International Journal of Mathematical, Engineering and Management Sciences
Vol. 9, No. 4, 914-930, 2024
https://doi.org/10.33889/IJMEMS.2024.9.4.048

Ram Arti Publishers

Intelligent Search Engine Tool for Querying Database Systems

Gagandeep Kaur
Computer Science and Engineering Department,
Symbiosis Institute of Technology, Nagpur Campus,
Symbiosis International (Deemed University), Pune, Maharashtra, India.
Corresponding author: gagandeep.kaur@sitnagpur.siu.edu.in

Poorva Agrawal
Computer Science and Engineering Department,
Symbiosis Institute of Technology, Nagpur Campus,
Symbiosis International (Deemed University), Pune, Maharashtra, India.
E-mail: poorva.agrawal@sitnagpur.siu.edu.in

Hemlata Shelar
Senior Software Developer,
IBM Cloud, Bangalore, India.
E-mail: hemlata.shelar@sitpune.edu.in

Gebrehiwot Teklehaymanot Abraha
Computer Science and Engineering Department,
Aksum University, Aksum, Ethiopia.
E-mail: getabme2006@gmail.com

(Received on September 29, 2023; Revised on December 19, 2023 & March 4, 2024 & April 11, 2024;
Accepted on April 14, 2024)

Abstract
The amount of data being produced and used every day is increasing rapidly. Different operations are performed on the data that is stored in database systems to generate meaningful information. The task of retrieving data from the database is difficult for inexperienced users who don't have great knowledge of Database Management System (DBMS) languages such as Structured Query Language (SQL). Using Natural Language Processing, anyone can directly interact with the database system. To achieve this type of communication, we have developed a novel and improved Intelligent Search Engine (ISE) Tool using natural language processing. Using this search engine, the user can query the database system in natural language (human-understandable language). The intelligent search engine will convert the natural language query into the DBMS language query to retrieve the data from the database. The proposed system also has a query recommendation engine that recommends the possible queries in case of any error or if the result is not found. The intelligent search engine proposed in this work is a generic tool that is not dependent on the front or back end of the system, therefore it can be used for any database system.

Keywords- Natural language processing, Query autocompletion, Named entity recognition, Text categorization, Term frequency-inverse document frequency (TF-IDF).

1. Introduction

The database management system plays an imperative role in organizing and storing data in a structured format precisely so that data can be retrieved and processed to acquire knowledge and information. Nowadays, almost every company, institute, and organization are using database systems to store and process all data to generate information (Kumar et al., 2019). However, a significant challenge persists for users unfamiliar with the intricacies of traditional Database Management System (DBMS) languages, such as Structured Query Language (SQL), Data Definition Language (DDL), and Data Control Language

(DCL). This challenge impedes their ability to effectively interact with and extract information from databases, limiting the democratization of data access across diverse user profiles (Kate et al., 2018).

An advanced solution for addressing this challenge is Natural Language Interfaces to Databases (NLIDB), which integrates Natural Language Processing (NLP) into database interactions (Affolter et al., 2019; Ozcan et al., 2020). Natural language processing allows computers and people to converse using their native languages (Androutsopoulos et al., 1995; Singh and Solanki, 2016). Natural language processing and database querying results enable users to interact naturally with databases, access and extract information using DBMS languages like SQL, DDL, and DCL, etc. (Malhotra and Rishi, 2019).

Earlier natural language interfaces had a limited vocabulary, making it difficult to express complex queries (Aditya et al., 2002; Tata and Lohman, 2008). There are enormous studies to understand the complete natural language (NL) queries (Baik et al., 2019; Saha et al., 2016; Yaghmazadeh et al., 2017; Zhong et al., 2017). In some queries, rules were used to transform NL queries into SQL, and in some other queries, machine learning algorithms were used. NL query interpreters like NaLIR (Li and Jagadish, 2014) and AT HENA (Saha et al., 2016) use rules to process words. Some machine learning methods (Yaghmazadeh et al., 2017; Zhong et al., 2017) work well with different ways people express themselves, but they need a lot of training data and may not work well in new situations. Others need extra input from users (Li et al., 2005; Kupper et al., 1993) or records of past queries (Baik et al., 2019) to clear up uncertainties, which can make things harder for users or be tricky to get. Most of these studies create simple SQL queries, like picking data from one table or combining data from different tables (JOIN).

This paper presents a generic architecture of an intelligent search engine enabling the querying of the database system in English. As users initiate a query, the ISE dynamically generates and presents a curated list of potential queries, streamlining the user experience and facilitating more accurate information retrieval. The ISE further employs key NLP components, including Named Entity Recognition, Tokenization, and Text Classification, to convert user-entered queries into a JSON file. This file serves as a versatile intermediary, allowing the translation of natural language queries into any DBMS language, thus eliminating the need for users to possess extensive knowledge of SQL or other specific DBMS languages.

Section 2 discusses previous research efforts and approaches aimed at addressing similar challenges. Section 3 details the methodology for the proposed Intelligent Search Engine Tool for Querying Database Systems. In Section 4, the experimental outcomes are presented, and a comprehensive discussion is facilitated, and in Section 5, the conclusion and future research are discussed.

2. Related Work

Kumar et al. (2019) presented a technique that allows users to query the database using the English natural language. With this method, the required data is obtained in the same language by generating an SQL query. Using a natural language query, an SQL query is created, incorporating specific details such as the names of the database and its columns. Tokenization is initially applied, and all stop words are eliminated from the input. The required operation is then determined by comparing the filtered input sentence with a list of keywords. Subsequently, a JSON file is produced, which is utilized to generate the SQL query. The user interface displays the generated SQL query for user utilization. In the future, the authors aim to offer a natural query recommendation framework.

Javanmard et al. (2020) proposed a study that considered the network load and database structure to compare the costs of optimizing database queries. According to the authors, hypercubes are more cost-effective than stochastic topologies for query optimization. In a distinct study, Agrawal et al. (2014)

proposed DBIQS, an innovative approach to automated, rapid, and dependable database querying. The authors demonstrated high efficiency in converting unsolvable queries into partially or fully answered queries.

Shah and Vanusha (2019) created a natural language interface for relational databases (RDBMS), which accepts English-like statements as inputs and generates SQL queries as responses. It parses natural language using semantic grammar and then translates it into SQL using NLP. In a separate study, Xu et al. (2019) presented Natural Language Database Querying (NADAQ). It enhances the encoder-decoder model with schema-aware bits during input processing. Additionally, the authors devised several methods for the augmented neural network to filter out irrelevant queries and propose natural language alternatives derived from SQL queries.

Bais et al. (2019) suggested a natural language interface for multimodal databases. The proposed application generates database language queries that are assessed against a database to answer questions written in English. The proposed interface works with SQL and XPATH databases making it easier to reuse queries and speeding up processing time. In addition, it can easily be altered to interface with other database models. Furthermore, the proposed system has the capability of improving its knowledge base (KB) through experience. In the future, the authors intend to provide a generic interface with multilingual capabilities for multimodal databases. Shelar et al. (2020) discussed various NLP libraries, such as Apache OpenNLP, Python's SpaCy, and TensorFlow. Some of these libraries offer pre-built NER models that can be customized. The authors evaluated these libraries using metrics like model size, F-score, training accuracy, prediction time, and ease of training. All models employed the same training and testing datasets. Among the models utilized, SpaCy, within the Python environment, emerged as the most effective performer.

Choi et al. (2021) introduced the RYANSQL, which stands for Recursively Yielding Annotation Network for SQL. By employing Statement Position Codes (SPC), the authors devised a method to generate non-nested SELECT statements from nested SQL queries. The authors also presented two input manipulation techniques to improve generation performance. Notably, RYANSQL secured the 3rd position overall in the challenging Spider benchmark dataset and ranked 1st among systems not utilizing database content. The authors propose that further enhancements could be achieved by encoding relation structures of the database schema, including foreign keys.

Wu et al. (2021) addressed the issue of natural language instructions by employing a hierarchical classification of semantic parsing algorithm. The natural language instructions were classified using the KWECS method. Subsequently, new key-value pairs are created based on TF-IDF keyword extraction and cosine similarity metrics. This information is then utilized in the SCADA control intermediate language for further conversion into the instruction query. The authors claimed an intent recognition accuracy of 96.5%, demonstrating enhanced capability in resolving the human-computer natural language interaction problem with the suggested model for control interface hierarchical classification and natural language query.

Wu et al. (2022) proposed an Intelligent Search Engine (ISE) based on semantic analysis, which extracts targets and conditions for natural queries from a constructed electrical power knowledge graph. It generates an index network by mapping vertices to edges, analyzing ontology networks, and conducting searches for direct and associated knowledge. This method integrates knowledge graphs and graph computing characteristics, enabling correlation analysis and multi-hop data mining within a domain knowledge graph at high speed. In addition to identifying user interests and intelligently filtering information, the recommended model also serves as a source for push notifications.

Yu et al. (2020) proposed a novel pretraining method, called GRAPPA, for table semantic parsing. The authors utilized a synchronous context-free grammar (SCFG) inferred from existing text-to-SQL datasets for mapping natural language queries to SQL queries. By generating question-SQL templates from text-to-SQL examples, the authors exclude references to schema components, values, and SQL actions. Their exclusive text-schema linking goal forecasts the syntactic function of a table column in the SQL for each pair. GRAPPA is trained on synthetic question-SQL pairings and related tables, using 475k artificial cases and 391.5k instances from existing data sets. This approach expressively decreases training time and Graphical Processing Unit cost. GRAPPA outperforms existing methods on four key semantic parsing benchmarks under both strong and light supervision.

Abbas et al. (2022) studied the NLIDB with Deep learning methodology and observed that the most used model for NLIDB systems is supervised learning with RNN. The fundamental techniques for constructing text-to-SQL models are sequence-to-sequence and sequence-to-set along with their limits and rewards. For example, sequence-to-sequence methods suffer order-matter difficulty, and sequence-to- set methods suffer a lack of context. At present, both methods are being accepted and tested simultaneously, and efforts are being made to resolve the challenges associated with them. The problem of order can be mitigated through reinforcement learning in the standard sequence-to-sequence model. Attention mechanisms are primarily employed to establish explicit connections and provide context for tokens, addressing the contextual challenge in sequence-to-set methodologies. These issues can also be alleviated by integrating NLP techniques with machine learning concepts. The main challenges in this research area have been identified as datasets like WikiSQL and Spider, lacking the capability to train models on real-time datasets.

3. Method and Material

Figure 1 describes the framework of the proposed Intelligent Search Engine (ISE). The architecture is divided into seven main parts: query autocompletion model, query categorization model, named entity recognition model, text categorization model, history manager, query recommendation model, and query generator. As soon as the user starts typing the query in the search textbox, the Query Autocompletion Model (QAM) takes the partially typed query as input and generates a list of suggestions in the dropdown box, the user can select the query from the suggestions or can type the whole query.

After the user enters a query, the Query Categorization Model identifies the main category of the query which helps to identify the user’s search intent. Next, the query is tokenized, and an array of tokens is generated. The Named Entity Recognition Model takes the same array of tokens as input and detects entities in the query. The identified entities are mapped to their internal names by the text categorization model. The history manager maintains the user's search history, eliminating the need to reprocess previously processed queries. The Query Autocompletion Model prioritizes frequently searched queries, making the ISE engine user-specific. If the user's query fails to retrieve data or encounters an error, the Query Recommendation Model suggests alternative queries related to the user-entered query. Finally, the Query generator will produce the JSON file from the natural query. This JSON file can then be easily converted into any other DBMS language query, such as an SQL query.

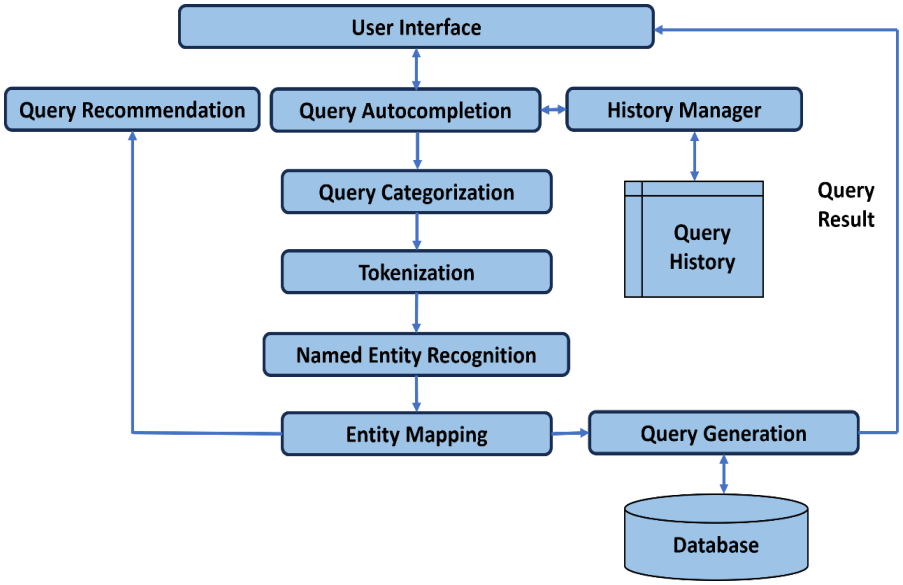


Figure 1. Intelligent search engine architecture.

3.1 Query Autocompletion Model

The Query autocompletion model generates a list of queries when the user starts typing. This decreases the physical and mental effort of the user (Jiang et al., 2017). In response to the user's keystroke, QAM generates and provides a ranked list of query suggestions matching the prefix typed. In addition to selecting a query from QAM's ranked list, the user can type the entire query manually. In this paper, a new algorithm based on the concept of Term Frequency-Inverse Document Frequency (TF-IDF) is presented for automatically generating queries.

3.1.1 TF-IDF

TF-IDF calculates how often a word appears in a document relative to its frequency across the entire corpus (Parmar and Kumbharana, 2017). It is used to find how relevant a given word is for a particular document. The TF-IDF weight is formed by the combination of two terms:

***Term Frequency* (*TF*):** It measures the frequency of a term in a document. The frequency of a word is higher in a long document than in a shorter document. The term frequency is divided by the total number of terms in the document.

$$\text{TF(t)} = \frac{Number\ of\ times\ term\ t\ appears\ in\ a\ document}{Total\ count\ of\ terms\ in\ the\ document} \quad (1)$$

***Inverse Document Frequency* (*IDF*):** IDF measures the significance of a term. In TF, all the terms are considered equally significant, but certain terms like "is", "the", and "of", have little importance and may appear repeatedly. In IDF computing, common terms are weighed down while rare terms are scaled up.

$$\text{IDF(t)} = log_e \frac{total\ count\ of\ documents}{number\ of\ documents\ with\ term\ t\ in\ it} \quad (2)$$

TF-IDF is utilized for the pre-processing of the knowledge base. The knowledge base employed here comprises IBM's natural language query data, containing various types of natural language queries utilized by different users. This knowledge base undergoes monthly updates to incorporate new queries and enhance the performance of the Query Autocompletion Model. During the pre-processing phase, a word dictionary

is generated for each word within the knowledge base. **Figure 2** illustrates the flow for creating the word dictionary.

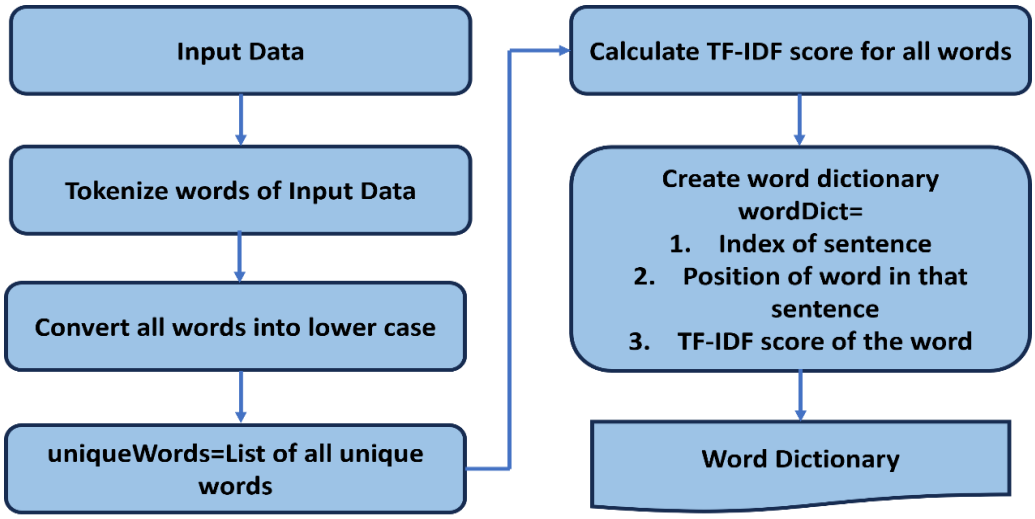


Figure 2. The flow of word dictionary creation.

First, all natural language queries from the knowledge base are tokenized into individual tokens. Then, these tokens (words) are converted to lowercase for simplicity, and a list of all unique words is created. Subsequently, utilizing the TF-IDF formula, the TF-IDF score for each unique word is calculated. The word dictionary stores data in key-value pairs, where the key represents a single word, and the corresponding value consists of multiple arrays. Each array contains three values: the index of the sentence in which the word (key) appears, the position of the word in that sentence, and the TF-IDF value of the word. This word dictionary serves as the knowledge base for the query autocompletion algorithm.

3.1.2 Trie Data Structure

Trie is a data structure in which data is organized in a tree format (Kobayashi et al., 2017). In this, data is organized as individual characters, and every node consists of multiple branches. Every single character of the keyword is stored in each single branch. The last node of every key is marked as the end of the word node, and *isEndOfWord* is a flag that is used to distinguish the node as the end of the word node. A simple Trie structure is shown in **Figure 3**, which has keywords like [Desktop, Device, System, Product, Program]. In the last node, the flag is set to true to indicate *isEndOfWord*.

QAM utilizes the Trie data structure to create lookups based on the query prefix. When the user initiates typing a query without specific keywords, containing only a few characters (e.g. Q = {'De'}), QAM searches the Trie data structure using the query prefix. The result of the Trie search is an array containing all the attribute names with the prefix ‘De’. Referring to the example in **Figure 3**, this search will produce the lookup array ["Device", "Desktop"] for the query prefix ‘De’. The user's query is separated into two parts: the query prefix and the query context. When the user inputs only characters that do not constitute any keywords, it is considered a query prefix. However, when the query includes keywords, that part of the query is regarded as the query context.

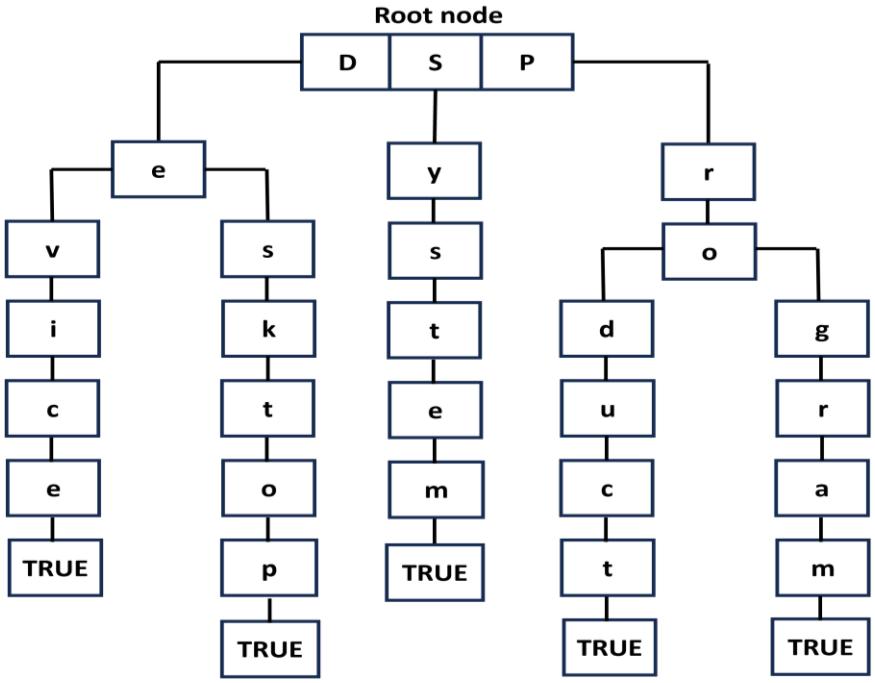


Figure 3. Trie data structure.

***Definition* 1:** Given,
D = {d1,d2,...,dn}, a document/natural language queries collection.
T = {t1,t2,t3,...,tn}, terms/lookup collection.

Q denotes the incomplete query. The query Q is divided into two parts: Query Context Qc (the sequence of complete keywords in the query) and Prefix Qp (the partially typed keyword), where $|Qc| \geq 0$ and $||Qp|| \geq 1$. |Qc| signifies the count of keywords in Qc, while ||Qp|| denotes the count of characters in Qp.

L = {l1,l2,l3,...,lk}, the set of lookup extracted from T having the same prefix as Qp. The probability of lookup suggestion 'l', given the partial query Qp, is calculated as shown below:

$$P(l|Qp) \tag{3}$$

S = {s1, s2, . . . , sk}, the collection of query suggestions retrieved from D having the same prefix as Qp and all the lookup of Qp. Assuming a partial query Q, the following equation gives the probability of a candidate suggestion 's':

$$P(s|Qc, l) \tag{4}$$

For the following partial query typed by the user:

product name e

The product name serves as the context of the query i.e. Qc. In this query text, the count of keywords in the Query context is |Qc| = 2. The "e" represents the query prefix, and the number of characters in the query prefix is one (||Qp|| = 1), while S denotes the Query Suggestions provided by the Query Autocompletion

Model. S = {product name equal to mobile, product name ends with m, product name equal to laptop, product name ends with p}.

In our problem definition, ||Qp|| ≥ 1 because the QAM provides the query suggestions when the user has type at least one character. Initially, the query doesn't contain complete keywords, with |Qc| ≥ 0.

3.1.3 Suggestion Without Context

In the case of a partial query with (|Qc| > 0) without any context (|Qc| = 0), we analyze the probability of P(s|Qp) using the equation. When the Query Context (|Qc| = 0) is not present, the model will consider the Query prefix as soon as the user types any keyword and calculates the marginalization of 'l' over the entire terms corpus. Thus, P(l|Qp) can be calculated as:

$$P(l|Qp) = \frac{P(l,Qp)}{P(Qp)} \tag{5}$$

where, l and Qp are defined in equation. Equation (5) can be estimated as follows:

$$P(l|Qp) = \frac{\sum_{t\in T} P(l,Qp,t)}{\sum_{t\in T}\sum_{l\in L} P(l,Qp,t)} \propto \sum_{t\in T} P(l,Qp,t) \tag{6}$$

After the Lookup computation, the QAM calculates the marginalization of suggestions candidate 's' over the whole natural language corpus taking all the extracted lookups (L) Qp.

$$\sum_{l\in L} P(s|l) = \sum_{l\in L} \frac{P(s,l)}{P(l)} \tag{7}$$

Equation (7) computes the probability of query suggestions 's' over-extracted lookups 'l'.

$$\sum_{l\in L} \frac{P(s,l)}{P(l)} = \sum_{l\in L} \left[\frac{\sum_{d\in D} P(s,l,d)}{\sum_{d\in D}\sum_{s\in S} P(s,l,d)}\right] \sum_{l\in L} P\left(\frac{s}{l}\right) \approx \sum_{l\in L} \left[\frac{\sum_{d\in D} P(s,l,d)}{\sum_{d\in D}\sum_{s\in S} P(s,l,d)}\right] \propto \sum_{l\in L}[\sum_{d\in D} P(s,l,d)] \tag{8}$$

3.1.4 Suggestion with Context

When the context is present in the query (|Qc| > 0), it is taken into consideration by QAM when generating suggestions. With a given context, the probability of a suggestion can be calculated as:

$$\sum_{l\in L} P(s|Qc,l) \quad \sum_{l\in L} P(s|l)P(s|Qc) \tag{9}$$

Putting Equation (8) in (9), we get:

$$\sum_{l\in L} P(s|Qc,l) = \sum_{l\in L}[\sum_{d\in D} P(s,l,d)] \frac{P(s,Qc)}{P(Qc)} \propto$$
$$\sum_{l\in L}[\sum_{d\in D} P(s,l,d)][\sum_{d\in D} P(s,Qc,d)] \ \sum_{l\in L} P(s|Qc,l) = \sum_{l\in L}[\sum_{d\in D} P(s,l,Qc,d)] \tag{10}$$

According to Equation (10), the conditional probability of the suggestion 's', given Qc and Qp, is estimated as the sum of the joint probability of 's', Qc, l, and d over the set of Query Document (D) and extracted lookups. Below is the proposed algorithm for query autocompletion:

Algorithm 1: Pseudo-code for Query Automation

a) Count the number of words in the input query.
b) If the number of words is not equal to 1 then go to step 5.
c) Find Lookup for input word using Trie Data structure.
d) For every lookup/word go to step 9.
e) For every word in the input query:
   substring words [0 to Length-2]

f) Find Lookup for words [Length-1].
g) For j in every lookup value:
   Sublist ⟵ (substring + " " + j).
h) If the sublist is not empty then go to step 9.
i) Calculate the following values in key-value pairs using the word dictionary.
   occurrence_count = (Index no of the sentence, Number of occurrences in the sentence)
   weight_count = (Index no of the sentence, Percentage of words appear in a sentence)
   tfidf_count = (Index no of the sentence, TF-IDF score of that word)
   sequence_count = (Index no of the sentence, number of sequences matches in the sentence)
j) Rules for ranking the result.
k)
Rule 1. Result [] ⟵ add value having high sequence_count and 100% weight_count.
Rule 2. Result [] ⟵add value having sequence_count greater than 1.
Rule 3. Result [] ⟵add value having the highest tfidf_count.
Rule 4. Result [] ⟵add value having a high weight_count.
Rule 5. Result [] ⟵add the remaining values.
l) Stop.

3.2 Query Categorization Model

In a database management system, data is typically organized into predefined classes for categorization purposes. For instance, in an e-commerce website, the database may categorize data into classes such as customer, product, delivery, etc. The Query Categorization model is employed to identify these types of categories within natural language queries.

To develop the Query Categorization model, the proposed system utilizes text classification. Text classification involves automatically assigning text to one or more predefined classes (Kobayashi et al., 2017). It determines the category of a given document or text and provides results in the form of binary or multiple classifications. The steps of text classification comprise text pre-processing, text feature extraction, and constructing the classification model. When the user enters a query or selects one from suggestions, the query is processed by the Query Categorization model.

The query categorization model classifies that query to some pre-defined class of the database. For example, when the user enters the following natural language query:

"Give the list of all users whose email addresses contain gmail.com".

The query categorization model identifies that this query belongs to the Customer class. **Figure 4** represents the architecture of the query categorization model. Once the query categorization model identifies the main category of the natural query, then ISE can determine the type of query and the type of information the user seeks. This information can also be utilized to comprehend the user's search pattern or their most interesting domain of information. The Query categorization model aids in improving the performance of the ISE to provide accurate results for the user's query. Additionally, it helps reduce time complexity by limiting the database search scope.

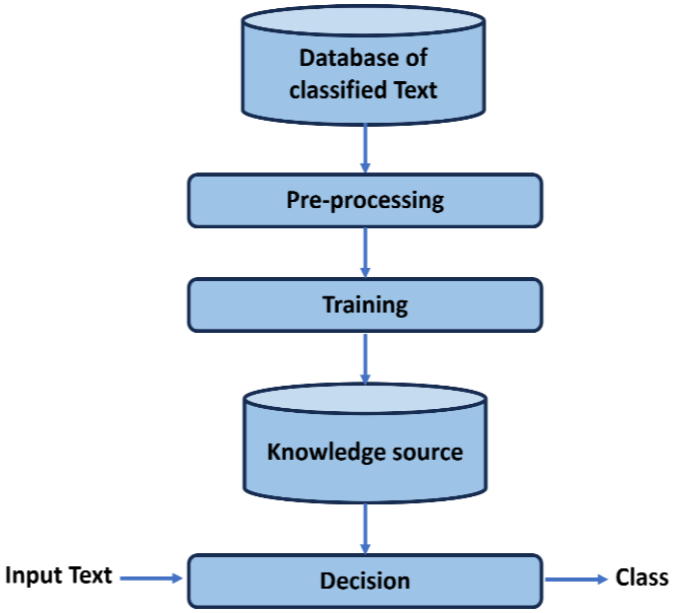


Figure 4. Query categorization model.

3.3 Named Entity Recognition and Tokenization

Tokenization is the process of splitting text into a list of tokens. In the context of ISE, tokenization splits the input query into tokens. Named Entity Recognition (NER) takes this list of tokens as input and identifies the entities from the token list. This model retrieves the keywords or entities from the input statement and comprehends the meaning of those entities, sentences, and their relationships, as shown in **Figure 5**. Utilizing previously published information, the NER system detects keywords such as person, institution, geographical location, and time. The basic NER algorithm accepts structured and unstructured text as input, processes it, and detects or locates entities (Ertopçu et al., 2017). The NER model, for example, will identify "Steve Jobs" as a single entity rather than separate "Steve" and "Jobs".

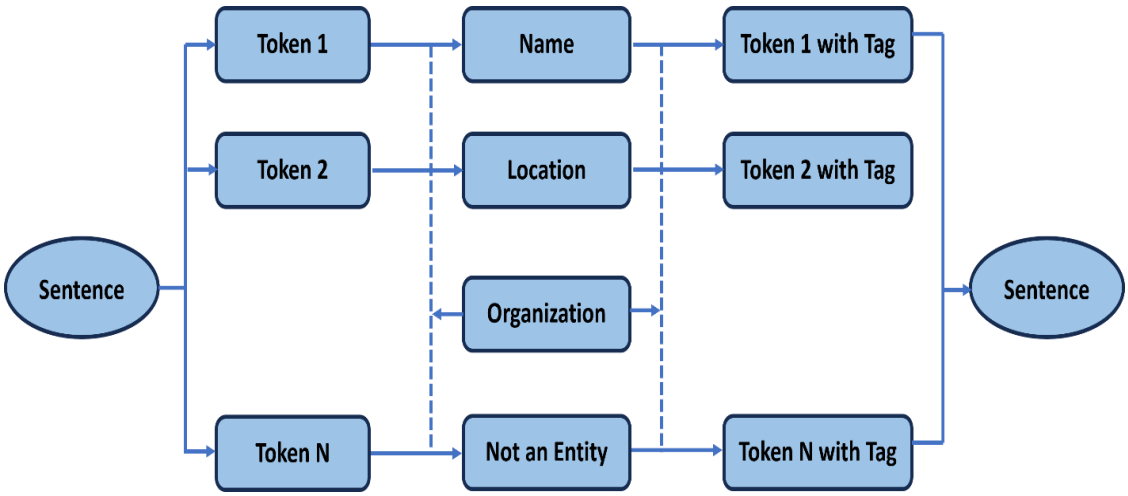


Figure 5. Named entity recognition.

The NER model of the intelligent search engine identifies entities as attributes, criteria, and values to create a structured query. For example, consider the search statement: “Give the list of all users whose email addresses contain gmail.com”. In this instance, the NER model will identify the “email addresses” as an attribute, “contain” as the criteria, and “gmail.com” as the value.

3.4 Entity Mapping Model

The entity mapping model is used to map the entity in the natural query to the internal name of that entity to form a JSON. It is possible for users to refer to a single entity with different names. For example, a user may refer to the email address entity as an email ID, mail ID, mail address, mail IDs, email addresses, etc. The entity mapping model will fill this gap by mapping all such entities to a single internal name, such as an email address. To develop the entity mapping model, the text categorization method used in the Query categorization model is also applied here.

3.5 Query Generation Model

The query generation model generates the JSON file using all the entities identified from the natural language query. This JSON file can then be converted to any required DBMS language query. In our case, the JSON file is converted to an SQL query to fetch data from the database.

3.6 History Manager

A history manager (HM) is used to make ISE more user-specific. It maintains users' search history by storing the recently searched and processed queries. If the user searches the same query again, there is no need to process the recently processed query again. The history manager keeps track of the most and least used queries by specific users, leading to changes in the weight of words in the word dictionary. When the user starts typing a natural language query, the most searched query by the user will have higher weightage and will thus be suggested first by the Query Autocompletion Model. HM also maintains a log file of new and unique search statements typed by the user, which are not suggested by the Query Autocompletion Model. This allows new search queries to be used as a knowledge base for the Query Autocompletion Model.

3.7 Query Recommendation Model

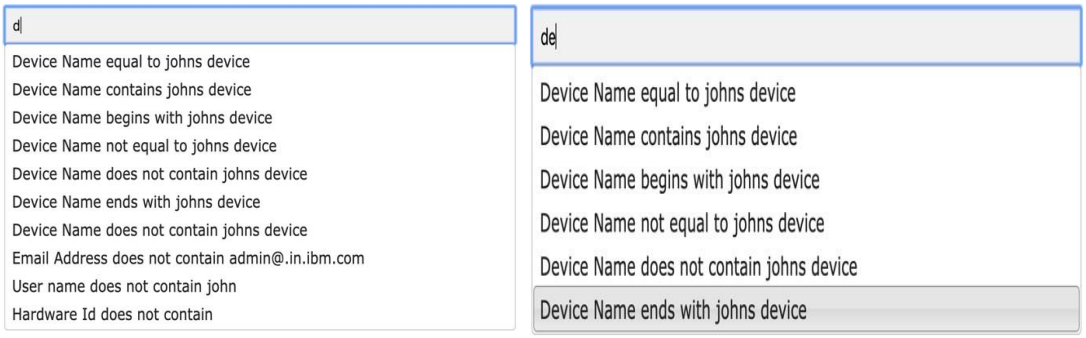
If the query entered by the user fails to construct a structured query, the Query Recommendation model generates a list of queries similar to the one entered by the user. This model produces queries that resemble the user's query or generates a list of queries related to the words present in the user's query.

4. Result and Discussion

To implement the Intelligent Search Engine (ISE), we have used IBM's dataset. The dataset for the query auto-completion model comprises all the possible lists of natural language queries a user can type, ensuring comprehensive coverage of user query patterns and improving the robustness of the model to accommodate wide-array of search queries. This dataset gets updated in a fixed time to add new natural queries searched by users.

Figure 6(a) demonstrates the result of the query auto-completion model, offering clarity on the data considered. When 'd' is entered in the search box, a list of possible natural language queries is produced and displayed in the dropdown. This list includes all queries starting with 'd', having the prefix 'd' (which are prioritized and ranked highly), and queries with 'd' as a substring appearing at the end, which are ranked lower. **Figure 6(b)** demonstrates that upon entering 'd' followed by 'e', 'de' is considered as input and the suggestions comprise all queries starting with 'de'.

The Query Autocompletion Model initially displays suggestions whose prefix matches the query, ensuring clarity regarding the data used. If it does not find any suggestion with a matching prefix, it then presents suggestions containing the query as a substring. Furthermore, the dataset for the Query Categorization model encompasses all possible lists of natural language queries a user can search along with predefined classes, addressing the need for clarification regarding the data considered. When the user inputs a natural language query such as 'Email Address equal to sam@gmail.com', the Query Categorization model assigns the query to the appropriate category. To evaluate the model, we generated a set of queries and supplied them as input to the model, ensuring precise consideration of the data. Table 1 illustrates the categories identified by the Query Categorization model for the input queries, providing clarity on the data utilized.



a) b)

Figure 6. Query auto-completion result.

Table 1. Result of query categorization.

Natural Language Query	Category identified by Query Categorization model
Email address equal to sam@maas.com	User
Device name not equal to work device	Device
User name not equal to sam	User
User name does not contain f	User
Device type equal to Laptop	Device
Email address not equal to mark@maas.com	User
Model begins with e	Hardware
User name not equal to Alexa	User
Device name equal to work device	Device
Model equal to lumia	Hardware
Billing ID not equal to 736651534	User

The named entity recognition model identifies the entities in the natural language query to create the JSON file. In our case, these entity types are attributes, criteria, and values. The dataset for entity mapping requires all possible entity names with the internal entity name. **Figure 7** shows the output of the NER model. The NER model takes tokenized data as input and identifies the entities from the same input, as shown in **Figure 7**. It also provides the probability of each entity prediction.

The Entity mapping training dataset contains all possible versions or abbreviations of attribute names and their corresponding internal names. We tested the model by inputting various abbreviations and versions of attribute names, some of which include underscores instead of spaces. Table 2 displays some inputs and the probability of mapping those inputs to their respective internal names. For example, when the

user enters the attribute name 'dev TYPES', the model maps it to the internal name 'device Type' with a probability of 0.9836.

```
--- Entities (detected with custome NER) ---
0 to 1: attribute (Device name)
2 to 3: criteria (equal to)
4 to 4: value (Samsung)
6 to 7: attribute (Email address)
8 to 9: criteria (equal to)
10 to 10: value (hem@smail.com)

--- Entities and scores (detected with beam search) ---
0 to 1: attribute (1.000000)
2 to 3: criteria (1.000000)
4 to 4: value (1.000000)
6 to 7: attribute (1.000000)
8 to 9: criteria (1.000000)
10 to 10: value (1.000000)
```

Figure 7. Result of named entity recognition.

Table 2. Result of entity mapping.

Entities	Device Nm	Internal Names		User Nm
		Device Type	Email Add	
emails_adds	0.0000	0.0000	0.9999	0.0000
dev TYPES	0.0159	0.9836	0.0001	0.0002
usrnms	0.0000	0.0000	0.0000	0.9999
devic nam	0.9999	0.0000	0.0000	0.0000
devtyp	0.0001	0.9998	0.0000	0.0000
emailadds	0.0000	0.0000	0.9999	0.0000
emails_addresses	0.0000	0.0000	0.9999	0.0000
user nms	0.0000	0.0000	0.0000	0.9999
devices nams	0.9999	0.0000	0.0000	0.0000

In the example below, the step-by-step result of ISE is shown:

- User's Natural Language query:
  show the list of devices where dev names are equal to john's device.
- result of Query categorization model:
  show the list of devices where dev names equal to john's devicecategory = Device
- result of Tokenization
  <show> <the> <list> <of> <device> <where> <dev> <names> <equal> <to> <john's> <device>
- result of the Named entity recognition model
  <show, 0> <the, 0> <list, 0> <of, 0> <device, 0> <where, 0> <dev, attribute> <names,attribute> <equal, criteria> <to, criteria>< john's, value> <device, value>
- result of Text categorization model
  <show, 0> <the, 0> <list, 0> <where, 0> <device, attribute> <name, attribute> <equal,criteria> <to, criteria> < john's, value> <device, value>
  - result of Query generator
    JSON File

4.1 Comparison of the Proposed Model with the Existing Techniques

To analyze the quantitative representation of the proposed model, we examined its performance against some of the following state-of-the-art approaches using the IBM dataset:

***RAT-SQL* (Wang et al., 2019):** An integrated framework utilizing the relation-aware self-attention mechanism to handle schema encoding, schema linking, and feature representation in the context of a text-to-SQL encoder.

***Context-Dependent Seq2Seq* (Yu et al., 2019):** The model uses a bi-LSTM database schema encoder for context-dependent SQL generation across various domains. The encoder utilizes bag-of-words representations for column headers and can adjust to choose between an SQL keyword or a column header.

***SQLNet* (XU et al., 2018):** This approach utilizes a sketch-based methodology, combining a dependency graph with a sketch. The system uses a sequence-to-set model and column attention mechanism for query synthesis based on a provided sketch, allowing each prediction to consider previous ones.

***IR-Net* (Guo et al., 2019):** The model, known as Intermediate Representation (IR)-Net, is designed to tackle text mismatches and column name prediction issues caused by out-of-domain words. The model consists of distinct components: the NL encoder encodes natural language input into embedding vectors, a bi-directional LSTM builds hidden states, the schema encoder processes the database schema to generate column and table representations, and the decoder utilizes context-free grammar to construct Sem-QL queries.

Table 3 compares and contrasts various methods, highlighting their overall accuracy. In comparison to existing models, our proposed model performs better. The improved performance of our model is the result of the synergistic integration of several key components. Several models, including query autocomplete, query categorization, named entity recognition, text categorization, a history manager, a query recommendation model, and a query generator, contribute to the efficacy of the model. As a result, the model shows more capabilities for interpreting and generating SQL queries from natural language inputs.

Table 3. Comparison of the proposed model with the existing techniques.

Technique	Overall Accuracy
RAT-SQL	0.921
CD-Seq2Seq	0.902
SQLNet	0.891
IR-Net	0.936

5. Conclusion

This paper presents an Intelligent Search Engine (ISE) that converts natural language queries into JSON-formatted database queries. The system functions independently of database systems; therefore, its adaptability and relevance have been enriched. The primary components of the proposed model are Named Entity Recognition, Query Autocompletion, and Query Categorization. The NER model has a 99% accuracy rate in identifying entities within queries. The Query Autocompletion model helps users to finish partial queries, while the Query Categorization model identifies user search intent with a 98% accuracy rate. The Entity Mapping feature greatly updates the query creation process by assigning internal names to different entity forms.

Furthermore, the recommended model exhibits an overall accuracy of 93.6%. We effectively compared it

with state-of-the-art approaches like RAT-SQL, CD-Seq2Seq, SQLNet, and IR-Net. The results prove the superiority of our model, demonstrating its ability to outperform current approaches. Our tool allows users to access complex database systems by translating human-readable queries into database-specific language. This reduces the learning curve for users. The performance of the application is also significantly enhanced by features such as named entity recognition, entity mapping, and query autocompletion.

Future research directions of the proposed work include addressing more complex natural language queries in order to improve the system's capability. The aim is to tailor the entire system to the user's needs, in order to deliver a superior user experience. In addition, the research can be extended by implementing a multilingual interface for multimodal databases. With the multi-language interface, users can interact with the database in different languages, increasing accessibility and flexibility.

Conflict of Interest

The authors confirm that there is no conflict of interest to declare for this publication.

Acknowledgments

We thank all those referees who have contributed their time and suggestions to our work.

References

Abbas, S., Khan, M.U., Lee, S.U.J., Abbas, A., & Bashir, A.K. (2022). A Review of NLIDB with deep learning: findings, challenges, and open issues. *IEEE Access, 10*, 14927-14945. https://doi.org/10.1109/access.2022.3147586.

Aditya, B., Bhalotia, G., Chakrabarti, S., Hulgeri, A., Nakhe, C., Parag, & Sudarshanxe, S. (2002). Chapter 107-Banks: Browsing and keyword searching in relational databases. In: Bernstein, P.A., Ioannidis, Y.E., Ramakrishnan, R., Papadias, D. (eds) *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Morgan Kaufmann, Hong Kong SAR, China. pp. 1083-1086. https://doi.org/10.1016/b978-155860869-6/50114-1.

Affolter, K., Stockinger, K., & Bernstein, A. (2019). A comparative survey of recent natural language interfaces for databases. *The VLDB Journal, 28*(5), 793-819. https://doi.org/10.1007/s00778-019-00567-8.

Agrawal, R., Chakkarwar, A., Choudhary, P., Jogalekar, U.A., & Kulkarni, D.H. (2014). DBIQS-An intelligent system for querying and mining databases using NLP. In *2014 International Conference on Information Systems and Computer Networks* (pp. 39-44). IEEE. Mathura, India. https://doi.org/10.1109/iciscon.2014.6965215.

Androutsopoulos, I., Ritchie, G.D., & Thanisch, P. (1995). Natural language interfaces to databases - an introduction. *Natural Language Engineering, 1*(1), 29-81. https://doi.org/10.1017/s135132490000005x.

Baik, C., Jagadish, H.V., & Li, Y. (2019, April). Bridging the semantic gap with SQL query logs in natural language interfaces to databases. In *2019 IEEE 35th International Conference on Data Engineering* (pp. 374-385). IEEE. Macao, China. https://doi.org/10.1109/icde.2019.00041.

Bais, H., Machkour, M., & Koutti, L. (2019). An independent-domain natural language interface for multimodal databases. *International Journal of Computational Intelligence Studies, 8*(3), 206-222.

Choi, D.H., Shin, M.C., Kim, E.G., & Shin, D.R. (2021). RYANSQL: Recursively applying sketch-based slot fillings for complex text-to-SQL in cross-domain databases. *Computational Linguistics, 47*(2), 309-332.

Ertopçu, B., Kanburoglu, A.B., Topsakal, O., Acikgoz, O., Gürkan, A.T., Özenç, B., Çam, I., Avar, B., Ercan, G., & Yıldız, O.T. (2017). A new approach for named entity recognition. In *2nd International Conference on Computer Science and Engineering* (pp. 474-479). IEEE. Antalya, Turkey. https://doi.org/10.1109/ubmk.2017.8093439.

Guo, J., Zhan, Z., Gao, Y., Xiao, Y., Lou, J.G., Liu, T., & Zhang, D. (2019). Towards complex text-to-SQL in cross-domain database with intermediate representation. *Computation and Language*. arXiv preprint arXiv:1905.08205.

Javanmard, M.M., Ahmad, Z., Kong, M., Pouchet, L.N., Chowdhury, R., & Harrison, R. (2020). Deriving parametric multi-way recursive divide-and-conquer dynamic programming algorithms using polyhedral compilers. In *Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization* (pp. 317-329). IEEE. New York. https://doi.org/10.1145/3368826.3377916.

Jiang, D., Chen, H., & Cai, F. (2017). Exploiting query's temporal patterns for query autocompletion. *Mathematical Problems in Engineering, 2017*, 7490879. https://doi.org/10.1155/2017/7490879.

Kate, A., Kamble, S., Bodkhe, A., & Joshi, M. (2018). Conversion of natural language statement into SQL query. In *Proceedings of the 2018 Second International Conference on Electronics, Communication and Aerospace Technology* (pp. 488-491). IEEE. Coimbatore, India. https://doi.org/10.1109/iceca.2018.8474639.

Kobayashi, V.B., Mol, S.T., Berkers, H.A., Kismihok, G., & Hartog, D.N. (2017). Text classification for organizational researchers: A tutorial. *Organisational Research Method, 21*(3), 766-799. https://doi.org/10.1177/1094428117719322.

Kumar, A.H.M., Kumar, A.R., Mahadevaswamy, H.P., & N, S.D. (2019). Providing natural language interface to database using artificial intelligence. *International Journal of Scientific and Technology Research, 8*(10), 1074-1077.

Küpper, D., Storbel, M., & Roesner, D. (1993). NAUDA: A cooperative natural language interface to relational databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data* (pp. 529-533). Washington DC, USA. Association for Computing Machinery. https://doi.org/10.1145/170035.171543.

Li, F., & Jagadish, H.V. (2014). Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment, 8*(1), 73-84. https://doi.org/10.14778/2735461.2735468.

Li, Y., Yang, H., & Jagadish, H.V. (2005). NaLIX: An interactive natural language interface for querying XML. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data* (pp. 900-902). Baltimore, Maryland. Association for Computing Machinery. https://doi.org/10.1145/1066157.1066281.

Malhotra, D., & Rishi, O.P. (2019). A comprehensive review from hyperlink to intelligent technologies based personalized search systems. *Journal of Management Analytics, 6*(4), 365-389.

Őzcan, F., Quamar, A., Sen, J., Lei, C., & Efthymiou, V. (2020). State of the art and open challenges in natural language interfaces to data. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (pp. 2629-2636). Association for Computing Machinery. Portland, OR, USA.

Parmar, V.P., & Kumbharana, C.K. (2017). Implementation of trie structure for storing and searching of English spelled homophone words. *International Journal of Scientific and Research Publications, 7*(1), 1-5.

Saha, D., Floratou, A., Sankaranarayanan, K., Minhas, U.F., Mittal, A.R., & Ozcan, F. (2016). ATHENA: An ontology-driven system for natural language querying over relational data stores. *Proceeding of the VLDB Endowment, 9*(12), 1209-1220. https://doi.org/10.14778/2994509.2994536.

Shah, D., & Vanusha, D. (2019). Optimizing natural language interface for relational database. *International Journal of Engineering and Advanced Technology, 8*(4), 131-135.

Shelar, H., Kaur, G., Heda, N., & Agrawal, P. (2020). Named entity recognition approaches and their comparison for custom NER model. *Science & Technology Libraries, 39*(3), 324-337.

Singh, G., & Solanki, A. (2016). An algorithm to transform natural language into SQL queries for relational databases. *Selforganizology, 3*(3), 100-116. http://www.iaees.org/publications/journals/selforganizology/onlineversion.asp.

Tata, S., & Lohman, G.M. (2008). SQAK: doing more with keywords. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data* (pp. 889-902). Association for Computing Machinery. Vancouver, Canada. https://doi.org/10.1145/1376616.1376705.

Wang, B., Shin, R., Liu, X., Polozov, O., & Richardson, M. (2019). RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. *Computational and Language.* arXiv preprint arXiv:1911.04942. https://doi.org/10.48550/arXiv.1911.04942.

Wu, H., Shen, C., He, Z., Wang, Y., & Xu, X. (2021). SCADA-NLI: A natural language query and control interface for distributed systems. *IEEE Access*, *9*, 78108-78127. https://doi.org/10.1109/access.2021.3083540.

Wu, X., Tang, Y., Zhou, C., Zhu, G., Song, J., & Liu, G. (2022). An intelligent search engine based on knowledge graph for power equipment management. In *2022 5th International Conference on Energy, Electrical and Power Engineering* (pp. 370-374). IEEE. Chongqing, China. https://doi.org/10.1109/ceepe55110.2022.9783291.

Xu, B., Cai, R., Zhang, Z., Yang, X., Hao, Z., Li, Z., & Liang, Z. (2019). NADAQ: natural language database querying based on deep learning. *IEEE Access*, *7*, 35012-35017. https://doi.org/10.1109/access.2019.2904720.

Xu, X., Liu, C., & Song, D. (2018). SQLNeT: Generating structured queries from natural language without reinforcement learning. *Computation and Language.* arXiv preprint arXiv:1711.04436. https://doi.org/10.48550/arXiv.1711.04436.

Yaghmazadeh, N., Wang, Y., Dillig, I., & Dillig, T. (2017). SQLizer: query synthesis from natural language. *Proceedings of the ACM on Programming Languages, 1*(OOPSLA), 1-26. https://doi.org/10.1145/3133887.

Yu, T., Wu, C.S., Lin, X.V., Wang, B., Tan, Y.C., Yang, X., Radev, D., Socher, R., & Xiong, C. (2020). Grappa: Grammar-augmented pre-training for table semantic parsing. *Computation and Language*. arXiv preprint arXiv:2009.13845. https://doi.org/10.48550/arXiv.2009.13845.

Yu, T., Zhang, R., Yasunaga, M., Tan, Y.C., Lin, X.V., Li, S., Er, H., Li, I., Pang, B., Chen, T., Ji, E., Dixit, S., Proctor, D., Shim, S., Kraft, J., Zhang, V., Xiong, C., Socher, R., & Radev, D. (2019). Sparc: Cross-domain semantic parsing in context. *Computation and Language.* arXiv preprint arXiv:1906.02285. https://doi.org/10.48550/arXiv.1906.02285.

Zhong, V., Xiong, C., & Socher, R. (2017). Seq2sql: Generating structured queries from natural language using reinforcement learning. *Computation and Language.* arXiv 2017. arXiv preprint arXiv:1709.00103. https://doi.org/10.48550/arXiv.1709.00103.