (100 - AccuracyRate)\% \tag{3}$$

TABLE VIII. Analogy of Different Types of Model

Model	**Accuracy (%)**	**Error Rate (%)**
Encoder-Decoder Model	71.5	28.5
REINFORCE-algorithm Model	84.2	15.8
Proposed Model	88.76	11.24

Table VIII represents BLEU portion of efficiency for forecasting correct NoSQL query. Using the WikiSQL reshape dataset the proposed model is passed down for comparing with the existing other models. Fig. 11 illustrates the three models' estimated efficiency and error rates. It demonstrates the accuracy of other measure rates of converting the natural language query into the non-structured query language (that scored 88.76%) is better or at least competitive than the earlier results.

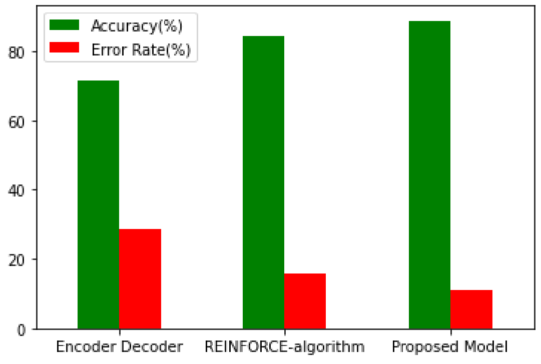

Fig. 11. Performance factor between previous and our proposed model

V. Conclusion

In the age of digitalization, internet users have been increasing continuously. So a large amount of data needs to be stored in a database. Relational databases faced some challenges in search engines and social networking services. Here, the NoSQL database helps maintain a broad range of hierarchical data models. The proposed model deals with the NLP to non-relational query conversion. Initially, preprocessing the text (English) by NLTK, then used LD algorithm for collection, attribute extraction and BERT model for operation extraction and finally, query generation. Our model can generate queries for Find, Insert, Update, Remove clause with an average accuracy of 88.76%. In the future, we intend to improve more complex NoSQL queries such as logical function queries, using other incentive mechanisms for better performance.

Acknowledgment of Funding

This work was supported by the UGC Jagannath University Research Branch, Dhaka, Bangladesh, Under JnU/research/rp/2020-21/science/44.

References

[1] J. Han, E. Haihong, G. Le, and J. Du, "Survey on nosql database," in 2011 6th international conference on pervasive computing and applications. IEEE,2011, pp. 363–366.

[2] A. Nayak, A. Poriya, and D. Poojary, "Type of nosql databases and its comparison with relational databases,"International Journal of Applied Information Systems, vol. 5, no. 4, pp. 16–19, 2013.

[3] R. S. Al Mahruqi, "Migrating web applications from sql to nosql databases," Ph.D. dissertation, Queen's University (Canada), 2020.

[4] S. Batra, C. Tyagi, "Comparative Analysis of Relational And Graph Databases", IJSCE,vol.2(2), pp. 509-512, 2012.

[5] R. Alexander, P. Rukshan, and S. Mahesan, "Natural language web interfacefor database (nlwidb),"arXiv preprint arXiv:1308.3830, 2013.

[6] Z. Wei-ping, L. Ming-xin and C. Huan, "Using MongoDB to implement textbook management system instead of MySQL", IEEE-ICCSN,2011, pp. 303-305.

[7] P. Chen, C. Xhang,"Data-intensive applications, challenges, techniques and technologies: A survey on Big Data", Information Sciences, Elsevier, vol.275, pp.314–347, 2014.

[8] B. Jose, S. Abraham, Unstructured Data Mining for Customer Relationship Management: A Survey, International Journal of Management, Technology And Engineering 8, Issue 7. ISSN NO (2018) 2249–7455.

[9] O. Ferschke, J. Daxenberger, and I. Gurevych, "A survey of nlp methods and resources for analyzing the collaborative writing process in wikipedia," in The People's Web Meets NLP. Springer, 2013, pp. 121–160.

[10] Garrido-Mu ñoz, A. Montejo-R áez, F. Marť ınez-Santiago, and L. A. Ure ña- L ópez, "A survey on bias in deep nlp," Applied Sciences, vol. 11, no. 7, p. 3184, 2021.

[11] S. Srivastava, A. Shukla, and R. Tiwari, "Machine translation: from statisticalto modern deep-learning practices,"arXiv preprint arXiv:1812.04238, 2018.

[12] Kłosowski, P. (2018). Deep learning for natural language processing and language modelling. In 2018 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA), September 2018, pp. 223-228. IEEE. 10.23919/SPA.2018.8563389

[13] U. K. Acharjee, M. Arefin, K. M. Hossen, M. N. Uddin, M. A. Uddin, and L. Islam, "Sequence-to-sequence learning-based conversion of pseudo-code to source code using neural translation approach," IEEE Access, vol. 10, pp. 26 730–26 742, 2022.

[14] B. Jose and S. Abraham, "Intelligent processing of unstructured textual data in document based nosql databases," Materials Today: Proceedings, 2021.

[15] N. Yaghmazadeh, X. Wang, and I. Dillig, "Automated migration of hierarchical data to relational tables using programming-by-example, "Proceedings ofthe VLDB Endowment, vol. 11, no. 5, pp. 580–593, 2018.

[16] S. Zhang, Y. Hu, and G. Bian, "Research on string similarity algorithm basedon levenshtein distance," in2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC).IEEE, 2017, pp.2247–2251.

[17] T. Ho, S.-R. Oh, and H. Kim, "A parallel approximate string matching under levenshtein distance on graphics processing units using warp-shuffle opera-tions,"PloS one, vol. 12, no. 10, p. e0186251, 2017.

[18] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," CoRR, vol. abs/1810.04805, 2018.

[19] Woods, W. A.,. 1973. Progress in natural language understanding: An application to LUNAR geology. AFIPS Natl. Computer. Conj: Expo.. Conference Proc. 42, 441-450.

[20] M. Arefin, K. M. Hossen, and M. N. Uddin, "Natural language query to sql conversion using machine learning approach," in 2021 3rd International Conference on Sustainable Technologies for Industry 4.0 (STI). IEEE, 2021, pp. 1–6.

[21] B. Mallikarjun, K. Annapoorneshwari, M. Yadav, L. R. Rakesh, and S. Suhaas, "Intelligent automated text processing system-an nlp based approach," in 2020 5th International Conference on Communication and Electronics Systems (IC- CES). IEEE, 2020, pp. 1026–1030.

[22] B. Jose and S. Abraham, "Intelligent processing of unstructured textual data in document based nosql databases," Materials Today: Proceedings, 2021.

[23] F. Abdelhedi, A. A. Brahim, and G. Zurfluh, "Ocl constraints checking on nosql systems through an mda-based approach," International Journal of Data Warehousing and Mining (IJDWM), vol. 17, no. 1, pp. 1–14, 2021.

[24] M. T. Majeed, M. Ahmad, and M. Khalid, "Automated xquery generation for nosql," in 2016 Sixth International Conference on Innovative Computing Technology (INTECH). IEEE, 2016, pp. 507–512.

[25] S. Mondal, P. Mukherjee, B. Chakraborty, and R. Bashar, "Natural language query to nosql generation using query-response model," in 2019 International Conference on Machine Learning and Data Engineering (iCMLDE). IEEE, 2019, pp. 85–90.

[26] T, Pradeep and P C, Rafeeque and Murali, Reena, Natural Language To NoSQL Query Conversion using Deep Learning (August 13, 2019). In proceedings of the International Conference on Systems, Energy & Environment (ICSEE) 2019, GCE Kannur, Kerala, July 2019, Available at SSRN: https://ssrn.com/abstract=3436631 or http://dx.doi.org/10.2139/ssrn.3436631

[27] S. Blank, F. Wilhelm, H.-P. Zorn, and A. Rettinger, "Querying nosql with deeplearning to answer natural language questions," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, no. 01, 2019, pp. 9416–9421.

[28] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," Advances in neural information processing systems, vol. 27, 2014.

[29] M. D. Gadekar, B. M. Jadhav, A. S. Shaikh, and R. B. Kokare, "Natural language (english) to mongodb interface, "International Journal of AdvancedResearch in9 Computer Engineering & Technology (IJARCET), vol. 4, no. 3, 2015.

[30] P. Anand and Z. Farooqui, "Rule based domain specific semantic analysis for natural language interface for database," International Journal of Computer Applications, vol. 164, no. 11, 2017.

[31] T. Mahmud, K. A. Hasan, M. Ahmed, and T. H. C. Chak, "A rule based approach for nlp based query processing," in 2015 2nd International Conference on Electrical Information and Communication Technologies (EICT), pp. 78–82, IEEE, 2015.

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

[33] J. Bornholt, E. Torlak, D. Grossman, and L. Ceze, "Optimizing synthesis with metasketches," in Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2016, pp. 775–788.

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

[35] K. Guu, P. Pasupat, E. Z. Liu, and P. Liang, "From language to programs: Bridging reinforcement learning and maximum marginal likelihood," arXiv preprint arXiv:1704.07926, 2017.

[36] G. B. Boullanger and M. Dumonal, "Search like a human: Neural machinetranslation for database search," Technical report, Tech. Rep., 2019.

[37] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun,Y. Cao, Q. Gao, K. Machereyet al., "Google's neural machine translationsystem: Bridging the gap between human and machine translation," arXivpreprint arXiv:1609.08144, 2016.

[38] M. K. Chakravarthy and S. Gowri, "Interfacing advanced nosql database with python for internet of things and big data analytics," Materials Today: Pro- ceedings, 2021.

[39] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in Proceedings of the 40th annual meeting on association for computational linguistics, pp. 311–318, Association for Computational Linguistics, 2002.