Received 29 August 2022, accepted 8 September 2022, date of publication 15 September 2022, date of current version 26 September 2022.

*Digital Object Identifier 10.1109/ACCESS.2022.3206794*

RESEARCH ARTICLE

# Efficient SPARQL Queries Generator for Question Answering Systems

**YI-HUI CHEN[1,2], (Member, IEEE), ERIC JUI-LIN LU[3], (Member, IEEE), AND YING-YEN LIN[3]**
[1]Department of Information Management, Chang Gung University, Taoyuan 33302, Taiwan
[2]Kawasaki Disease Center, Kaohsiung Chang Gung Memorial Hospital, Kaohsiung 83301, Taiwan
[3]Department of Management Information Systems, National Chung Hsing University, Taichung 402204, Taiwan

Corresponding author: Eric Jui-Lin Lu (jllu@nchu.edu.tw)

The work of Yi-Hui Chen was supported in part by the National Science and Technology Council of the Republic of China, Taiwan, under Grant 110-2221-E-182-026-MY3; and in part by the Kaohsiung Chang Gung Memorial Hospital under Grant CMRPD3M0011.

**ABSTRACT** Much like traditional database querying, the question answering process in a Question Answering (QA) system involves converting a user's question input into query grammar, querying the knowledge base through the query grammar, and finally returning the query result (i.e., the answer) to the user. The accuracy of query grammar generation is therefore important in determining whether a Question Answering system can produce a correct answer. Generally speaking, incorrect query grammar will never find the right answer. SPARQL is the most frequently used query language in question answering systems. In the past, SPARQL was generated based on graph structures, such as dependency trees, syntax trees and so on. However, the query cost of generating SPARQL is high, which creates long processing times to answer questions. To reduce the query cost, this work proposes a low-cost SPARQL generator named Light-QAWizard, which integrates multi-label classification into a recurrent neural network (RNN), builds a template classifier, and generates corresponding query grammars based on the results of template classifier. Light-QAWizard reduces query frequency to DBpedia by aggregating multiple outputs into a single output using multi-label classification. In the experimental results, Light-QAWizard's performance on Precision, Recall and F-measure metrics were evaluated on the QALD-7, QALD8 and QALD-9 datasets. Not only did Light-QAWizard outperform all other models, but it also had a lower query cost that was nearly half that of QAWizard.

**INDEX TERMS** Question answering system (QA), SPARQL query, query cost, recurrent neural network (RNN), question answering over linked data (QALD).

## I. INTRODUCTION

In recent years, Natural Language Processing (NLP) has become an important research field in Artificial Intelligence (AI), and Question Answering (QA) systems are a key subject within NLP [1], [2], [3]. When a user asks a question, the system can understand the question and give the correct answer. For example, well-known real-world products such as Siri, Alisha, and OK Google use voice question inputs to look up answers on the Internet and return search results.

QA systems can be divided into two types [4]: one is information retrieval-based, and the other is knowledge-based. The former extracts the keywords in the question sentence, and uses the search engine to search for related documents on the Internet. Afterwards, the searched web links or document paragraphs are acquired and filtered by search engines. The latter pulls data from knowledge bases (KB for short), such as DBpedia [5]. In DBpedia, the entity is described by an URI as linked data. The related entities are linked together through linked data. For example, the place "England" is linked to the URI http://dbpedia.org/resource/England, which can be denoted by dbr:England. Linking the data, the request question can be converted into a query language (such as SPARQL [6], SQL, etc.) that is compatible with the KB's data structure through a semantic parser. The resulting SPARQL yields query results that fulfill the user's query intentions.

The associate editor coordinating the review of this manuscript and approving it for publication was Wai-keung Fung.

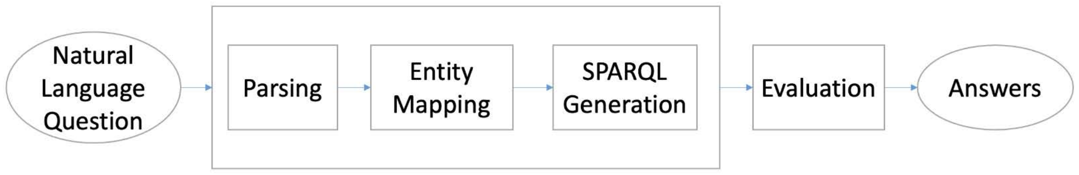


**FIGURE 1.** Typical QALD system architecture [7].

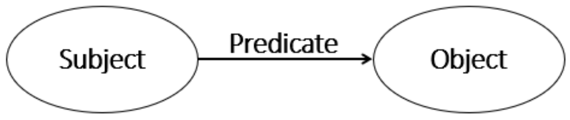


**FIGURE 2.** Illustration of an RDF triple.

Therefore, the accuracy of SPARQL generation decides the accuracy of the query results. To improve accuracy, past works [2], [3], [7] have sought to generate as many SPARQLs as possible to find all query results in order to address the user's query intent. However, this approach results in high query costs, which means DBpedia must handle a lot of unnecessary query requests. This paper focuses on proposing a low-cost, highly accurate KB-based QA system.

A typical KB-based QA system architecture, as shown in Figure 1, consists of four components: parsing, entity mapping, SPARQL generation, and evaluation. In the parsing phase, the lemma and part-of-speech tags (POS tags) of words or phrases are obtained from the input question. Named-entity recognition (NER) is used to label the entity type. In the entity mapping phase, words and phrases are mapped and linked to entities in DBpedia based on the tagged entity type. The entity type can be divided into three types: Named entity, property entity, and class entity. Names, events, objects, and places are marked as named entities (e.g., dbr:England). Property entities are used to represent relationships between named entities (e.g., dbp:capital). The class entity describes the class of the entity (e.g., dbo:country). The dependency tree structure can determine pivot words or phrases according to the tagged entity types. Afterward, the SPARQL queries can be generated by the basis of the tree structure. The SPARQL queries are executed to query the answers from KB in the evaluation phase. A SPARQL query is composed of RDF triples. An RDF triple is of the form (Subject, Predicate, Object) as shown in Figure 2, which can be expressed as an entity (Subject), an attribute (Predicate) and a value (Object). For example, the statement "the capital city of England is London", consists of a subject ("England"), a predicate ("the capital city"), and an object ("London"). Note that the subject and the predicate can only be described by a URI, and the object can be addressed by a URI or literal.

The three main types of SPARQL generation are Semantic Query Graph (SQG) searches [2], [3], [8], template designs [1], and machine learning solutions [7], [9]. Based on SQG generated by a dependency tree structure of a question, Ochieng [8] proposed a framework to translate natural language to SPARQL. gAnswer2 [3] uses knowledge graph structure to recursively search the subgraphs to find all the possible RDF triples. gAnswer2 looks for all possible SPARQL queries to increase the probability of finding an accurate answer, but it spends a lot of time sending unnecessary queries to DBpedia, increasing query costs. The query cost here refers to the frequency of queries to DBpedia. TWDAqua [2] uses Ngrams in entity mapping, which also increases the frequency of queries to DBpedia. In addition, WDAqua only considers the semantic graph in the KB, but does not consider syntax issues, resulting in low precision of query results. The experiments compare the performance of our proposed method to those of gAnswer2 [3] and TWDAqua [2], as they have both won the QALD (Question and Answering Linked Data) competition [10], [11], [12] in the last three years. QALD is an open-domain question dataset that serves as a benchmark dataset for researchers to evaluate the performance of designed models against other models.

In template design, the question is transformed into an intermediate data format (IDF) [1], such as an entity type tagger, and then a SPARQL query base of the dependency tree is generated to reduce the cost of searching subgraphs in SQG. However, depending on the template, all the possible generated SPARQL queries increase the query cost. For machine learning solutions, Yin *et al.* [13] designed a neural SPARQL machine that uses eight neural network models to transform problems into SPARQL queries. It answers simple and fixed-answer questions well, such as closed-domain questions, but is not suitable for complex and diverse datasets, such as open-domain questions (such as the QALD dataset [ [10], [11], [12]). QAWizard [7] answers input questions through two stages of Maximum Entropy Markov Model (MEMM) training, including an entity type tagging stage and an RDF type tagging stage. Seven predefined templates are used to generate SPARQL queries based on the results of the two stages. Although QAWizard's query accuracy is better than those of gAnswer2 and WDAqua, its query cost is still high.

To reduce query cost and achieve high query accuracy, the proposed approach, named Light-QAWizard, consists of two parts: an entity type tagger and template classifier. A Recursive Neural Network (RNN) is used to build the template classifier. To improve accuracy, Light-QAWizard converts the question answering problem from a multi-class problem to a multi-label problem by using the label powerset method [14]. Based on the results of the template classifier, the necessary SPARQL queries are generated. Therefore, this model avoids generating unnecessary SPARQL queries in order to reduce query cost and thereby speed up system response time. In the experiments, Light-QAWizard had the lowest query cost among the models compared to schemes [3], [13]. Furthermore, when evaluating the QALD dataset [10], [11], [12], the precision, recall and F-measure of the proposed QA system were higher than those achieved with [2], [3], [13].

The rest of the paper is organized as follows. In Section 2, we describe related studies. Section 3 describes the proposed method in detail. Section 4 presents our experiments and compares other results. Finally, Section 5 draws conclusions and presents possible future work.

## II. RELATED LITERATURE

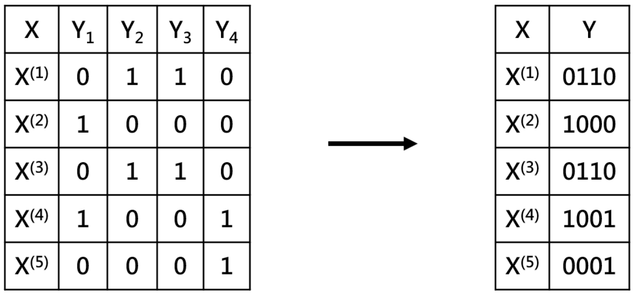
This section introduces the concept of the label powerset method [14] in subsection II-A. We also briefly review the differences between our work and that in [2], [3], and [13].

### A. LABEL POWERSET [14]

Multi-label classification trains a target object with multiple classifications. For example, movies can be labeled "comedy" and "romance". Based on multi-label classification, McCallum [15] used a Bayesian machine learning algorithm and a hybrid model applied to the Reuters-21578 dataset. Liu *et al.* [16] labeled a huge dataset with the most relevant labels, calling this process Extreme Multi-Label Text Classification (XMTC). Label powerset (LP for short), is a multi-classification method that combines multiple labels into one classification. Taking Figure 3 as an example, the original input $X$ can output multiple output $Y$ types. $X^{(1)}$ can output two types, $Y_2$ and $Y_3$, as shown in Figure 3(a). The results of the output types can be combined together as $Y$, as shown in Figure 3(b). The advantage is that the classifier predicts one value, $Y$, instead of four values, $Y_1$, $Y_2$, $Y_3$, $Y_4$. It is more difficult for a classifier to accurately predict four values than one. Therefore, after such a transformation, the accuracy is improved. This is the reason that we chose to base our proposed approach on LP-based template classifier.

### B. SPARQL GENERATOR

SPARQL is a query language that requests data in the form of RDF triples. Intui3 [17] divided each question into chunks, then uses right-to-left sequential chunks to generate SPARQL queries. Freitas *et al.* [18], [19] used Partially Ordered Dependency Structures (PODS) to transform the problem into a dependency tree. A pre-defined rule was applied to merge or prune the nodes of the tree. Next, a pivot node was selected as the start node of the tree to generate the SPARQL queries. Between 2019 and 2021, gAnswer2 [3] and WDQqua [2] were the winners of the QALD competition. gAnswer2 [3] is a QA system developed by Hu *et al.*, and



| X | $Y_1$ | $Y_2$ | $Y_3$ | $Y_4$ |
|---|---|---|---|---|
| $X^{(1)}$ | 0 | 1 | 1 | 0 |
| $X^{(2)}$ | 1 | 0 | 0 | 0 |
| $X^{(3)}$ | 0 | 1 | 1 | 0 |
| $X^{(4)}$ | 1 | 0 | 0 | 1 |
| $X^{(5)}$ | 0 | 0 | 0 | 1 |

| X | Y |
|---|---|
| $X^{(1)}$ | 0110 |
| $X^{(2)}$ | 1000 |
| $X^{(3)}$ | 0110 |
| $X^{(4)}$ | 1001 |
| $X^{(5)}$ | 0001 |

**FIGURE 3. An LP example.**

has a problem-based dependency tree structure, searching the tree from the root node to its children until the leaf nodes, forming relational phrases according to depth-first search (DFS). Then, the entity of DBpedia determines whether it matches the relational phrase. After that, the dependency tree is applied to create an SQG. Each subgraph of the SQG is assigned a score, and the scores are then sorted to select the top-$k$ subgraphs as candidates for generating SPARQL queries. WDAqua [2] uses N-grams to find all related entities in DBpedia. That is, by calculating the distances between entities, the mapped entity is like a root searching the DBpedia based on breath-first search (BFS) at depth 2. The SPARQL queries are generated according to the calculated distance. The results of the queried results are ranked, and the first ranked answer is returned to the user as the final answer.

In contrast to the aforementioned approaches, Xser [1] is a new branch of designing structured perceptrons to detect entity types. Based on entity types, a dependency graph, a semantic DAG, is constructed to present the relationships between entities. According to the relationships between entities, the pre-defined rules are used to generate the SPARQL queries that satisfy the query intents. After executing the SPARQL query, the answers are evaluated against the ranking results. Using MEMM, the past work QAWizard [13] proposed two stages, namely the training stage and query stage. The training phase aims to learn entity types and RDF labels from experience with questions defined in the QALD dataset. In the query phase, the answering processing for input questions contains several steps: preprocessing, entity type tagging, entity mapping, RDF tagging, SPARQL generation, evaluation, and answer filtering. The processing steps of QAWizard are similar to those in Light-QAWizard. The main differences between QAWizard and light-QAWizard are that in the latter, (1) RDF tagging is directly integrated into the next SPARQL generation; (2) RNN is used to answer questions considering the context of the question; (3) the multi-label problem is treated as a multi-class problem using LP to reduce the query cost of QAWizard; and (4) a bidirectional LSTM-CRF (conditional random field), referred to as BiLSTM-CRF [20], is used to label the entities of the input question in order to improve the accuracy of entity tagging.

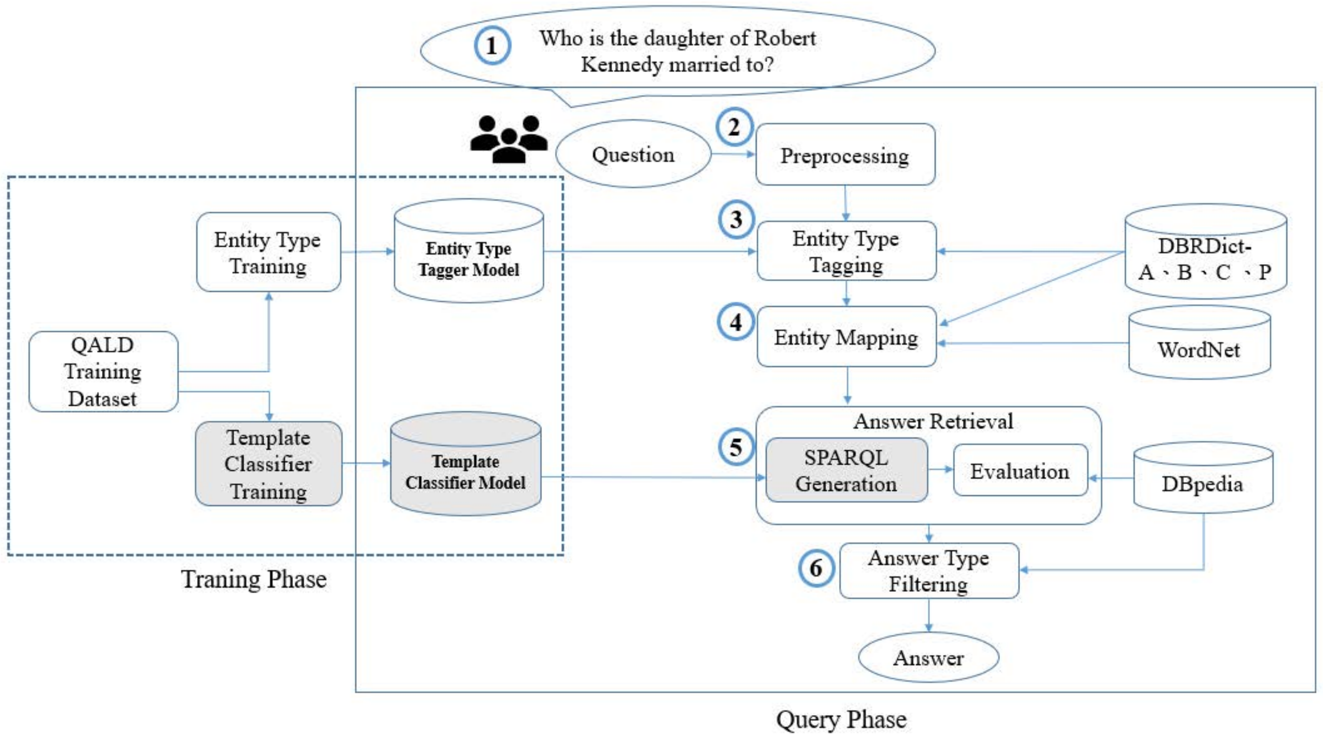


**FIGURE 4.** Architecture of the proposed approach.

## III. THE PROPOSED APPROACH

The system architecture shown in Figure 4 consists of two phases: the training phase and the query phase. During the training phase, using RNN model, we use BiLSTM-CRF [20] as the entity type tagger model to train entity types, and LP as the output of the template classifier model to train appropriate query templates for later generation of SPARQL queries. The RNN model, as shown in Figure 5, consists of four parts: input, processing, dense and output. For the input part, the words in a given question are transformed into a word embedding or multiple embeddings. In the processing part, the RNN is trained by LSTM, GRU and BiLSTM for the first, second and third fully connected layers. The problem can be directed to a template ID after processing and intensive for generating the corresponding RDF queries. The query stage consists of six steps: preprocessing, entity type tagging, entity mapping, SPARQL generation, evaluation, and answer filtering. After a SPARQL query request is sent to DBpedia, the result evaluates whether the answer is valid. Answer type filtering is used to filter the results and return the filtered answer to the user.

### *A. TRAINING PHASE*

As shown in Figure 4, there are two learning models: entity type tagger model and template classifier model. The input QALD training dataset, shown in Figure 6, is a manually labeled dataset based on the QALD dataset, including questions, corresponding SPARQL queries, and answers. The question includes the language encoding, the question, and the keywords in the question, where "query" is the SPARQL query used to request answers from DBpedia, and "answer"

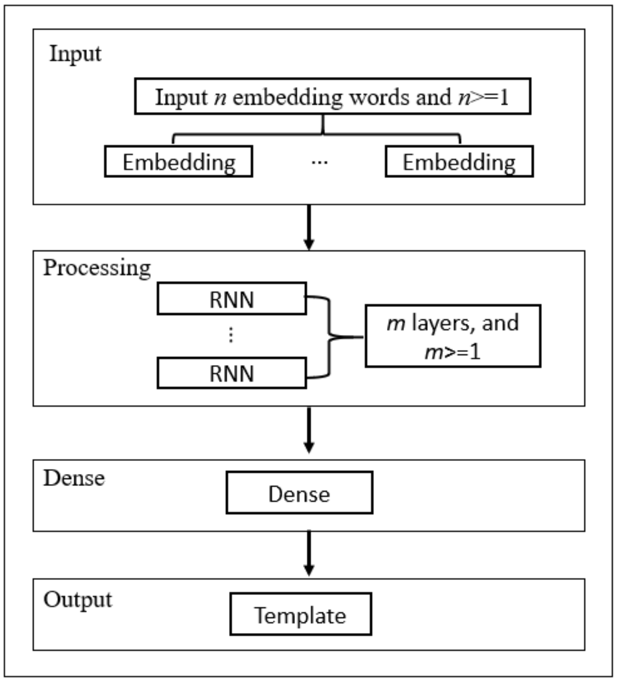


**FIGURE 5.** RNN network architecture.

**TABLE 1.** Precision of various entity type tagging models [20].

| Models | Precision |
|---|---|
| Structured Perceptron | 0.84 |
| MEMM | 0.81 |
| CRF | 0.86 |
| LSTM | 0.79 |
| BiLSTM | 0.84 |
| LSTM-CRF | 0.87 |
| BiLSTM-CRF | 0.91 |

is the answer to the question. The entity type tagger model is used to assign the correct entity type to a keyword or phrase in the question. Based on QALD-7 [10], the accuracy of entity type tagging results is shown in Table 1. Light-QAWizard uses BiLSTM-CRF, which had the highest precision, as the entity type tagging model.

The keywords or phrases $w_i$ from the question are assigned the right entity types through the first model, which can be tagged as E, R, C, V, or N to represent a named entity, relation entity, class entity, variable, or non-useful entity, respectively. Named entity, relation entity and class entity describe $w_i$'s name, property/relation, and class, respectively. If no entities are found for $w_i$, $w_i$ is tagged as N. Since one entity might contain more than one word, the notations -B (Begin), -I (Intermediate), and -E (End) are used to concatenate consecutive words to represent one entity. Only -B is appended to entity types with a slash "/" if the entity is a single word. In the second case, if the entity contains two words, the first and second words are labeled -B and -I, respectively. Otherwise, if the entity contains three words or more, the first and last words are marked -B and -E, respectively, and the remaining words are marked -I. For example, the sentence "List all the musicals with music by Elton John." is tagged as follows:

List/V-B all/N the/N musicals/C-B with/N music/R-B by/N Elton/E-B John/E-I

A SPARQL query consists of RDF triples. RDF triples take the form (subject, predicate, object), and predicates are S, P, and O. In the previous example, the resulting RDF triples are as follows. Here, Elton John is a named entity, described as dbr:Elton_John, Music maps to a class dbo:Musical, and the relationship between dbr:Elton_John and dbo:Musical is the attribute entity dbo:musicBy. When generating SPARQL queries, named entities (E) are placed in Subject or Object, relationships (R) are placed in Predicate, and class entities (C) are placed in Object.

```
SELECT DISTINCT ?ans WHERE {
?ans rdf:type dbo:Musical.
?ans dbo:musicBy dbr:Elton_John.
}
```

In Table 2, the RDF triples designed in Light-QAWizard can be assigned as the following cases. For a given question, the ID A, B, and D can handle a single SPARQL query, and the ID A, B, and C can handle more than one SPARQL query. In some questions, the intermediate points ?x and ?ans can be used to generate RDF triples, where ?x is a bridge connecting two RDF triples and ?ans is the answer to be queried. For example, in the sentence "Who is Robert Kennedy's daughter married to?", the daughter's name, denoted by ?x, can be looked up by the named entity "Robert Kennedy". ?x can be used to find the final answer ?ans. To handle multiple SPARQL query pruning, LP is used to prune unnecessary templates to reduce query cost, as shown below.

a) If ?x appears in every RDF triple, you will never get the answer ?ans. Therefore, ?x cannot be used in all RDF triples.
b) ?x must be used in at least two triples, and one of the RDF triples must contain both ?x and ?ans.
c) An RDF triple is a valid SPARQL query when the triple is described by one of the IDs A, a, B, and b in Table 3.

The pruned results are shown in Table 3, where midpoint ?x is represented by the lowercase letter (such as IDs a, b, and c), and ?ans is represented by the uppercase letter (such as IDs A, B, and C). After pruning unnecessary templates, the filtered templates for two RDF triples and three RDF triples are as shown in Table 4. For example, the generated SPARQL query for the sentence "How many pages are there in War and Peace?" is shown below:

(dbr:War_and_Peace, dbo:numberOfPages, ?ans)

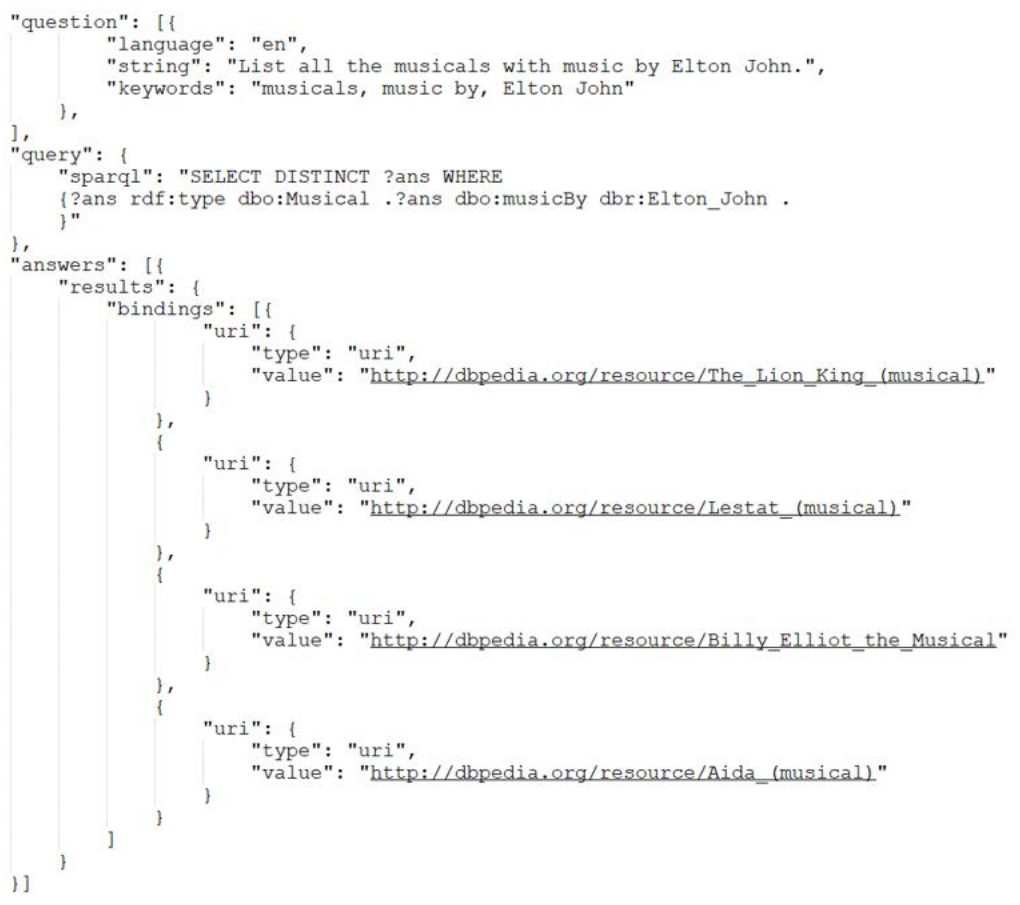
Taking "How many people live in the capital of Australia?" for the ac type, the SPARQL query is generated as follows:

```
SELECT DISTINCT ?ans WHERE {
(dbr:Australia, dbo:capital, ?x .)
(?x, dbo:populationTotal,?ans.)
}
```

The template ID shown in Table 3 is the same as the label defined in the LP. The labels are merged to transform the problem into multi-class classification via LP methods. For example, the merged label appears in the Table 4 as an ID field, as the result of the template classifier, which is used later to generate the corresponding SPARQL queries. The number of labels is reduced from 259 labels to 32 labels. The statistical results are shown in Table 5.

## B. QUERY PHASE

After the user enters a question, the question is parsed and processed in six steps, as shown in Figure 1, including: preprocessing, entity type tagging, entity mapping, answer retrieval, and answer type filtering. The preprocessing step parses the input problem, and includes tokenization, lemmatization, and part-of-speech (POS) tagging. The trained entity tagger is used to assign an entity type to each token in the entity type tagging step. Next, the tokenized token finds the name entity in the entity mapping step. The answer retrieval step generates the corresponding SPARQL query and evaluates the query results. Finally, the query result is filtered



```
"question": [{
        "language": "en",
        "string": "List all the musicals with music by Elton John.",
        "keywords": "musicals, music by, Elton John"
    },
],
"query": {
    "sparql": "SELECT DISTINCT ?ans WHERE
    {?ans rdf:type dbo:Musical .?ans dbo:musicBy dbr:Elton_John .
    }"
},
"answers": [{
    "results": {
        "bindings": [{
                "uri": {
                    "type": "uri",
                    "value": "http://dbpedia.org/resource/The_Lion_King_(musical)"
                }
            },
            {
                "uri": {
                    "type": "uri",
                    "value": "http://dbpedia.org/resource/Lestat_(musical)"
                }
            },
            {
                "uri": {
                    "type": "uri",
                    "value": "http://dbpedia.org/resource/Billy_Elliot_the_Musical"
                }
            },
            {
                "uri": {
                    "type": "uri",
                    "value": "http://dbpedia.org/resource/Aida_(musical)"
                }
            }
        ]
    }
}]
```

**FIGURE 6.** Sample QALD dataset.

**TABLE 2.** Possible RDF triple scenarios.

| ID | RDF triple | Example | SPARQL query |
|---|---|---|---|
| A | (S, P, ?O) | How many pages does War and Peace have? | (dbr:War_and_Peace, dbo:numberOfPages, ?ans) |
| B | (?S, P, O) | Whom did Lance Bass marry? | (?ans, dbo:spouse, dbr:Lance_Bass) |
| C | (?S, P, ?O) | What other books have been written by the author of The Fault in Our Stars? | (dbr:The_Fault_in_Our_Stars, dbo:author, ?x) (?ans, dbo:author, ?x) |
| D | (S, P, O) | Is Michelle Obama the wife of Barack Obama? | (dbr:Barack_Obama, dbo:spouse, dbr:Michelle_Obama) |

**TABLE 3.** Possible scenarios for two RDF triples.

| ID | RDF triple |
|---|---|
| A | (S, P, ?ans) |
| a | (S, P, ?x) |
| B | (?ans, P, O) |
| b | (?x, P, O) |
| C | (?ans, P, ?x) |
| c | (?x, P, ?ans) |

according to the answer type through the answer type filtering step, and the final answer is returned to the user.

### 1) PRE-PROCESSING

According to the input question, the question is divided into several tokens. Afterward, the token is lemmatized and then tokenized with the corresponding POS tag. For example, Table 6 shows the question sentence ‘‘List all musicals with music by Elton John’’ after the preprocessing step.

### 2) ENTITY TYPE TAGGING AND ENTITY MAPPING

Based on the sentence processed by the pre-processing step, we use the trained entity type tagger model to tag the sentence as follows:

List/V-B all/N the/N musicals/C-B with/N music/R-B by/N Elton/E-B John/E-I

Here, the POS tags Verb, None, Named Entity, Class Entities and Relation Entities are labeled V, N, E, C, and R, respectively. The beginning, the intermediate, and the end tokens are denoted as -B, -I, and -E, respectively. When the

**TABLE 4.** Possible scenarios for RDF triples.

| ID | template | ID | template |
|---|---|---|---|
| A | (S, P, ?ans) | bc | (?x, P, O) (?x, P, ?ans) |
| B | (?ans, P, O) | aBC | (S, P, ?x) (?ans, P, O) (?ans, P, ?x) |
| D | (S, P, O) | aBc | (S, P, ?x) (?ans, P, O) (?x, P, ?ans) |
| AA | (S, P, ?ans) (S, P, ?ans) | ABB | (S, P, ?x) (?ans, P, O) (?ans, P, O) |
| AB | (S, P, ?ans) (?ans, P, O) | BBB | (?ans, P, O) (?ans, P, O) (?ans, P, O) |
| aC | (S, P, ?x) (?ans, P, ?x) | Bbc | (?ans, P, O) (?x, P, O) (?x, P, ?ans) |
| ac | (S, P, ?x) (?x, P, ?ans) | bbc | (?x, P, O) (?x, P, O) (?x, P, ?ans) |
| BB | (?ans,P,O) (?ans,P,O) | | |

**TABLE 5.** Possible scenarios for two RDF triples.

| Number of RDF triples | Multi-label | Multi-class |
|---|---|---|
| A triple | 7 | 3 |
| Two triples | 36 | 7 |
| Three triples | 216 | 22 |
| Total | 259 | 32 |

**TABLE 6.** A pre-processing example.

| Token | POS tagger | Lemmatization |
|---|---|---|
| List | NN | List |
| all | PDT | all |
| the | DT | the |
| musicals | NNS | musical |
| with | IN | with |
| music | NN | music |
| by | IN | by |
| Elton | NNP | Elton |
| John | NNP | John |

tagged results of tokens are presented as the same entity type but with different indicator -B and -I, the entity mapping step concatenates the tokens together with "_" as a single token. For example, the tokens Elton and John are both labeled E, so the concatenated result is:

List/V all/N the/N musicals/C with/N music/R by/N Elton_John/E

Only E, C and R are considered in the entity mapping step. As for E, the token is matched to DBRDict-A and DBRDict-B [21], which are the designate and abbreviation indicating the DBpedia repository as shown in Table 7 and Table 8. As for C, the entity look-up from the DBRDict-C [21], which collects the possible class entities and the example, is shown in Table 9. For entity type R, the PATTY [22] is used to build DBRDict-P for relation entities, an example of which is shown in Table 10.

**TABLE 7.** DBRDict-A example.

| Label | URI |
|---|---|
| Elton John | http://dbpedia.org/resource/Elton_John |
| War and Peace | http://dbpedia.org/resource/War_and_Peace |
| United States | http://dbpedia.org/resource//United_States |

**TABLE 8.** DBRDict-B example.

| Label | URI |
|---|---|
| ASUS | http://dbpedia.org/resource/Asus |
| Aena | http://dbpedia.org/resource/ENAIRE |
| Aena | http://dbpedia.org/resource/ENAIRE |

**TABLE 9.** DBRDict-C example.

| Label | URI |
|---|---|
| movie | http://dbpedia.org/ontology/Film |
| automobile | http://dbpedia.org/ontology/Automobile |
| university | http://dbpedia.org/ontology/University |

**TABLE 10.** DBRDict-P example.

| Label | URI |
|---|---|
| be bear at | http://dbpedia.org/ontology/birthPlace |
| marry by | http://dbpedia.org/ontology/spouse |
| be write by | http://dbpedia.org/ontology/author |

### 3) SPARQL GENERATION

According to the template classifier model, the input question can be classified into one of the templates listed in Table 4. Algorithm 1 is the main function to call the corresponding sub-function to generate the SPARQL queries based on the classified results. For example, if the delivered result is BB, the function B is called twice. The entity type E is placed in the position S or O, the R is placed in P position, and C is placed in O position, and P is designated rdf:type.

Take the input question "List all the musicals with music by Elton John." for example. In a template classifier model, the template result is BB, i.e. the SPARQL queries are (?ans, P, O) (?ans, P, O). Thus, Algorithm 1 will call fun_B in Algorithm 2 twice. The fun_B is used to judge whether ERC tag contains C in advance. If it contains C, fun_B generates a RDF triple as (?ans, rdf:type, C); then, it removes the C from the ERC tag to call fun_B again. This time, the call occurs without C so that it processes the remaining tags music/R and Elton_John/E. From left to right, the SPARQL queries are generated as:

(?ans, rdf:type, dbo:Musical)
(?ans, dbo:musicBy, dbr:Elton_John)

When DBpedia is queried with the generated SPARQL queries, the answers are returned as:

**Algorithm 1** Template Matching Algorithm

```
procedure match template:
    case"A":
        fun_A(ERC)
    case"B":
        fun_B(ERC)
    case"D":
        fun_D(ERC)
    case"AA":
        fun_A(ERC)
        fun_A(ERC)
    case"AB":
        fun_A(ERC)
        fun_B(ERC)
    case"aC":
        fun_a(ERC)
        fun_C(ERC)
    case"ac":
        fun_a(ERC)
        fun_c(ERC)
    case"BB":
        fun_B(ERC)
        fun_B(ERC)
    case"bc":
        fun_b(ERC)
        fun_c(ERC)
    case"aBC":
        fun_a(ERC)
        fun_B(ERC)
        fun_C(ERC)
    case"aBc":
        fun_a(ERC)
        fun_B(ERC)
        fun_c(ERC)
    case"ABB":
        fun_A(ERC)
        fun_B(ERC)
        fun_B(ERC)
    case"BBB":
        fun_B(ERC)
        fun_B(ERC)
        fun_B(ERC)
    case"Bbc":
        fun_B(ERC)
        fun_b(ERC)
        fun_c(ERC)
    case"bbc":
        fun_b(ERC)
        fun_b(ERC)
        fun_c(ERC)
```

**Algorithm 2** Slot Filling Algorithm

```
fun_A(ERC)
    choose first name entity from left to right
    S = e + " " + r + "?ans."
    remove E from ERC
    return S
fun_a(ERC)
    choose first name entity from left to right
    S = e + " " + r + "?x."
    remove E from ERC
    return S
fun_B(ERC)
    If ERC tag Contain(C):
        S = "?ans " + "rdf:type " + c + "."
        remove C from ERC
    else:
        choose first name entity from left to right
        S = "?ans " + r + " " + e + "."
        remove E from ERC
    return S
fun_b(ERC)
    If ERC tag Contain(C):
        S = "?ans " + "rdf:type " + c + "."
        remove C from ERC
    else:
        choose first name entity from left to right
        S = "?x " + r + " " + e + "."
        remove E from ERC
    return S
fun_C(ERC)
    S = "?ans " + r + " ?x."
    return S

fun_c(ERC)
    S = "?x " + r + " ?ans."
    return S
fun_D(ERC)
    S = e + " " + r + " " + e + "."
    return S
```

Aida_(musical) The_Lion_King_(musical) Billy_Elliot_the_Musical Lestat_(musical)

The answer type filters the answers according to the word tagged V in the entity type tagger model. The question can be roughly divided into six answer types, such as People/Organization, Place, Time, True/False Question, Returned Value, and Others. If the word is "Who", the answers are kept if the corresponding type of rdf:type are dbo:Person and dbo:Organisation. If the word is "Where", the answers are kept when rdf:type equals dbo:Place. If the word is "When", the answers might be dates or times, so the answers containing xsd:date or xsd:dateTime are kept. If the starting word in the question is "Did" or "Does", the returned result is True if at least one answer is found; otherwise, False. If the starting word in the given question is "How", (e.g., how many), two divided answers are given as: (1) obtained by a property, such as dbo:populationTotal from the question "How many people live in Poland?", and (2) gotten by counting the number of the properties, such as the number of properties dbo:child from the question "How many children does Benjamin Franklin have?". The starting words "Which", "What", "List", "Give" and "Show" belong to the Others category type, which return a name or a collection as the final result. For example, the only result $QT_1$ is the result for the question, "List all the musicals with music by Elton John", as shown in Table 11, including three fields, template number (id), SPARQL queries and query results.

**TABLE 11.** Sample SPARQL queries and their results.

| id | SPARQL queries | query result |
|---|---|---|
| QT$_1$ | (?ans, rdf:type, dbo:Musical) (?ans, dbo:musicBy, dbr:Elton_John) | Aida_(musical) The_Lion_King_(musical) Billy_Elliot_the_Musical Lestat_(musical) |
| QT$_2$ | (?ans, rdf:type, dbo:Musical) (?ans, dbo:producer, dbr:Elton_John) | no result |
| QT$_3$ | (?ans, rdf:type, dbo:Musical) (?ans, dbo:Artist, dbr:Elton_John) | no result |
| QT$_4$ | (?ans, rdf:type, dbo:Musical) (?ans, dbo:associated_band, dbr:Elton_John) | no result |
| QT$_5$ | (?ans, rdf:type, dbo:MusicalWork) (?ans, dbo:musicBy, dbr:Elton_John) | no result |
| QT$_6$ | (?ans, rdf:type, dbo:MusicalWork) (?ans, dbo:producer, dbr:Elton_John) | no result |
| QT$_7$ | (?ans, rdf:type, dbo:MusicalWork) (?ans, dbo:Artist, dbr:Elton_John) | no result |
| QT$_8$ | (?ans, rdf:type, dbo:MusicalWork) (?ans, dbo:associated_band, dbr:Elton_John) | no result |

## IV. EXPERIMENTAL RESULTS

Python was used to implement the system with a Tensorflow [23] network architecture using an RNN deep learning model. The parameters used are listed in Table 12. The study compares the LSTM, GRU, and Bi-LSTM models as well as the parameters for LSTM Layer and LSTM Unit set as 1, 2, 3 and 64, 128, and 256, respectively. BERT, which was proposed by Devlin *et al.* [24], could be used to resolve the synonym problem according to the context between sentences. The POS tag embedding was trained by the data set Treebank [25] to classify the POS Tag, which could help to realize the semantics of natural language in order to increase the precision of the trained model. The embedding size was set to 30 because the number of labels of Part-of-Speech Tags was 36. Window size was the parameter for training POS tag embedding. On average, 7 words needed to be analyzed for the questions QALD-7, QALD-8, and QALD-9; thus, the Window Size was set at 5. Training time was set at 100 epochs to reduce the loss. Batch Size referred to the batch of data. Optimizer Adam was applied to adjust the weights and bias to minimize the loss. We compared the performance when the learning rates were 0.01, 0.05 and 0.001. The evaluation of loss for each learning rate is shown in Figure 7. The result shows that the loss is stable if the learning rate is 0.001. Dropout was used to handle the over fitting problem.

### A. DATA

The question datasets QALD-7 [10], QALD-8 [11] and QALD-9 [12] provided by QALD were used as the experiment datasets. As mentioned in [10], [11], and [12], QAKIS, gAnswer2 and WDAqua were evaluated on QALD-7, QALD-8, QALD-9, and both gAnswer2 and WDAqua won the QALD-7, QALD-8, and QALD-9 challenges. However, the QAKIS, gAnswer2, and WDAqua models were

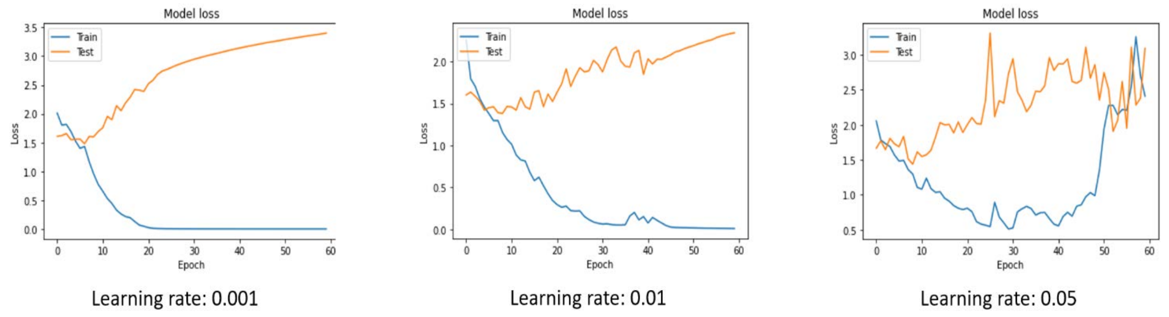


**FIGURE 7.** Loss rate based on different learning rates.

**TABLE 12.** Parameter settings.

| Parameter | Values | Parameter | Value |
|---|---|---|---|
| Model | LSTMGRU Bi-LSTM | Epoch | 100 |
| Layer | 123 | Batch Size | 8 |
| LSTM Unit | 64128256 | Optimizer | Adam |
| Word Embedding | Random GloVeBERT | Learning Rate | 0.001 |
| Pos Embedding window size | 5 | Dropout | 00.050.1 0.150.2 0.250.3 |
| Pos Embedding size | 30 | | |

**TABLE 13.** Number of training sets and testing sets for different datasets.

| | Training sets | Testing sets |
|---|---|---|
| QALD-7 | 183 | 23 |
| QALD-8 | 174 | 33 |
| QALD-9 | 302 | 82 |

all designed to answer simple questions without comparatives, superlatives, or interrogative sentences requiring URIs with sko:Category entities. Therefore, the proposed scheme excludes complex sentences including comparatives and superlatives sentences to preserve the sentences listed in Table 13.

## *B. EXPERIMENTS*

Three metrics, namely Precision, Recall, and F-measure, were used to evaluate the performance of the QA systems, which are defined as follows, where $T(q)$ is the number of answers to questions $q$, $A(q)$ is the number of the correct answers to questions $q$, and $C(q)$ is the number of gold standard answers to questions $q$. F-measure is the harmonic mean of $Precision(q)$ and $Recall(q)$ defined in Equation(3). For example, there are 50 questions, the QA systems can correctly answer 25 questions, and QALD provides 45 gold standard answers. Therefore, the values of $T(q)$, $A(q)$ and $C(q)$ are 50, 25, and 45, respectively. The values of $Precision(q)$, $Recall(q)$ and F-measure$(q)$ are $\frac{25}{50}$, $\frac{25}{45}$ and $\frac{5}{19}$, respectively.

$$Precision(q) = \frac{A(q)}{T(q)} \tag{1}$$

**TABLE 14.** Comparison of tagging results on different test sets based on three different models.

| | QALD-7 | QALD-8 | QALD-9 |
|---|---|---|---|
| LSTM | 69.56% | 81.81% | 69.51% |
| GRU | 65.21% | 81.81% | 68.29% |
| Bi-LSTM | 73.91% | 84.84% | 70.73% |

**TABLE 15.** Comparison of the proposed system with other QA systems on different test sets.

| QALD | System | Precision | Recall | F-measure |
|---|---|---|---|---|
| QALD-7 | QAwizard | 0.59 | 0.59 | 0.59 |
| | gAnswer2 | 0.469 | 0.498 | 0.487 |
| | WDAqua | 0.16 | 0.162 | 0.163 |
| | Ours | 0.565 | 0.652 | 0.594 |
| QALD-8 | QAwizard | 0.375 | 0.3584 | 0.3429 |
| | gAnswer2 | 0.3862 | 0.3902 | 0.388 |
| | WDAqua | 0.3912 | 0.4065 | 0.3872 |
| | Ours | 0.462 | 0.5 | 0.457 |
| QALD-9 | QAwizard | 0.311 | 0.469 | 0.33 |
| | gAnswer2 | 0.261 | 0.267 | 0.250 |
| | WDAqua | 0.293 | 0.327 | 0.298 |
| | Ours | 0.398 | 0.426 | 0.406 |

$$Recall(q) = \frac{A(q)}{C(q)} \tag{2}$$

$$\text{F-measure}(q) = \frac{2 \times Precision(q) \times Recall(q)}{Precision(q) + Recall(q)} \tag{3}$$

To give further detail on the tagging results of different types of QALD, Table 14 shows the precision on QALD-7, QALD-8, and QALD-9 at 73.91%, 84.84% and 70.73%, respectively, when Bi-LSTM is adopted. Table 15 shows different precision based on different models and different data sets, with average precision (Precision), average recall (Recall), and average F-measure (F-measure). The precision scores of Light-QAWizard were 0.565, 0.462 and 0.398, respectively, representing the best precision compared to QAWizard, gAnswer2, and WDAqua. Also, the average recall scores were better than those of gAnswer2 and WDAqua based on three different datasets. The F-measures of Light-QAWizard were 0.594, 0.457 and 0.406 as tested on QALD-7, QALD-8, and QALD-9 datasets, respectively, outperforming those of QAWizard, gAnswer2, and WDAqua.

**TABLE 16.** Comparison of cost computation.

| | Entity | Property | Class | Query Cost |
|---|---|---|---|---|
| gAnswer2 | e | r | c | e$\times r \times c$ |
| WDAqua | e | r | c | (e+r+c)$^2 \times 2 + w$ |
| QAwizard | e | r | c | e$\times r \times c \times 2$ |
| Ours | e | r | c | e$\times r \times c$ |

**TABLE 17.** Query cost comparison.

| Template no | template | ganswer2 | QAwizard | Ours |
|---|---|---|---|---|
| A | (S,P,?ans) | n/2 | n | n/2 |
| B | (?ans,P,O) | n/2 | n | n/2 |
| D | (S,P,O) | n | n | n |
| AA | (S,P,?ans) (S,P,?ans) | n/2 | n | n/2 |
| AB | (S,P,?ans) (?ans,P,O) | n/2 | n | n/2 |
| aC | (S,P,?x) (?ans,P,?x) | n/2 | n | n/2 |
| ac | (S,P,?x) (?x,P,?ans) | n/2 | n | n/2 |
| BB | (?ans,P,O) (?ans,P,O) | n/2 | n | n/2 |
| bc | (?x,P,O) (?x,P,?ans) | n/2 | n | n/2 |
| aBC | (S,P,?x) (?ans,P,O) (?ans,P,?x) | n/2 | n | n/2 |
| aBc | (S,P,?x) (?ans,P,O) (?x,P,?ans) | n/2 | n | n/2 |
| ABB | (S,P,?ans) (?ans,P,O) (?ans,P,O) | n/2 | n | n/2 |
| BBB | (?ans,P,O) (?ans,P,O) (?ans,P,O) | n/2 | n | n/2 |
| Bbc | (?ans,P,O) (?x,P,O) (?x,P,?ans) | n/2 | n | n/2 |
| bbc | (?x,P,O) (?x,P,O) (?x,P,?ans) | n/2 | n | n/2 |

The query costs, listed in Table 16, indicate the search frequency via DBpedia, where the times of name entity, attribute entity, and class entity are denoted by $e$, $r$ and $c$. Note that $w$ is the number of SPARQL queries generated with a distance condition using WDAqua. gAnswer2[3] generates a dependency tree for natural language questions and converts them into a query graph that contains semantic information, finds subgraphs in the graph through the graph knowledge base, and use the subgraphs to generate relative query syntax. gAnswer2 can clearly determine the placement of named entities and attribute entities in the RDF triple, so the query cost is $e \times r \times c$. WDAqua [2] uses N-grams to perform entity comparisons with DBpedia for each word in the question sentence. Each entity is treated as a starting point, a breadth-first search (BFS) at depth 2 is started in DBpedia, and its distance is calculated to generate SPARQL queries. The query cost of WDAqua is $(e+r+c)^2 \times 2+w$. QAWizard [7] contains two stages: entity type tagging and RDF type tagging. The pre-designed templates are used to generate the SPARQL queries. The query cost of QAwizard is $e \times r \times c \times 2$ on average. Based on the query cost calculation in Table 17, the query cost of the QAWizard system method is $n$, the query cost of the gAnswer2 system is $n/2$, and the query cost of the WDAqua system is $5n$. This research method uses the multi-label classification method to reduce the query cost of QAWizard to $n/2$. The advantages of Light-QAWizard are summarized below:

1) Light-QAWizard outperforms QAWizard, gAnswer2, and WDQqua in terms of average precision, recall and F-measure.
2) SPARQL query templates are trained on the QALD-7, QALD-8 and QALD-9 datasets. Thus, only the necessary SPARQL queries are kept, reducing query costs. Light-QAWizard achieves the lowest query cost when compared to QAWizard and WDAqua.

## V. CONCLUSION

A QA system can accurately answer users' questions. SPARQL query generation often drives the query costs, which reflects the frequency of queries to DBpedia. The necessary queries consider the efficiency of answering the question. This paper proposes a classification model and integrates RNN to train a model that can learn from the experiences of picking out suitable SPARQL queries. To reduce query costs, LP is adopted to combine the labels to generate the SPARQL queries. The accuracies on QALD-7, QALD-8 and QALD-9 are 73.91%, 84.84% and 70.73% respectively. The outstanding performance on metrics including precision, recall, F-measure, and query costs, surpass those of all other systems evaluated on the same test sets.

Although the proposed system achieves superior performance, further work should be considered to improve the quality of answers. For example, multilabel classification algorithms, such as binary relevance, classifier chains, and pairwise, could be used to answer complex questions that include comparatives and superlatives. Moreover, the experimental dataset, QALD, is still small, and is therefore limited in its ability to train a model that can sufficiently satisfy almost all types of questions. LC-QuAD [26], a larger dataset, could be used for training to increase the accuracy of the model.

## REFERENCES

[1] K. Xu, S. Zhang, Y. Feng, and D. Zhao, "Answering natural language questions via phrasal semantic parsing," in *Proc. CCF Int. Conf. Natural Lang. Process. Chin. Comput.* Shenzhen, China: Springer, 2014, pp. 333–344.
[2] D. Diefenbach, A. Both, K. Singh, and P. Maret, "Towards a question answering system over the semantic web," *Semantic Web*, vol. 11, no. 3, pp. 421–439, Feb. 2020.
[3] S. Hu, L. Zou, J. X. Yu, H. Wang, and D. Zhao, "Answering natural language questions by subgraph matching over knowledge graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 5, pp. 824–837, May 2017.
[4] J. Daniel and J. H. Martin, *Speech and Language Processing*. Upper Saddle River, NJ, USA: Prentice-Hall, 2009.

[5] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann, "DBpedia—A crystallization point for the web of data," *J. Web Semantics*, vol. 7, no. 3, pp. 154–165, 2009.

[6] *Sparql 1.1 Overview*, World Wide Web Consortium, Cambridge, MA, USA, 2013.

[7] Y.-H. Chen, E. J.-L. Lu, and T.-A. Ou, "Intelligent SPARQL query generation for natural language processing systems," *IEEE Access*, vol. 9, pp. 158638–158650, 2021.

[8] P. Ochieng, "PAROT: Translating natural language to SPARQL," *Exp. Syst. Appl., X*, vol. 5, 2020, Art. no. 100024.

[9] T. Soru, E. Marx, D. Moussallem, G. Publio, A. Valdestilhas, D. Esteves, and C. B. Neto, "SPARQL as a foreign language," 2017, *arXiv:1708.07624*.

[10] R. Usbeck, A.-C. N. Ngomo, B. Haarmann, A. Krithara, M. Röder, and G. Napolitano, "7th open challenge on question answering over linked data (QALD-7)," in *Semantic Web Challenges*. Vienna, Austria: Springer, 2017, pp. 59–69.

[11] R. Usbeck, A.-C. N. Ngomo, F. Conrads, M. Röder, and G. Napolitano, "8th challenge on question answering over linked data (QALD-8)," *Language*, vol. 7, no. 1, pp. 51–57, 2018.

[12] N. Ngomo, "9th challenge on question answering over linked data (QALD-9)," *Language*, vol. 7, no. 1, pp. 58–64, 2018.

[13] X. Yin, D. Gromann, and S. Rudolph, "Neural machine translating from natural language to SPARQL," *Future Gener. Comput. Syst.*, vol. 117, pp. 510–519, Apr. 2021.

[14] E. Gibaja and S. Ventura, "A tutorial on multilabel learning," *ACM Comput. Surv.*, vol. 47, no. 3, pp. 1–38, 2015.

[15] A. K. McCallum, "Multi-label text classification with a mixture model trained by EM," in *Proc. AAAI Workshop Text Learn.*, 1999, pp. 1–7.

[16] J. Liu, W.-C. Chang, Y. Wu, and Y. Yang, "Deep learning for extreme multi-label text classification," in *Proc. 40th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, 2017, pp. 115–124.

[17] C. Dima, "Answering natural language questions with intui3," in *Proc. CLEF Work. Notes*, 2014, pp. 1201–1211.

[18] A. Freitas and E. Curry, "Natural language queries over heterogeneous linked data graphs: A distributional-compositional semantics approach," in *Proc. 19th Int. Conf. Intell. User Interfaces*, 2014, pp. 279–288.

[19] A. Freitas, J. A. G. Oliveira, S. O'riain, J. C. D. Silva, and E. Curry, "Querying linked data graphs using semantic relatedness: A vocabulary independent approach," *Data Knowl. Eng.*, vol. 88, pp. 126–141, Nov. 2013.

[20] C.-Y. Kuo and E. J.-L. Lu, "A BiLSTM-CRF entity type tagger for question answering system," in *Proc. IEEE Int. Conf. Ind. Artif. Intell., Commun. Technol. (IAICT)*, Jul. 2021, pp. 161–166.

[21] J. Liu, W. Li, L. Luo, J. Zhou, X. Han, and J. Shi, "Linked open data query based on natural language," *Chin. J. Electron.*, vol. 26, no. 2, pp. 230–235, 2017.

[22] N. Nakashole, G. Weikum, and F. Suchanek, "PATTY: A taxonomy of relational patterns with semantic types," in *Proc. Joint Conf. Empirical Methods Natural Lang. Process. Comput. Natural Lang. Learn.*, 2012, pp. 1135–1145.

[23] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, and S. Ghemawat, "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," 2016, *arXiv:1603.04467*.

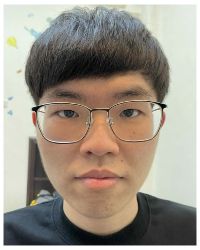
[24] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.

[25] M. A. Marcinkiewicz, "Building a large annotated corpus of English: The Penn treebank," in *Using Large Corpora*, vol. 273. Cambridge, MA, USA: MIT Press, 1994.

[26] P. Trivedi, G. Maheshwari, M. Dubey, and J. Lehmann, "LC-QuAD: A corpus for complex question answering over knowledge graphs," in *Proc. Int. Semantic Web Conf.* Vienna, Austria: Springer, 2017, pp. 210–218.

**YI-HUI CHEN** (Member, IEEE) received the Ph.D. degree in computer science and information engineering from the National Chung Cheng University. Later on, she worked at Academia Sinica as a Postdoctoral Fellow. Later, she worked at IBM's Taiwan Collaboratory Research Center as a Research Scientist. After that, she worked at the Department of M-Commerce and Multimedia Applications, Asia University. She is currently an Associate Professor with the Department of Information Management, Chang Gung University. Her research interests include data mining, semantic analysis, and multimedia security.

**ERIC JUI-LIN LU** (Member, IEEE) received the Ph.D. degree in computer science from the Missouri University of Science and Technology (formerly the University of Missouri–Rolla), USA, in 1996. He is currently a Professor with National Chung Hsing University. His research interests include machine learning, natural language processing, and semantic web.



**YING-YEN LIN** received the M.S. degree from Department of Management Information Systems, National Chung Hsing University. He is currently working for Trend Micro Inc. His research interests include semantic web and natural language processing.