IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 7 July 2023, accepted 20 August 2023, date of publication 25 August 2023, date of current version 30 August 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3308691

RESEARCH ARTICLE

Enhancing SPARQL Query Performance With Recurrent Neural Networks

YI-HUI CHEN[1,2], (Member, IEEE), ERIC JUI-LIN LU[3], (Member, IEEE), AND JIN-DE LIN[3]

[1]Department of Information Management, Chang Gung University, Taoyuan 33302, Taiwan
[2]Kawasaki Disease Center, Kaohsiung Chang Gung Memorial Hospital, Kaohsiung City 83301, Taiwan
[3]Department of Management Information Systems, National Chung Hsing University, Taichung City 402204, Taiwan

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

This work 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 CMRPD3N0011.

ABSTRACT DBpedia is one of the most resourceful link databases today, and users need to use query syntax (e.g., SPARQL) to access information in DBpedia databases. However, not all users know SPARQL, so a natural language query system can be used to translate the user's query into the corresponding syntax. Generating query syntax through the query system is both time-consuming and expensive. To improve the efficiency of query syntax generation from user questions, the multi-label template approach, specifically Light-QAwizard, is utilized. Light-QAwizard transforms the problem into one or more single-label classifications using multi-label learning template approaches. By implementing Light-QAwizard, query costs can be reduced by 50%, but it introduces a new label during the transformation process leading to sample imbalance, compromised accuracy, and limited scalability. To overcome these limitations, this paper employs two multi-label learning methods, Binary Relevance (BR) and Classifier Chains (CC), for question transformation. By employing Recurrent Neural Networks (RNNs) as a multi-label classifier for generating RDF (Resource Description Framework) triples, all the labels are predicted to align with the query intentions. To better account for the relationship between RDF triples, BR is integrated into an ensemble learning approach to result in the Ensemble BR. Experimental results demonstrate that our proposed method outperforms previous research in terms of improving query accuracy. The favorable experiments substantiate that the Ensemble BR or CC model demonstrates competitiveness by integrating label relationships into the trained model.

INDEX TERMS DBPedia, SPARQL, question answering systems, multi-label classifier, recurrent neural network (RNN), binary relevance (BR), classifier chains (CC), resource description framework (RDF).

I. INTRODUCTION

Question Answering (QA) systems have been introduced as technology has progressed. Designing a QA system is a challenging task that involves utilizing Natural Language Processing (NLP) techniques to help users promptly find specific and accurate answers to their queries. Abbasiantaeb and Momtazi [1] classified QA systems into Text-based and Knowledge Graph-based (KG) approaches. The Text-based QA system [2] uses Information Retrieval (IR) techniques to process the questions. It breaks them down into indexing words or keywords to identify the most relevant answers. Then, the answer set is sorted and provided to the user, similar to popular search engines such as Google, Yahoo, and Askme. The Knowledge Graph-based Question Answering (KGQA) System uses Semantic Parsing (SP) to comprehend the question, and it creates a query syntax (e.g., SPARQL) that aligns the semantics of the question. Then, this query syntax is used to retrieve the answer from the knowledge base. The study [3] employs IR and KG-based techniques to gain insights into how textual and structured information behave in the ranking of scholarly articles.

The associate editor coordinating the review of this manuscript and approving it for publication was Liangxiu Han.

Many open knowledge bases are available, such as DBpedia [4], Wikidata [5], and YAGO [6], and they often utilize the Resource Description Framework (RDF) format [7]. RDF is a technical specification for a markup language proposed by W3C, and it consists of subjects, predicates, and objects. The subject and the object represent an entity, and the predicate represents the relationship between two entities. For instance, the answer to the question ''Is camel of the chordate phylum?'' can be represented as an RDF triple, such as (camel, phylum, chordate). KGQA systems provide more accurate answers than Text-Based QA systems because they store data in a structured format [8]. For example, when answering the question, ''Count the total number of mammals whose taxonomy is Diprotodontia and phylum is Chordate.'' The keywords in this question are ''mammals,'' ''taxonomy,'' ''Diprotodontia,'' ''phylum,'' and ''Chordate.''

The traditional IR approach cannot understand the logical connections between entities. It can only retrieve all the keywords in the question to obtain potential answers. The set is sorted according to the query formula, and the top-ranking results are returned. However, SP can perform advanced operations using the knowledge graph as follows.

- Multi-hop reasoning: The question may include multiple facts, such as the relationship between ''Diprotodontias'' and ''Chordates,'' which can be represented as (?ans, taxonomy, Diprotodontia), (?ans, phylum, Chordate).
- Constrained relations: Consideration must be given to the limitation specified in the question, which states that the entity class must be ''mammal,'' or in other words, (?ans, is-a, mammal).
- Numerical operations: When answering the question, one should be prepared to count or sort because the question may include phrases like ''Count the total number of,'' which would require determining the exact number of answers.

KGQA systems typically are divided into four steps, i.e., data preprocessing, entity mapping, query generation, and evaluation. With the growth of machine learning, many current QA systems utilize machine learning techniques to aid in the preprocessing, query generation, and evaluation processes. The main objective of the preprocessing step is to identify essential words in the question, while the entity mapping step aims to find significant words in the knowledge base. The query generation step creates a structured question representation, such as SPARQL. The evaluation stage uses the generated query syntax to retrieve answers from the knowledge base and provides them to the user. Currently, research on query generation syntax can be classified broadly into two categories [9], i.e., (1) template-free approaches [10], [11], [12], [13], which involve constructing SPARQL query syntax by utilizing the syntactic structure of question sentences, and (2) template-based approaches [14], [15], which involve mapping question sentences to manually or automatically create SPARQL query templates.

There are three main categories of template-free approaches, i.e., using knowledge graph structures [11], dependency structures [10], and machine translation [12], [13]. The knowledge graph structure approach [11] utilizes a knowledge graph structure and a subgraph search algorithm to find all possible RDF triples and their matching subgraphs in the entity mapping step. The dependency structure method [10] involves constructing a Directed Acyclic Graph (DAG) to identify the relationships between entities, which are then used to generate the query syntax. The approach using knowledge graph structures and dependency structures may incur additional costs to ensure that all possible basic graph patterns are searched. Machine translation [12], [13] translates the query as a sequence-to-sequence task to convert the query into syntax through machine learning directly. However, due to the large number of entities in DBpedia, it can be challenging to generate correct query syntax for different domains, and the architecture of machine translation might be challenging to design and interpret.

Two main template-based approaches, namely the Basic template [14] and the multi-label template [15]. The Basic template is pre-determined based on the fundamental rules of RDF triples, and resources obtained from entity matching are inserted into the template to generate the query syntax. The multi-label template approach utilizes the problem transformation method in multi-label learning [15] to convert multi-label problems into one or more single-label classification algorithms. RDF triples in SPARQL are treated as labels designed based on the problem transformation method, with essential words inserted according to the entity types to generate query syntax.

QAwizard [14] is the multi-label template approach, which performs well in Precision, Recall, and F-measure metrics. However, it struggles to determine the placement of entities in terms of subject or object during the Entity Mapping step. This results in a costly process of generating query syntax due to the many SPARQLs generated. Applying the Label Powerset (LP) in multi-label learning, known as Light-QAwizard [15], can reduce query cost by 50%, but it creates a new label for all combinations of RDF triples, which leads to several problems. First, it can cause sample imbalance. Second, if the test data contain RDF triples that were not present in the training data, it cannot generate the correct query. Thirdly, the number of labels increases exponentially with the number of RDF triples, and finally, LP has poor scalability.

To address these issues, the paper adopts the approach outlined in Gibaja and Ventura [16]. Their research emphasizes the effectiveness of Binary Relevance (BR) and Classifier Chains (CC) in text categorization for problem transformation. Therefore, this paper designs SPARQL tags based on these two methods and constructs a model using the Recursive Neural Network (RNN). The results of the experiment indicated that BR and CC methods perform well, with BR achieving accuracies of 82.6%, 93.94%, 73.17%, and 62.2% for QALD-7, QALD-8, QALD-9, and LC-QuAD,

respectively. Question Answering over Linked Data (QALD) and Large-Scale Complex Question Answering Dataset (LC-QuAD) are the well-known benchmarks in the semantic web, a series of evaluation competitions for QA systems applied to knowledge bases. So far, nine competitions have been conducted. In contrast, CC achieved accuracies of 78.26%, 93.94%, 72.29%, and 60.2%, respectively.

The BR method demonstrates high accuracy across all data sets. However, it does not consider the correlation between labels, making it difficult to predict SPARQL queries involving multiple RDF triples. To address this issue, we propose the Ensemble BR framework, which utilizes the concept of Ensemble Learning to train the correlation between labels. The proposed method demonstrates effectiveness through experiments, with accuracy rates of 82.6%, 93.94%, 76.82%, and 76.1% for QALD-7, QALD-8, QALD-9, and LC-QuAD, respectively. Furthermore, compared to other End-to-End systems, this system achieves superior results in QALD metrics, such as Precision, Recall, and F-measure. Below are the contributions of the proposed method:

- Improved efficiency: The proposed method enhances the efficiency of query syntax generation in template-free approaches by employing the multi-label template approach.
- Addressing sample imbalance and compromised accuracy: The method utilizes BR and CC as problem transformations in text categorization to resolve issues related to sample imbalance, compromised accuracy, and limited scalability caused by Light-QAwizard.
- Consideration of label correlations: The ensemble BR framework trains the correlation between labels, allowing for the prediction of SPARQL queries involving multiple RDF triples while considering the correlation between labels.
- Enhanced performance and scalability: By incorporating problem transformations and considering label correlations, the proposed method aims to improve the overall performance and scalability of the system.

The structure of this paper is as follows. Section II presents a detailed discussion of multi-tag learning applied to text classification and past. In Section III, this study proposes a research method and process for applying multi-tag learning to generate query syntax. Section IV presents a complete introduction to this system's query process. In Section V of this paper, the experimental results of multi-tag classification are analyzed and compared with other End-to-End systems.

II. RELATED LITERATURES

Our research team initially introduced QAwizard [14] as an approach to tackle the multiclass classification problem. It utilized a two-stage maximum-entropy Markov model to learn from human experiences in entity-type identification and RDF-type identification. However, the generation of SPARQL queries incurred high query costs. To address this, the Light-QAwizard [15] integrates multi-label classification into an RNN. This integration reduces the frequency of queries to DBpedia by aggregating multiple outputs into a single output through multi-label classification.

To minimize the query cost and enhance search accuracy, the proposed method employs multi-tag templates for SPARQL generation. Each set of RDF triples in SPARQL is treated as a tag, and each natural language query is associated with its corresponding tag set. By doing so, we aim to reduce the query cost while ensuring accurate search results. This section provides insights into applying multi-label learning to text classification, specifically for generating query syntax in KGQA.

A. MULTI-TAG LEARNING FOR TEXT CLASSIFICATION

The classification task is a fundamental aspect of machine learning. As machine learning continues to evolve, the demand for classification techniques increases, as seen in various examples, such as Netflix's movie collection. The age classification of the collection, for instance, may be divided into multiple categories, such as "all ages," "18+," "16+," "13+," and "7+," which is a multi-class classification task. In addition, the collection's labels may include "Japanese," "family film," "anime film," and "emotion," and this is referred to as a multi-class classification.

Multi-class classification is a common task in machine learning, where a model is trained to predict unseen instances based on multiple labeled samples. Traditional supervised learning typically focuses on single-label classification tasks, such as binary and multi-class classification, but, in reality, many textual tasks are not labeled uniquely. To address this issue, many researchers have studied multi-label classification, a method in which an object is given a set of appropriate category labels to express its meaning.

Multi-label learning algorithms can be classified broadly into two categories, i.e., problem transformation and algorithm adaptation [16], [17]. Problem transformation transforms the problem into a single label (e.g., LP) or multiple single labels (e.g., BR and CC). Algorithm adaptation involves modifying an existing algorithm to make it suitable for multi-label classification. The Light-QAwizard utilizes LP to convert the problem into a single label, thereby reducing half of the QAwizard's cost. Nevertheless, Light-QAwizard is plagued by issues such as sample imbalance, compromised accuracy, and limited scalability.

B. PROBLEM TRANSFORMATION METHOD

This section draws on the work of Gibaja and Ventura [16] to provide a basic definition of multi-tag learning, where X represents the input space and $Y = \{y_1, y_2, \ldots, y_q\}$ represents the output space comprising class q. The number of classes, q, must be greater than or equal to 1, meaning the tags cannot be empty. Table 1 presents an example of a multi-label dataset with 5 instances (X_1, X_2, X_3, X_4, X_5) and 4 labels y_i, where $y_i \subseteq Y = \{y_1, y_2, y_3, y_4\}$.

TABLE 1. Multi-label dataset example.

Examples	Multi-label outputs	LP transformation y
X_1	$\{y_2, y_3, y_4\}$	A
X_2	$\{y_1, y_3, y_4\}$	B
X_3	$\{y_2, y_3\}$	C
X_4	$\{y_1, y_4\}$	D
X_5	$\{y_1, y_4\}$	D

$\mathcal{D}_1$

EX	y_1
X_1	0
X_2	1
X_3	0
X_4	1
X_5	1

$\mathcal{D}_2$

EX	y_2
X_1	1
X_2	0
X_3	1
X_4	0
X_5	0

$\mathcal{D}_3$

EX	y_3
X_1	1
X_2	1
X_3	1
X_4	0
X_5	0

$\mathcal{D}_4$

EX	y_4
X_1	1
X_2	1
X_3	0
X_4	1
X_5	1

FIGURE 1. BR Transformation.

Given a multi-label training set $D = \{(X_i, Y_i)\ 1 \leq i \leq m\}$, where $Y_i \subseteq Y$ is a set of labels corresponding to X_i, and m represents the size of this training set. The task of Multi-label Classification (MLC) is to learn a classifier $h : X \rightarrow 2^q$, where X corresponds to a q-dimensional binary vector $\{0, 1\}^q$. The h function maps X_i to its label Y_i. Given an input, X_i, the classifier returns the set of labels, Y_i, associated with that input, i.e., $h(X_i) \subseteq Y_i$. Multiclass classification (MC) can be viewed as a special case of MLC in which the result of g prediction is a subset of labels from the set Y (i.e., $g : X \rightarrow Y$). Here, the g function of MC classifies X_i to a single label $Y_i \in Y$.

C. BINARY RELEVANCE (BR)

BR transforms a multi-label learning problem into q separate binary classification tasks, each corresponding to a label in the label set (Y) as shown in Table 1. Since there are four different labels in the label set (y_1, y_2, y_3, y_4), i.e., $q = 4$, BR transforms the data set sequentially into four binary pieces of training corresponding to the labels (D_1, D_2, D_3, D_4). While building the training set D_1, we check whether the set of labels in the training instance satisfies y_1. If so, mark it as 1; otherwise, when we create the training set, D_2, we check whether the set of tags in the training instance contains y_2, and so on. Take X_1 for example, the set of labels for X_1 is $\{y_2, y_3, y_4\}$. The four-stroke binary training set has the following format: $(D_1(X_1, y_1 = 0))$, $(D_2(X_1, y_2 = 1))$, $(D_3(X_1, y_3 = 1))$, $(D_4(X_1, y_4 = 1))$ of the data, and the results are shown in Figure 1. This approach is similar to the One-Versus-All (OVA) approach in that the model is given an input instance, and the output is a collection of related labels. The difference between using a binary classifier to solve multi-class problems and a traditional binary classifier is that the traditional binary classifier has only one relevant label in one instance. After constructing the training set, BR uses binary learning (B). After constructing the training set, BR uses the binary learning algorithm to train the binary classifier $B(D_j)(1 \leq j \leq q) \rightarrow B_j$.

Zhou et al. [18] highlighted three primary limitations of BR. Firstly, it assumes label independence, disregarding label correlation. Secondly, it can encounter sample imbalance issues in cases of low label density. Lastly, it necessitates training numerous classifiers when dealing with a large number of labels (q). However, BR has several advantages, such as its simplicity and adaptability to changing scenarios. In addition, since each label is independent, adding or removing labels without affecting other label models is easy, making it well-suited for changing scenarios. In the example, even though the label combination of $\{y_1, y_2\}$ does not appear in the training data, BR still has a chance to predict it utilizing the predictions of classifiers D_1, D_2, D_3, and D_4. That is, when handling the instance X, even though the label combination of $\{y_1, y_2\}$ is not present in the training data, BR still allows for the possibility of predicting $\{y_1, y_2\}$ by using the classifiers of D_1, D_2, D_3, and D_4 to potentially predict 1, 1, 0, and 0, respectively.

D. CLASSIFIER CHAINS (CC)

Despite its advantages, the main drawback of BR is that it cannot utilize the correlation between labels to improve the performance of the classifier. CC incorporates the correlation between labels into the binary classifier to address this issue. We transform the multi-label data set in Table 1, since there are four different labels in the label set (y_1, y_2, y_3, y_4), i.e., $q = 4$. Hence, BR transforms the data set into four binary data sets (D_1, D_2, D_3, D_4). We create training sets for each label in order, starting from the second binary data set except for the first binary data set D_1 (i.e., D_2, D_3, D_4) will add the previous labels to the training data. Taking X_1 as an example, y_1 is 0 when D_1 is created, so the training data when D_2 is created (X_1,[0]); y_2 is 1 when building D_2, so the training data in building D_3 is (X_1, [0, 1]); in building D_3, y_3 is 1, so the training data in building D_4 is (X_1, [0, 1, 1]), and the results are shown in Figure 2. After constructing the training set, CC will use the binary learning algorithm to train the binary classifier $B(D_j)(1 \leq j \leq q) \rightarrow B_j$.

$\mathcal{D}_1$

EX	y_1
X_1	0
X_2	1
X_3	0
X_4	1
X_5	1

$\mathcal{D}_2$

EX	y_2
X_1,[0]	1
X_2,[1]	0
X_3,[0]	1
X_4,[1]	0
X_5,[1]	0

$\mathcal{D}_3$

EX	y_3
X_1,[0,1]	1
X_2,[1,0]	1
X_3,[0,1]	1
X_4,[1,0]	0
X_5,[1,0]	0

$\mathcal{D}_4$

EX	y_4
X_1,[0,1,1]	1
X_2,[1,0,1]	1
X_3,[0,1,1]	0
X_4,[1,0,0]	1
X_5,[1,0,0]	1

FIGURE 2. CC Transformation.

The CC approach improves the generalization of some binary classifiers. Still, it has a significant drawback, i.e., if a previous classifier makes a prediction error, it will affect the later classifiers along the chain. This effect is more pronounced when errors occur early [19]. In addition, due to its linking property, CC loses the ability to add or remove labels dynamically, which BR can do easily.

III. PROPOSED SCHEME

The system depicted in Figure 3 comprises two phases: the training phase and the query phase. The training phase

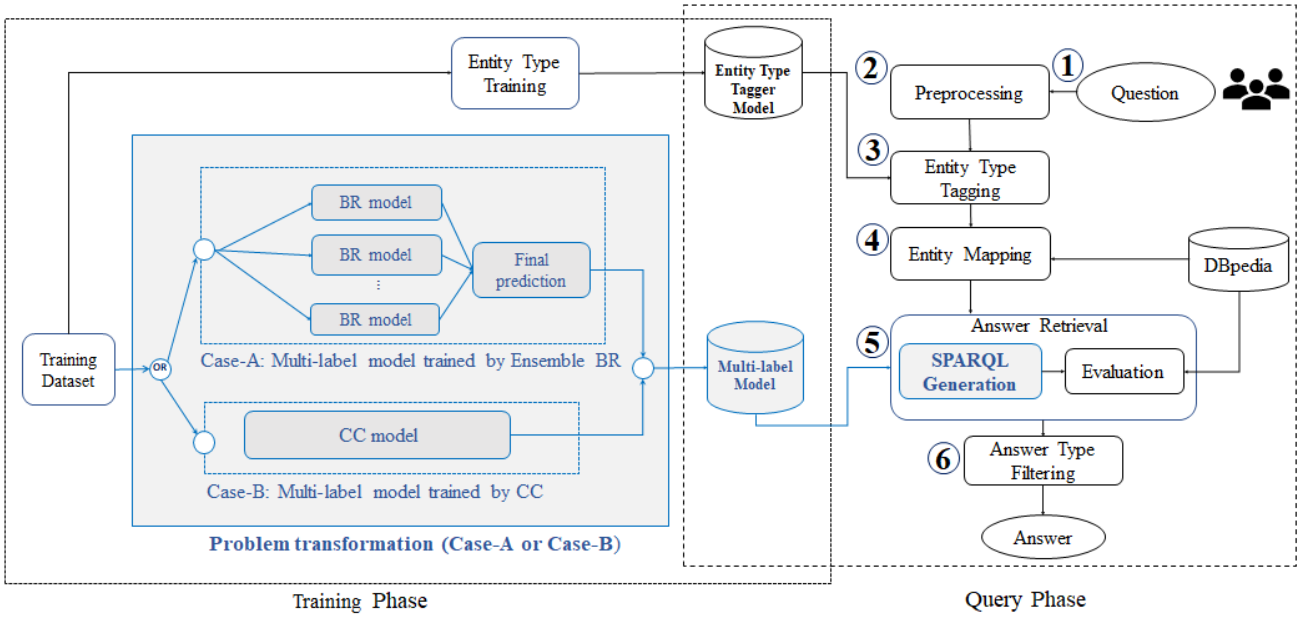

FIGURE 3. Architecture of the proposed approach.

is based on two learning models, i.e., the Entity Type Tagger and the Multi-label Model. Compared to previous works [14], [15], the distinguishing aspects are represented in background gray colors. Specifically, this includes the development of a multi-label model for the subsequent SPARQL generation phase. Two problem transformations, Case-A and Case-B, are trained independently, and one of them is selected as the adopted multi-label model.

The proposed scheme uses both QALD [20], [21], [22] and LC-QuAD [23] datasets as the training data, as illustrated in Figure 4. Each set of questions in these datasets includes various information, such as the language used in the question, the question string, keywords, the SPARQL query that matches the question's semantics, and the corresponding answers. The detailed contents of the multi-label model are outlined in Section III-A.


```
"question" : [ {
   "language" : "en",
   "string" : "In which films did Julia Roberts as well as Richard Gere play?",
   "keywords" : "film, play, Julia Roberts, Richard Gere" },
   "query" : {
      "sparql" :   "SELECT DISTINCT ?ans WHERE {
                        ?ans rdf:type dbo:Film .
                        ?ans dbo:starring dbr:Julia_Roberts .
                        ?ans dbo:starring dbr:Richard_Gere . }",
   "answers" : [ {
      "results" : {
         "bindings" : [ {
            "uri" : {
            "type" : "uri",
            "value" : "http://dbpedia.org/resource/Runaway_Bride_(film)" }
         } ]
      }
   } ]
```


FIGURE 4. QALD dataset example.

A. MULTI-LABEL MODEL

Each SPARQL RDF triple is considered to be a tag, and the structure and rules of SPARQL are analyzed to create templates. We attempt to use the problem transformation methods of BR and CC, treating the q labels as q binary problems to transform the SPARQL generation problems into multiple binary problems, making it easier to build binary classification models utilizing RNNs. The proposed model, namely Ensemble BR model (EBR model for short), considers the tags' correlation.

As stated in the definition of SPARQL [7], it comprises mainly RDF triples. The example in Figure 4 clearly illustrates the SPARQL syntax, consisting of three RDF triples. An RDF triple can be seen as a kind of label. Finding all potential triples, the templates for RDF triples can be simplified through the following process to prevent rapid growth in the number of tags.

- The simplification of "?ans rdf:type dbo:Film" to "?S P O" involves replacing "rdf:type" with "P (i.e., Predicate)" and "dbo:Film" with "O (i.e., Object)", with "?S (i.e., Subject)" representing the variable in the subject position.
- Similarly, the RDF triples "?ans dbo:starring dbr:Julia_Roberts" and "?ans dbo:starring dbr:Richard_Gere" also are simplified to the simple form (?S P O).

Figure 5 demonstrates a single RDF triple consisting of S, P, and O; thus, a maximum of 8 (i.e., 2^3)

No.	Original RDF triple	One RDF triple		Two RDF triples	
		Changed RDF triple	Label	Changed RDF triples	Label
1	(S, P, ?O)	(S, P, ?ans)	A	(S, P, ?ans) (S, P, ?x)	A a
2	(?S, P, O)	(?ans, P, O)	B	(?ans, P, O) (?x, P, O)	B b
3	(?S, P, ?O)	-	-	(?ans, P, ?x) (?x, P, ?ans)	C c
4	(S, P, O)	(S, P, O)	D	-	-
5	(S, ?P, O)	-	-	-	-
6	(?S，?P, O)	-	-	-	-
7	(S, ?P, ?O)	-	-	-	-
8	(?S, ?P, ?O)	-	-	-	-

FIGURE 5. Labeling original RDF triple, one RDF triple, and two RDF triples.

combinations exist in a SPARQL sentence. For an RDF triple labeled by B, for example, in the triple (?S, dbo:starring, dbr:Julia_Roberts), P and O are represented by dbo:starring and dbr:Julia_Roberts respectively. This query looks up all the movies in which Julia Roberts has starred.

The following scenarios of SPARQL syntax were listed based on the QALD [20], [21], [22] and LC-QuAD [23] datasets.

- If SPARQL contains only one RDF triple, Subject and Object cannot be variables simultaneously (as demonstrated in scenario 3).
- The scenario, in which all three elements, Subject, Predicate, and Object, are known (e.g., in scenario 4), happens when only one RDF triple exists in SPARQL.
- The answer is in the position of Subject or Object only when Predicate is known (as indicated in scenarios 6, and 7 are not reasonable triples).
- Subject, Predicate, and Object cannot be variables simultaneously. As a result, the RDF triple pattern represented by scenario 8 is not considered.
- In SPARQL, all unknown variables cannot be designated as ?x because the purpose of ?x is to find the answer, represented by ?ans.
- If an RDF triple in SPARQL contains the variable ?x, the ?x must occur along with the variable ?ans in one of the triples to find the answer.

According to the following arrangement, if SPARQL has only one set of RDF triples, the scenarios are labeled with numbers 1, 2, and 4. The labeled numbers 1, 2, and 3 may appear in the SPARQL syntax of more than two sets of RDF triples.

To address converting SPARQL to a multi-tagged data set, we substitute the unknown variable to ?ans when SPARQL has only one RDF triple as shown in Figure 5. Take (S, P, ?O) for example, we replace the unknown variable "?O" with "?ans" to obtain (S, P, ?ans), represented by Label (A). Similarly, the RDF triple (?S, P, O) is replaced with (?ans, P, O),denoted as Label (B). But, since the RDF triple (S, P, O), it is directly labeled as Label (D).

In RDF triples, the variables with unknown values (?S and ?O) commonly are designated as ?ans or ?x. ?ans represents the answer to the question while ?x represents an answer that cannot be linked directly to the entity in the question and requires intermediate query resolution. For example, for the question "Where did the architect of the Eiffel Tower study?", we must first link to the architect "Stephen Sauvestre (?x)" through the Eiffel Tower, and then determine the answer through "Stephen Sauvestre (?x)".

In SPARQL with multiple sets of RDF triples, the RDF triples can be of the forms shown in numbers 1, 2, and 3 in Figure 5. To address the issue of intermediate queries, we designate Label (A, a) for the RDF triple (S, P, ?O) where ?O is replaced by ?ans and ?x, Label (B, b) for (?S, P, O) where ?S is replaced by ?ans and ?x, and Label (C, c) for (S, P, O), where ?S and ?O are replaced by ?ans and ?x, respectively, with the positions of ?ans and ?x switched.

For multiple RDF triples, to distinguish the template of RDF triple is ?x or ?ans, different labels are assigned as shown in Figure 5. Here, the RDF triple (S, P, ?O) is replaced with (S, P, ?ans) or (S, P, ?x), represented by Label A or a, respectively. Also, labels B and b are the cases (?ans, P, O) and (?x, P, O), respectively. Denoted by C or c if the (?S, P, ?O) is substituted by (?ans, P, ?x) or (?x, P, ?ans), respectively.

First Triple		Second Triple		Third Triple	
Label	RDF triple	Label	RDF triple	Label	RDF triple
A	(S，P，?ans)	E	(S，P，?ans)	H	(S，P，?ans)
a	(S，P，?x)	e	(S，P，?x)	h	(S，P，?x)
B	(?ans，P，O)	F	(?ans，P，O)	I	(?ans，P，O)
b	(?x，P，O)	f	(?x，P，O)	i	(?x，P，O)
C	(?ans，P，?x)	G	(?ans，P，?x)	J	(?ans，P，?x)
c	(?x，P，?ans)	g	(?x，P，?ans)	j	(?x，P，?ans)

FIGURE 6. Sub-label used in more than two RDF triples.

Extending to Figure 5, if the RDF triple pattern consists of three groups (S, P, ?ans), the first, the second, and the third RDF triples (S, P, ?ans) are labeled as A, E, and H, respectively, and the other cases will be assigned accordingly, as shown in Figure 6. According to these processing rules, if the RDF triple pattern of the SPARQL is (?ans, P, ?x)(?x, P, O)(S, P, ?ans), the resulting label set is {AfJ}. For SPARQL with two sets of RDF triples, there are 21 possible combinations (C_2^6+6), including 15 non-repeated label combinations and 6 duplicated labels. Figure 7 shows the 7 remaining RDF triple patterns after elimination.

For SPARQL with three sets of RDF triples, we combine the 6 RDF triples in Figure 5 and end up with a total of 56 possible patterns (C_3^6+36), 20 of which are from the combination of 6 types and 3 types in the case of non-repeatable labels, and 36 are from repeatable labels. After eliminating 22 remaining RDF triple patterns, RDF triple patterns also are excluded for labels not found in the dataset (QALD-7 [20], QALD-8 [21], QALD-9 [22], LC-QuAD [23]). The last label (Y) and its corresponding RDF triple pattern are shown in Figure 8.

Lable	RDF triple pattern	Example Quesiton
AE	(S , P , ?ans) (S , P , ?ans)	Do Urdu and Persian have a common root?
AF	(S , P , ?ans) (?ans , P , O)	In which U.S. state is Fort Knox located?
BF	(S , P , ?x) (?ans , P , ?x)	List all the musicals with music by Elton John.
aG	(S , P , ?x) (?x , P , ?ans)	How many companies were founded by the founder of Facebook?
ag	(?ans , P , O) (?ans , P , O)	How many seats does the home stadium of FC Porto have?
bG	(?x , P , O) (?ans , P , ?x)	Which games publishers are located in California?
bg	(?x , P , O) (?x , P , ?ans)	Give me all actors starring in movies directed by William Shatner.

FIGURE 7. Example of RDF triple pattern with two sets of triples.

Label	RDF triple pattern	Label	RDF triple pattern	Label set	RDF triple pattern
A	(S , P , ?ans)	ag	(S , P , ?x) (?x , P , ?ans)	BfJ	(?ans , P , O) (?x , P , O) (?ans , P , ?x)
B	(?ans , P , O)	bG	(?x , P , O) (?ans , P , ?x)	Bfj	(?ans , P , O) (?x , P , O) (?x , P , ?ans)
D	(S , P , O)	bg	(?x , P , O) (?x , P , ?ans)	aFJ	(S , P , ?x) (?ans , P , O) (?ans , P , ?x)
AE	(S , P , ?ans) (S , P , ?ans)	AEI	(S , P , ?ans) (S , P , ?ans) (?ans , P , O)	afj	(S , P , ?x) (?x , P , O) (?x , P , ?ans)
AF	(S , P , ?ans) (?ans , P , O)	AFI	(S , P , ?ans) (?ans , P , O) (?ans , P , O)	bfJ	(?x , P , O) (?x , P , O) (?ans , P , ?x)
BF	(?ans , P , O) (?ans , P , O)	AfJ	(S , P , ?ans) (?x , P , O) (?x , P , ?ans)	bfj	(?x , P , O) (?x , P , O) (?x , P , ?ans)
aG	(S , P , ?x) (?ans , P , ?x)	BFI	(?ans , P , O) (?ans , P , O) (?ans , P , O)		

FIGURE 8. The filtered labels after excluding the labels not found in the datasets.

B. SPARQL GENERATION

This section outlines how to transform the labels predicted by the Multi-label Model into SPARQL syntax. Given the four RDF triple structures (S, P, ?O), (?S, P, O), (?S, P, ?O), and (S, P, O) listed in Figure 5, four associated processing rules have been established by Pseudo-code. The key phrases in the question are denoted as PH. The SPARQL Generation step uses three key phrases, i.e., Named Entity Phrases, Relation Phrases, and Class Phrases. Named entity phrases are denoted by PH^E, where each entity phrase is represented as ph_i^E; in other words, PH^E is equivalent to a set of individually named entity phrases. That is, $PH^E = \{ph_i^E,\ i = 1 \ldots |PH^E|\}$. The Relation and Class phrases are represented by PH^R and PH^C, respectively. Here, $PH^R = \{ph_i^R, i = 1 \ldots |PH^R|\}$ and $PH^C = \{ph_i^C,\ i = 1 \ldots |PH^C|\}$. An example of the standard SPARQL for the question "Which company founded by Fusajiro Yamauchi also provides services for the Nintendo eShop?" is as follows:

```
SELECT ?ans WHERE {
?ans dbo:foundedBy dbr:Fusajiro_Yamauchi.
?ans dbo:services dbr: Nintendo_eShop.
?ans rdf:type dbo:Company.}
```

The entity phrases in this question are Fusajiro Yamauchi and Nintendo eShop, denoted by ph_1^E and ph_2^E, respectively. Therefore, the set of Named entity phrases is $PH^E = \{ph_1^E, ph_2^E\}$ with its size equal to 2 (i.e., $|PH^E| = 2$). The relationship phrases of the same concept include "founded by" and "services", represented by ph_1^R and ph_2^R, respectively, where $PH^R = \{ph_1^R, ph_2^R\}$ with a size of 2 (i.e., $|PH^R| = 2$). As for the class phrase, it is only "company," so ph_1^C is "company"; i.e., $PH^C = \{ph_1^C\}$ with a size of 1 (i.e., $|PH^C| = 1$).

The symbols R and C denote the sets of Uniform Resource Identifiers (URIs) for relational and class phrases, respectively. Figure 9 shows the Pseudo-code for generating these sets, with the code for generating R on the left and the code for generating C on the right. In the Pseudo-code, generating R is the union result of all R^i, which R^i is a set that contains the relation phrase ph_i^R through the ith loop. Similarly, the code for generating C involves processing each class phrase ph_i^C through a loop, adding the result of Entity Mapping (ph_i^C) to the set C^i, and then a union of all C^i to get C.

Pseudo-code for Define $\mathbb{R}$ and $\mathbb{C}$

Input :	**Input :**
$PH^R = \{ph_i^R, i = 1 \cdots |PH^R|\}$	$PH^C = \{ph_i^C, i = 1 \cdots |PH^C|\}$
Proccess :	**Proccess :**
1 **for** $i = 1$ to $|PH^R|$ **do**	1 **for** $i = 1$ to $|PH^C|$ **do**
2 $\mathbb{R}^i$ = Entity_Mapping(ph_i^R)	2 $\mathbb{C}^i$ = Entity_Mapping(ph_i^C)
3 $\mathbb{R} = \bigcup_{i=1,\ldots,\|PH^R\|} \mathbb{R}^i$	3 $\mathbb{C} = \bigcup_{i=1,\ldots,\|PH^C\|} \mathbb{C}^i$

FIGURE 9. Relationship resource collection and category resource collection.

An example of the question "Which company founded by Fusajiro Yamauchi also provides services of Nintendo eShop?" has two relational phrases. The first step is to search for the URIs of the first relational phrase, "founded by", resulting in R^1 = {dbo:foundedBy, dbo:founded, dbp:founder}. Next, the URIs of the second relational phrase, "services", are found to be R^2 ={dbo:service, dbo:services, dbp:militaryService}. Finally, the union of R^1 and R^2 results in R ={dbo:foundedBy, dbo:founded, dbp:founder, dbo:service, dbo:services, dbp:militaryService}. This question also has only one class phrase, i.e., "Company", for which the URIs are found to be C^1 ={dbo:Company, dbo:BusCompany} as there is only one class phrase, $C = C^1$.

The trained model's predicted label comprises multiple sub-labels, where $label = \{L_m,\ m = 1,\ \ldots,\ n\}$. A relevant

function is executed for each sub-label to produce a potential RDF triple; in the end, all RDF triples are joined to generate the SPARQL(s). If the sub-tags are A, E, or H, meaning the RDF triple is (S, P, ?ans); or if the sub-tags are a, e, or h, meaning the RDF triple is (S, P, ?x), with $Function_1$ described in Figure 10. In $Function_1$, the corresponding URIs of the named entity phrase are found through the *Entity_Mapping* method and added to E^i set, i.e., $E^i = Entity\_Mapping(ph_i^E)$. If the sub-label is A, E, or H, the generated RDF triple is $(e_j^i, r_k, ?ans)$; if the sub-label is a, e, or h, the generated RDF triple is $(e_j^i, r_k, ?x)$. Here, e_j^i represents the jth URI corresponding to the i-th named entity phrase (i.e., $e_j^i \in E^i$), and r_k represents the k-th URI in R (i.e., $r_k \in R$), thus composing $j \times k$ RDF triples.

In lines 5 and 7 of the Pseudo-code, the RDF triple is queried by DBpedia to check whether any results exist. If the result of ?ans or ?x is not empty, it signifies that the triple is present in DBpedia and is added to the set S. If the set S is empty, it implies that the currently named entity phrase doesn't match any r_k, and skip the generated RDF triples to process the next named entity phrase. If S is not empty, the RDF triples have been found for the named entity phrase, and then this procedure is skipped and removed from the set (PH^E).

For example, the label of the question, "For which games are Sam Loyd and Eric Schiller both famous?", is AE; thus, the sub-tags A and E call $Function_1$ once individually. Figure 11 shows the result of this question after *Entity_Mapping*, where the named entities are Sam Loyd and Eric Schiller. That is, R has three URIs, which are r_1, r_2 and r_3 as dbp:famousPact, dbp:famousFor, and dbo:KnownFor, respectively.

Sublabel A calls $Function_1$ and maps the first named entity phrase to Entity Mapped to get E^1 = {dbr:Sam_Loyd}. The combination of RDF triple $(e_j^i, r_k, ?ans)$ results in 3 sets of triples: < dbr:Sam_Loyd dbp:famousPact ?ans>, < dbr:Sam_Loyd dbp:famousFor ?ans>, and < dbr:Sam_Loyd dbo:KnownFor ?ans>. Only one set < dbr:Sam_Loyd dbo:KnownFor ?ans> exists in DBpedia, which is added to set S. If S is not empty, Sam Loyd is removed from the named entity phrase set (PH^E), and S is returned.

Next, sub-tag E also calls $Function_1$. Since Sam Loyd has been removed from the set of named entity phrases, only Eric Schiller remains in the set of named entity phrases. The same Entity Mapping is performed on the first named entity phrase to obtain E_1 ={dbr:Eric_Schille}, which only one URI is matched, so $j = 1$ (i.e., e_1^1 is dbr: Eric_Schille). We can get 3 groups of RDF triples, respectively, which are < dbr:Eric_Schiller dbp:famousPact ?ans>, < dbr:Eric_Schiller dbp:famousFor ?ans>, and < dbr:Eric_Schiller dbo:KnownFor ?ans>

This example also has only one RDF triple <dbr:Eric_Schiller dbo:KnownFor ?ans> in DBpedia, so we add this RDF triple to S. Because S is not an empty set, we remove Eric Schiller from the PH^E set and return S. Next, we remove

Algorithm : Pseudo-code for $Function_1$

Input :
$PH^E = \{ph_i^E, i = 1 \cdots |PH^E|\}$
$\mathbb{R} = \{r_k, k = 1 \cdots |\mathbb{R}|\}$
Outputs :
S : Possible RDF Triples
Proccess :
1 S = ∅
2 **for** i = 1 to $|PH^E|$ **do**
3 $\mathbb{E}^i = \text{Entity\_Mapping}(ph_i^E)$
4 **if** L_m is upper case **do**
5 $S = \left\{(e_j^i, r_k, ?ans) \mid \forall_{e_j^i} \forall_{r_k} (e_j^i, r_k, ?ans) \in \text{DBpedia}, e_j^i \in \mathbb{E}^i, j = 1, \cdots, |\mathbb{E}^i| ; r_k \in \mathbb{R}, k = 1, \cdots, |\mathbb{R}|\right\}$
6 **else do**
7 $S = \left\{(e_j^i, r_k, ?x) \mid \forall_{e_j^i} \forall_{r_k} (e_j^i, r_k, ?x) \in \text{DBpedia}, e_j^i \in \mathbb{E}^i, j = 1, \cdots, |\mathbb{E}^i| ; r_k \in \mathbb{R}, k = 1, \cdots, |\mathbb{R}|\right\}$
8 **if** S ≠ ∅ **do**
9 PH^E.remove(PH_i^E)
10 **break**

FIGURE 10. ***Function*$_1$'s Pseudo-code.**

the RDF triple with tags A and E from S. Finally, we need to combine the RDF triples of tag A and E together and add the SELECT clause to generate SPARQL as follows:

SELECT ?ans WHERE {
dbr:Sam_Loyd dbo:knownFor ?ans.
dbr:Eric_Schiller dbo:knownFor ?ans. }

Figure 12 shows the Pseudo-code of $Function_2$, which is executed if the sublabel is B, F, or I (i.e., the RDF triple is (?ans, P, O)) or b, f, or i (i.e., (?x, P, O)). If P^C is not empty, the P and O positions are replaced with rdf:type and c_k, respectively, where c_k represents the kth URIs in C (i.e., $c_k \in C$). For example, if the set of sublabels is B, F, or I, the generated RDF triple is (?ans, rdf:type, c_k). After generating all possible RDF triples, we empty the P^C.

After processing the P^C, we sequentially process the set of named entity phrases P^E. Similarly to $Function_1$, we perform $Entity\_Mapping(p_i^E)$ for each i to obtain the URIs corresponding to p_i^E, i.e., E^i. If the sublabel is B, F, or I, then RDF triple(?ans, r_k, e_j^i) is generated; if the sublabel is a, e, or h, then RDF triple(? x, r_k, e_j^i); where e_j^i represents the jth URIs corresponding to the ith named entity phrase (i.e., $e_j^i \in E^i$), and r_k represents the k-th URIs in R (i.e., $r_k \in R$). A $j \times k$ set of RDF triple is assembled as shown in line 12. Finally, check whether S is an empty set: if it is, then proceed to try the next named entity phrase; if it is not, it means that this named entity phrase has matched the knowledge graph in this RDF triple structure successfully, so we remove the named entity phrase from the set (PH_i^E) and exit the program.

Take the example of the question, "Which company founded by Fusajiro Yamauchi also provides services of

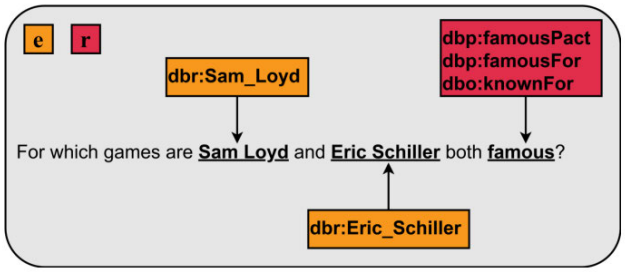

FIGURE 11. The results of Entity Mapping for question "What games are Sam Loyd and Eric Schiller both known for?".

Algorithm : Pseudo-code for $Function_2$

Input :

Label = $\{L_m, m = 1,2,3\}$

$P^E = \{p_i^E, i = 1 \cdots |P^E|\}$: Named entity phrases

$P^C = \{p_i^c, i = 1 \cdots |P^C|\}$: Class phrases

$\mathbb{R}$: All relation phrase's URIs

$\mathbb{C}$: All class phrase's URIs

Outputs :

S : Possible Triple

Proccess :

```
S = ∅
if P^C ≠ ∅ do
    if L_m is upper case do
        S = {(? ans, rdf: type, c_k), c_k ∈ ℂ, k = 1, ⋯ , |ℂ|}
    else do
        S = {(? x, rdf: type, c_k), c_k ∈ ℂ, k = 1, ⋯ , |ℂ|}
    P^C = ∅
else do
    for i = 1 to |P^E| do
        E^i = Entity_Mapping(p_i^E)
        if L_m is upper case do
            S = {(? ans, r_k, e_j^i) | ∀_{e_j^i} ∀_{r_k} (? ans, r_k, e_j^i) ∈ DBpedia, e_j^i ∈
                E^i , j = 1, ⋯ , |E^i | ; r_k ∈ ℝ, k = 1, ⋯ , |ℝ|}
        else do
            S = {(? x, r_k, e_j^i) | ∀_{e_j^i} ∀_{r_k} (? x, r_k, e_j^i) ∈ DBpedia, e_j^i ∈
                E^i , j = 1, ⋯ , |E^i | ; r_k ∈ ℝ, k = 1, ⋯ , |ℝ|}
        if S ≠ ∅ do
            P^E.remove(P_i^E)
            break
```

FIGURE 12. *Function*$_2$'s Pseudo-code.

Nintendo eShop?'', the label of this question is BFI. The set of named entity phrases of this question has Fusajiro Yamauchi and Nintendo eShop, and R has six URIs: r_1 is dbo:foundedBy, r_2 is dbo:found, r_3 is dbp: founder, r_4 is dbp:service, r_5 is dbp:services, and r_6 is dbp:militaryService; C has two URIs, i.e., c_1 is dbo:Company and c_2 is dbo:BusCompany.

First, after sublabel B calls $Function_2$, since P^C is not an empty set, it generates RDF triple (?ans, rdf:type, c_k). S gets <?ans rdf:type dbo:Company>, <?ans rdf:type dbo:BusCompany>, empties the P^C, and returns S.

Next, sub-label F similarly calls $Function_2$ because P^C is the empty set, so the 1st named entity phrase is Entity Mapped, which yields E^1 = < dbr:Fusajiro_Yamauchi>, because sub-label F is capitalized, so it yields (?ans, r_k, e_j^i); since Fusajiro Yamauchi maps to only 1 URI, so $j = 1$, i.e., e_1^1 is dbr:Fusajiro_Yamauchi, so we will generate 6 sets of RDF triples:

< ?ans dbo:foundedBy dbr:Fusajiro_Yamauchi>,
< ?ans dbo:founded dbr:Fusajiro_Yamauchi>,
< ?ans dbp:founder dbr:Fusajiro_Yamauchi>,
< ?ans dbp:service dbr:Fusajiro_Yamauchi>,
< ?ans dbp:services dbr:Fusajiro_Yamauchi>, and
< ?ans dbp:militaryService dbr:Fusajiro_Yamauchi>.

In this example, two sets of RDF triples that existed in DBpedia are added to S. Since S is not an empty set, we remove Fusajiro Yamauchi from the named entity phrase set (PH_i^E) and return S.

Next, while sub-label I calls $Function_2$, only Nintendo eShop remains in the set of named entity phrases. The first named entity phrase is to obtain E^1 = {dbr: dbr:Nintendo_eShop } and to generate (?ans, r_k, e_j^i). Nintendo eShop is only mapped to one URI (i.e., e_1^1 is dbr:Nintendo_eShop). Next, six sets of RDF triples are generated: <?ans dbofoundedBy dbr:Nintendo_eShop>, <?ans dbo:founded dbr:Nintendo_eShop>, <?ans dbp:founder dbr:Nintendo_eShop>, <?ans dbp:service dbr:Nintendo_eShop>, <?ans dbp:services dbr:Nintendo_eShop>, and <?ans dbp:militaryService dbr:Nintendo_eShop>. In this example, only RDF triples <?ans dbp:services dbr:Nintendo_eShop> exist in DBpedia, so these two are added to S. Since S is not an empty set, we remove Nintendo eShop from the named entity phrase set (PH^E) and return it to S. Finally, the SPARQL is generated by combining the four sets (i.e., $2 \times 2 \times 1$) of RDF triples generated by labels B, F, and I and adding the SELECT clause:

SELECT ?ans WHERE
?ans rdf:type dbo:Company.
?ans dbo:foundedBy dbr:Fusajiro_Yamauchi.
?ans dbp:services dbr:Nintendo_eShop.

SELECT ?ans WHERE
?ans rdf:type dbo:Company.
?ans dbo:founder dbr:Fusajiro_Yamauchi.
?ans dbp:services dbr:Nintendo_eShop.

SELECT ?ans WHERE
?ans rdf:type dbo:BusCompany.
?ans dbo:foundedBy dbr:Fusajiro_Yamauchi.
?ans dbp:services dbr:Nintendo_eShop.

SELECT ?ans WHERE
?ans rdf:type dbo:BusCompany.
ans dbo:founder dbr:Fusajiro_Yamauchi.
?ans dbp:services dbr:Nintendo_eShop.

Figure 13 shows the Pseudo-code of $Function_3$, which is executed if the sublabel is G or J (i.e., the RDF triple is (?ans, P, ?x)), the generated RDF triple is (?ans, r_k, ?x). If the sublabel is g or j (i.e., the RDF triple is (?x, P, ?ans)), RDF triple is generated as (?x, r_k, ?ans); where r_k represents the kth URIs in R.

Algorithm : Pseudo-code for $Function_3$

Input :
r : Lookup_Relation(P^R: relation phrases)
Outputs :
S : Possible Triple
Proccess :
1 $S = \emptyset$
2 **if** L_m is upper case **do**
3 $S = \{(?ans, r_k, ?x), r_k \in \mathbb{R}, k = 1, \cdots, |\mathbb{R}|\}$
4 **else do**
5 $S = \{(?x, r_k, ?ans), r_k \in \mathbb{R}, k = 1, \cdots, |\mathbb{R}|\}$

FIGURE 13. ***Function*$_3$'s Pseudo-code.**

Take the question "What is the alma mater of the scientist who is known for Rational analysis?" as an example, the label of this question is *bff*. b and f call $Function_2$, and j call $Function_3$ once. After Entity Mapping, the set of named entity phrases is "Rational analysis". R has six URIs, which are dbp:almaMeters, dbo:almaMater, dbp:almaMater, dbp:asKnownAs, dbo:knownFor and dbp:knownAs, denoted by r_1, r_2, r_3, r_4, r_5, and r_6, respectively. C has only one URI (i.e., c_1 is dbo:Scientist). First, after sublabel b calls $Function_2$, P^C is not an empty set to generate an RDF triple (?x, rdf:type, c_k). After that, S gets <?x rdf:type dbo:Scientist>, and clears P^C and returns S. Next, the sub-tag, f, also calls $Function_2$, and since P^C is empty, the first named entity phrase is mapped to Entity Mapping to get E^1 ={dbr: Rational_analysis}. Since the sub-tag f is lowercase, it produces (?x, r_k, e^i_j); since Rational analysis only maps to one URI, i.e., $j = 1$ and e^1_1 is dbr:Rational_analysis. six sets of RDF triples are generated, i.e., <?x dbp:almaMeters dbr:Rational_analysis>,<?x dbo:almaMater dbr:Rational_analysis>, <?x dbp:almaMater dbr:Rational_analysis>, <?x dbp:asKnownAs dbr:Rational_analysis>, <?x dbo:knownFor dbr:Rational_analysis>, and <?x dbp: dbp:knownAs dbr:Rational_analysis>.

In this example, only one set of RDF triples exists in DBpedia (i.e., <?x dbo:knownFor dbr:Rational_analysis>). S is not an empty set, so we remove Rational analysis from PH^E_i and return S. Next, while the sublabel j calls $Function_3$, the sublabel j is lowercase to generate the RDF triple (?x, r_k, ?ans). Since R has six URIs, we generate six sets of RDF triples to S as: <?x dbp:almaMeters ?ans>, <?x dbo:almaMater ?ans>, <?x dbp:almaMater ?ans>, <?x dbp:asKnownAs ?ans>, <?x dbo:knownFor ans>, and <?x dbp: dbp:knownAs ?ans>.

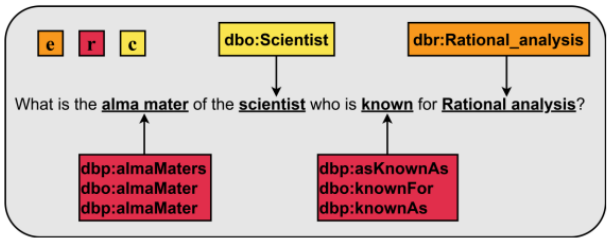

FIGURE 14. Figure 3-10 Question "What is the alma mater of the scientist who is known for Rational analysis?" After the results of Entity Mapping.

Finally, by combining the RDF triple with tags b, f, and j and adding the SELECT clause, the following six SPARQL syntaxes are generated:

SELECT ?ans WHERE {
?ans rdf:type dbo:Scientist.
?x dbo:knownFor dbr:Rational_analysis.
?x dbp:almaMeters ?ans.}

SELECT ?ans WHERE {
?ans rdf:type dbo:Scientist.
?x dbo:knownFor dbr:Rational_analysis.
?x dbo:almaMater ?ans.}

SELECT ?ans WHERE {
?ans rdf:type dbo:Scientist.
?x dbo:knownFor dbr:Rational_analysis.
?x dbp:almaMater ?ans.}

SELECT ?ans WHERE {
?ans rdf:type dbo:Scientist.
?x dbo:knownFor dbr:Rational_analysis.
?x dbp:asKnownAs ?ans.}

SELECT ?ans WHERE {
?ans rdf:type dbo:Scientist.
?x dbo:knownFor dbr:Rational_analysis.
?x dbo:knownFor ?ans.}

SELECT ?ans WHERE {
?ans rdf:type dbo:Scientist.
?x dbo:knownFor dbr:Rational_analysis.
?x dbp:knownAs ?ans.}

According to the rules of SPARQL, the label D, i.e., (S, P, O), usually appears in SPARQL alone to call $Function_4$ as shown in Figure 15. If the question is marked as D, two named entity phrases, denoted by E_1 and E_2, should be placed in S or O, so the two named entities must be swapped once, i.e., (E_1, P, E_2) or (E_2, P, E_1).

According to pseudo codes, all name entities p^E_i are to find the corresponding URIs through the Entity Mapping process. After $|P^E|$ times Entity Mapping, we can get $\{E^1, E^2, \ldots, E^{|P^E|}\}$, where $|P^E|$ is the number of

Algorithm : Pseudo-code for Function$_4$

Input :

P^E : Named entity phrases

r : Lookup_Relation(P^R: relation phrases)

Outputs :

S : Possible Triple

Proccess :

```
1   S = ∅
2   for i = 1 to |P^E| do
3       E^i = Entity_Mapping(p_i^E)
4   for r= 1 to (|P^E| − 1) do
5       for s = r + 1 to |P^E| do
6           S = S ∪ {(e_o^r, r_k, e_q^s), (e_q^s, r_k, e_o^r)} | e_o^r ∈ E^r, o = 1, ···, |E^r| ;
            e_q^s ∈ E^s, q = 1, ···, |E^s| ; r_k ∈ R, k = 1, ···, |R|
```

FIGURE 15. Function$_4$'s pseudo code.

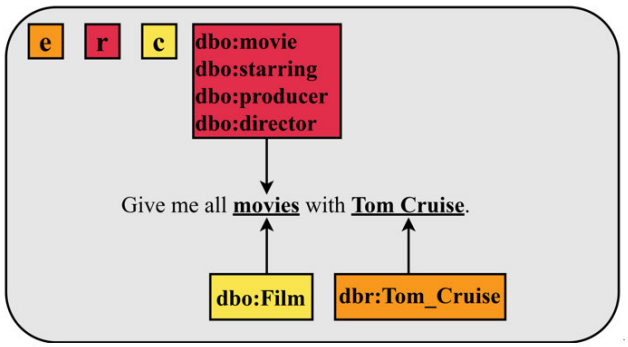

FIGURE 16. An Entity Mapping result example.

name entities. Since two name entities can be extracted to the URIs, $C_2^{|P^E|}$ loop iterations to generate RDF triples $\{(e_o^r, r_k, e_q^s), (e_q^s, r_k, e_o^r)\}$, where e_j^r indicates rth name entity to jth URIs (i.e., $e_j^r \in E^r$), r_k represents the kth URIs in R, and e_q^s shows the sth name entity to the qth URIs (i.e., $e_q^s \in E^s$).

Take the sentence "Is camel of the chordate phylum?" for example, the question is labeled as D to call the function *Function*$_4$. The values E^1 = {dbr:Camerl} and E^2 = {dbr:chordate} can be obtained according to Entity Mapping for C_2^2 loop iterations, which E^r and E^s are E^1 and E^2, respectively. Therefore, two RDF triple templates are fit as (dbr:Camel, r_k, dbr:chordate) and (dbr:chordate, r_k, dbr:Camel). R has three URIs to generate six RDF triples. Note that only one RDF triple is reserved based on labeled D. After adding the ASK clause, six SPARQL queries are generated as listed below.

ASK where {dbr: Camel dbp:infraphylum dbr:chordate}
ASK where { dbr:chordate dbp:infraphylum dbr: Camel}
ASK where {dbr: Camel dbp:unrankedPhylum dbr:chordate}
ASK where { dbr:chordate dbp:unrankedPhylum dbr: Camel}
ASK where {dbr: Camel dbp:phylum dbr:chordate}
ASK where { dbr:chordate dbp:phylum dbr: Camel}

IV. QUERY PROCESSING

As shown in Figure 3, the query phase has six major steps. This section introduces the six steps of the system, using the question "Give me all movies with Tom Cruise." as an example:

SELECT ?ans WHERE {
?ans rdf:type dbo:Film.
?ans dbo:starring dbr:Tom_Cruise.}

A. THE PRE-PROCESSING, ENTITY TYPE TAGGER, AND ENTITY MAPPING STEPS

The pre-processing process is the question conversion into the pre-trained numeric format, such as tokenization, lemmatization, Part-of-Speech Tagging, and others. After the pre-processing step, the example is translated to lexical tokens, i.e., "VB PRP DT NNS IN NN."

The tokens are used in the subsequent step, i.e., Entity Type Tagger, to find the keywords in the question, which is referred to the five entity type tags "V, N, E, R, C" designed by Xser [11], where V, N, E, R, and C stand for question word, an unimportant word or stop word, a named entity, an attribute entity, and a class entity, respectively. If the entity is composed of more than one word, "-B" and "-I" are added after the entity type to represent it as the same entity. For example, the named entity Tom Cruise would be marked as "Tom/E-B Cruise/E-I." In addition to tags V, N, E, R and C, we apply Chen's work [24] to expend "ER, CR, VR, CE, and RR" into entity type tagger. In this example, "movie" appears only once in the question. Still, two URIs correspond to "movie," such as the class entity "dbo:Film" in <?ans rdf:type dbo:Film> and the attribute entity "dbo:starring" in <?ans dbo:starring dbr:Tom_Cruise>. Thus, this example shows that an entity may need a compound tag, so "movie" is tagged as "CR." Thus, the tagging result is: "Give/V-B me/N all/N movies/CR-B with/N Tom/E-B Cruise/E-I."

The third step, Entity Mapping, is used to facilitate the keywords. When the entities are marked as E, R, and C, the Named Entity Mapping [25], the Attribute Entity Mapping [26], and the Class Entity Mapping [27] are performed, respectively. If the tag is "ER, CR, or CE," the entities are compared to the two corresponding tags, and if the tag is VR or RR, one attribute entity is performed. In the example, the entity "movie" performs Class Entity Mapping and Attribute Entity Mapping as shown in Figure 16.

B. MULTI-LABEL MODEL

Three trained multi-tag models, i.e., BR, CC, and Ensemble BR, are proposed to predict the tags associated with the question sentence X^*. In the BR model, the label *Label* associated with the question sentence is obtained by querying q separate BR models, denoted by $BR_j(.)$, with the following formula. Here, if $BR_j(X^*)$ is larger than or equal to 0.5, the sublabel y_j

is added as one of the labels for the question.

$$Label = \{y_j | BR_j(X^*) \geq 0.5, 1 \leq j \leq q\}$$

The difference between the CC model and the BR model in the query phase is that the CC model considers the correlation between the tags. Hence, the output of the previous model must be added to the next CC model, denoted by $CC_j([X^*, (CC_1(X^*), \ldots, CC_{j-1}(X^*))])$, with the following equation:

$$Label = \{y_j | CC_j([X^*, (CC_1(X^*), \ldots, CC_{j-1}(X^*))]) \geq 0.5, 1 \leq j \leq q$$

Based on the BR approach, the Ensemble BR model better considers the correlation between the labels, where a separate model is trained. The output of q models is used as the input to the new model with the following equation. Therefore, the relevant label for the example question is "BF."

$$Label = Ensemble\_BR([BR_1(X^*), BR_2(X^*), \ldots, BR_q(X^*))])$$

C. ANSWER EVALUATION

Answer retrieval consists of SPARQL Generation and Evaluation, which generate possible SPARQL queries and evaluate the SPARQL queries for the answer.

During the SPARQL Generation phase, the label of the example is "BF." Thus, "B" and "F" totally call $Function_2$ twice. After Entity Mapping, the set of named entity phrases is only Tom Cruise; thus, R has four URIs, denoted by r_1, r_2, r_3, and r_4, and they are dbo:movie, dbo:starring, dbo:producer, and dbo:director, respectively, and C has only one URI as c_1 is dbo:Film. Since P^C is not an empty set and the sublabel is capital "B," it generates RDF triple (?ans, rdf:type, c_k). After the process, S is <?ans rdf:type dbo:Film>, and resets P^C as empty.

Next, sublabel F similarly calls $Function_2$, and P^C is the empty set to obtain $E_1 = \{$dbr:Tom_Cruise$\}$ during Entity Mapping. Because the sub-tag F is uppercase, it generates (?ans, r_k, e^i_j). The named entity, Tom Cruise, only maps to an URI (i.e., $j = 1$) to get e^1_1 as dbr:Tom_Cruise. Four sets of RDF triples are listed as: <?ans dbo:movie dbr:Tom_Cruise>, <?ans dbo:starring dbr:Tom_Cruise>, <?ans dbo:producer dbr:Tom_Cruise>, and <?ans dbo:director dbr:Tom_Cruise>. In this example, two sets of RDF triples <?ans dbo:starring dbr:Tom_Cruise> and <?ans dbo:producer dbr:Tom_Cruise> existed in DBpedia are added to S. Since S is not an empty set, we remove Tom Cruise from the named entity phrase collection (PH^E_i) and pass back S.

Finally, combining the RDF triples generated by the tags "B" and "F" and adding the SELECT clause, two SPARQL queries are generated as denoted by $SPARQL_1$ and $SPARQL_2$ as follows:

```
SELECT ?ans WHERE {
?ans rdf:type dbo:Film.
?ans dbo:starring dbr:Tom_Cruise.}
```

```
SELECT ?ans WHERE {
?ans rdf:type dbo:Film.
?ans dbo:producer dbr:Tom_Cruise.}
```

The Evaluation phase evaluates whether the ith SPARQL, denoted by $SPARQL_i$, can request DBPedia to return the answers. The answers are concatenated as the final answer, represented by $\cup_{i=1,\ldots,n} SPARQL_i$. The example can generate two sets of SPARQLs, i.e., $SPARQL_1$ and $SPARQL_2$, mentioned in the previous subsection. The answers obtained from $SPARQL_1$ are based on the 44 answers available as of September 2022 in the DBpedia database. The total number of answers obtained by $SPARQL_2$ was 16. Finally, the answers from $SPARQL_1$ and $SPARQL_2$ were combined in response to the request "Give me all movies with Tom Cruise" in this step.

D. ANSWER FILTERING

The entities tagged with "V" according to the Entity Type Tagger are processed accordingly. The following order introduces six answer processing methods, i.e., person or organization, place, time, right or wrong question, value, and others. For this example, Entity Type Tagger marks "Give" as "V," so we can directly send the result back to the user.

- Who: The answer type is either Person or Organization. The category type (rdf:type) corresponding to Person in DBpedia is dbo:Person, while Organization is dbo:Organisation, so only answers matching dbo:Person or dbo:Organisation in $SPARQL_i$ are returned.
- Where: The answer corresponding to Place in DBpedia is dbo:Place, so only the answers that match dbo:Place in $SPARQL_i$ are returned.
- When: It is xsd:dateTime, so the $SPARQL_i$ containing the string of xsd:date is returned.
- Auxiliary verbs (e.g., Does, Did) or be verbs: The answer type is right and wrong. The result of the query is the basis for judgment. True is returned if there is an answer; otherwise, False is returned.
- How (How many): The answer type is numeric. The first is that the actual value of the answer has been recorded in the DBpedia. For example, for the question "How many pages do War and Peace have?", we can get the number of pages in the book directly through the attribute dbo:numberOfPage of War_and_Peace.
- Which, What, List, Give, and Show: These types of questions usually are used in querying nouns or sets. We use the result of the query directly as the final answer.

V. EXPERIMENTS

The system was developed using QALD-7 [20], QALD-8 [21], QALD-9 [22], and LC-QuAD [23] as the training and testing data for this experiment. LC-QuAD is a dataset that contains 5000 questions, and the SPARQL is required to answer the questions on DBpedia. Note that we only use the SPARQL questions that exclude the comparative or superlative questions, such as "FILTER" or "ORDER BY" in SPARQL

TABLE 2. Model superparameters.

Parameter	Value
Word Embedding	GloVeBERT
POS Embedding	20
Network	LSTMBi-LSTM
LSTM Units	64128256
LSTM Layers	123
Epochs	5255075100
Learning Rate	1 x 10^{-3}
Loss Function	binary crossentropy
Optimizer	Adam Optimizer

TABLE 3. Comparing the accuracy of three multi-label learning methods.

Dataset	Embedding	BR	CC	LP [15]
QALD-7	GloVe	**82.6%**	78.26%	78.26%
	BERT	78.26%	73.91%	73.91%
QALD-8	GloVe	**93.94%**	**93.94%**	90.91%
	BERT	78.79%	84.85%	84.84%
QALD-9	GloVe	71.95%	72.29%	**73.17%**
	BERT	**73.17%**	63.41%	70.73%
LC-QuAD	GloVe	59.3%	60.2%	**75.5%**
	BERT	64.1%	58.8%	75.3%

TABLE 4. Accuracy of BR and CC on each triple (LC-QuAD).

Number of Triples	BR	CC
1	83.51%	81.36%
2	56.68%	52.61%
3	56.42%	51.07%
Total	64.1%	60.2%

queries. The original LC-QuAD test set had 1000 questions that were unavailable in SPARQL, so they were removed and the remaining 604 questions were used for the End-to-End performance analysis.

A. EVALUATION INDICATOR

The 0/1 subset accuracy [28] (as shown in Equation (1) is used to evaluate the classification results of BR and CC, where n represents the number of samples. The metric detects the percentage of samples for which the predicted label set (P) matches precisely with the TRUE label set (T). If the entire TRUE label set is identical to the Predicted label set (i.e., $P = T$), return 1; otherwise, 0. For example, suppose that the true label set of one of the samples is $\{y_1, y_3\}$, and the predicted label set is also $\{y_1, y_3\}$, return 1. Otherwise, regardless of whether the predicted tag set is predicted with one more tag, such as $\{y_1, y_2, y_3\}$, or one less tag, such as $\{y_1\}$, return 0.

$$Accuracy = \frac{1}{n}\Sigma_{i=1}^{n}[T_i = P_i] \tag{1}$$

Furthermore, it calculates performance metrics, such as Precision, Recall, and F-measure, as benchmarks to measure the performance of the question and answer system, as shown in Equations (2), (3), (4), where C represents the number of correct system answers to the question, N denotes the total number of system answers, and G indicates the gold standard answers that correspond to the correct answer for the question in the test set.

TABLE 5. Ensemble BR accuracy rate.

Dataset	Embedding	LP [15]	BR	Ensemble BR
QALD-7	GloVe	78.26%	82.6%	78.26%(-4.34%)
	BERT	73.91%	78.26%	**82.6%**(+4.34%)
QALD-8	GloVe	90.91%	93.94%	**93.94%(-)**
	BERT	84.84%	78.79%	84.85%(+6.06%)
QALD-9	GloVe	73.17%	71.95%	**76.82%**(+4.87%)
	BERT	70.73%	73.17%	74.39%(+1.22%)
LC-QuAD	GloVe	75.5%	59.3%	72.1%(+12.8%)
	BERT	75.3%	64.1%	**76.1%**(+12%)

TABLE 6. Accuracy of Ensemble BR for each triple on LC-QuAD dataset.

Number of Triple	LP	BR	Ensemble BR
1	232/279(83.15%)	233/279(83.51%)	247/279(88.53%)
2	319/441(72.34%)	250/441(56.68%)	309/441(70.07%)
3	204/280(72.86%)	250/441(56.42%)	205/280(73.21%)
Total	75.5%	64.1%	76.1%

TABLE 7. LC-QuAD End-to-End performance analysis.

System	Precision	Recall	F-measure
QAMP	0.25	0.5	0.33
DTQA	0.33	0.34	0.33
Presented	**0.274**	**0.413**	**0.287**

$$Recall = \frac{C}{G} \tag{2}$$

$$Precision = \frac{C}{N} \tag{3}$$

$$\text{F-measure} = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{4}$$

B. PARAMETERS

The Embedding used in this study includes GloVe, BERT, and Parts of Speech (POS); GloVe [29] is an application of an unsupervised learning algorithm to obtain vector representations of single words. And it provides four different dimensional versions, i.e., 50d, 100d, 200d, and 300d.

BERT [30] is a Word Embedding model that Google introduced in 2018. It can solve part of the multiword problem by applying a corpus of 800 million words to train a Contextual Encoder to give different numerical representations of words according to different contexts. The pre-trained BERT uses a pre-trained bert-base-uncased file with an Embedding Size of 768.

POS notation is helpful in NLP tasks because it provides information about the adjacency of words (e.g., a noun may be preceded by a qualifier or an adjective) and syntactic structure (e.g., a noun usually is part of a noun phrase). Therefore, the research team trained POS Embedding using SIKP-gram [31], where the Window Size is set to n, in which

TABLE 8. QALD-7, QALD-8, QALD-9 End-to-End performance analysis.

Dataset	Measure metric	gAnswer2	WDAqua	QAwizard	Light-QAwizard	DTQA	**Presented**
QALD-7	Precision	0.469	0.16	0.59	0.565	-	**0.598**
	Recall	0.498	0.162	0.59	0.652	-	**0.696**
	F-measure	0.487	0.163	0.59	0.594	-	**0.612**
QALD-8	Precision	0.3862	0.391	0.375	0.462	-	**0.522**
	Recall	0.3902	0.407	0.358	0.5	-	**0.561**
	F-measure	0.388	0.387	0.343	0.457	-	**0.517**
QALD-9	Precision	0.293	0.261	0.311	0.398	0.314	**0.424**
	Recall	0.327	0.267	0.469	0.426	0.322	**0.476**
	F-measure	0.298	0.25	0.33	0.406	0.301	**0.43**

TABLE 9. Accuracy of each triple after removing the SPARQL that cannot be queried.

Number of Triples	1	2	3	Total	Accuarcy
LC-QuAD	189/256	152/192	106/156	447/604	74.01%

the first n words and the last n words are used as the prediction data during the training of the model. The POS Embedding training set used in this study is Treebank [32], and we set the Window size to 5.

After introducing the model inputs, the next step is introducing the model hyperparameters, as shown in Table 2. Epochs are the number of times the model is trained, and the experimental hyperparameters are 5, 25, 50, 75, and 100. The Loss Function is the loss function of the model. Since both BR and CC are binary classifiers, we use binary cross entropy. Optimizer is used to help the neural network adjust the parameters. Adam Optimizer was used because it has a better performance than other optimizers [27].

C. PERFORMANCES IN DIFFERENT MULTI-LABELING MODELS

Table 3 summarizes the experimental results of LP [15], BR, and CC in QALD-7, QALD-8, QALD9, and LC-QuAD. In the QALD-7 test set, the best accuracy of 82.6%, 78.26%, and 78.26% were obtained for BR, CC, and LP, respectively, using GloVe as the pre-training feature. In the QALD-8 test set, using GloVe as the pre-training feature, higher accuracies were achieved for BR, CC, and LP, with the best accuracies of 93.94% for both BR and CC, while LP had an accuracy of 90.91%. In the QALD-9 test set, BR achieved the best accuracy, i.e., 73.17%, using BERT as the pretraining feature, CC achieved the best accuracy of 72.29% using GloVe as the pretraining feature, and LP achieved the best accuracy of 73.17% using GloVe as the pretraining feature. In the LC-QuAD test set, the best accuracy of 64.1% was obtained for BR with BERT as the pre-trained feature, 60.2% for CC with GloVe as the pre-trained feature, and 75.5% for LP with GloVe as the pre-trained feature. The experiments show that BR's method outperforms CC's method except that the accuracy of BR and CC are equal in the QALD-9 test set. Glove can achieve relatively good results for small datasets, such as QALD. The LP approach treats the RDF triple pattern as a label directly to describe the correlation between the labels. Therefore, as the proportion of complex questions (more than two RDF triples) increases, the performance of BR/CC decreases slowly. The performance of LP also shows up in QALD-9 and LC-QuAD datasets with more RDF triples.

The accuracies of BR and CC on the LC-QuAD test set for each RDF triple are shown in Table 4. The model, trained by the BR method, can obtain better accuracy than CC for generating SPARQL structure. Still, the biggest drawback of BR's method is that the labels are independent of each other. Therefore, although the BR method achieved the highest accuracy in the QALD-7, QALD-8, and QALD-9 test sets, the LC-QuAD test set up to 72.1% of SPARQLs being more than two RDF triples is not as good as LP. The Ensemble model is trained by training a separate model and using the output of the q model as the input of the new model to obtain the final result.

The accuracy rate of Ensemble BR is shown in Table 5, which shows that Ensemble BR achieves the best results in all four test sets. In particular, in the LC-QuAD test set, because the SPARQL of two or more RDF triples in the test set is as high as 72.1%, the accuracy of the BR method is much lower than that of the LP method. Still, after applying the Ensemble BR method, it obtains a better accuracy than the LP.

According to the results in Table 6, Ensemble BR improves the accuracy of LC-QuAD by more than 10%. Ensemble BR improves the SPARQL accuracy for only one set of RDF triples by 5.02%. For two sets of RDF triples and three RDF triples, the SPARQL accuracy improves by 16.74%.

D. END-TO-END PERFORMANCES

Table 7 presents the End-to-End performance of the proposed system on the LC-QuAD test set and compares it with two other QA systems, i.e., QAMP [33] and DTQA [34], respectively.

Table 9 summarizes the number of questions in the SPARQL syntax of 604 questions in one, two, and three RDF triples to show that the number of questions is correctly predicted using the Ensemble BR model.

Suppose the SPARQL of the question has only one set of RDF triples. In that case, the accuracy rate is 73.83%, i.e., the number of questions correctly predicted by the model is

189 divided by the number of questions with only one set of RDF triples, i.e., 256. If there are two sets, the accuracy rate is 79.17% (i.e., 152/192); and if there are three sets, the accuracy rate is 67.95% (i.e., 106/156); and finally, the accuracy rate of 604 questions is 74.01% (447/604).

The QALD metrics (Precision, Recall, F-measure) are used to measure the performances of QALD-7, QALD-8, and QALD-9 that are shown in Tables 8. The proposed system was compared to other systems, i.e., gAnswer2 [35], Wdaqua [11], QAwizard [14], Light-QAwizard [15], and DTQA [34]. gAnswer2 and Wdaqua have won the QALD competition in the past. The results of the evaluation metrics show that the proposed method has good performance.

VI. CONCLUSION

The paper integrates problem transformation and Ensemble Learning into the multi-label classification to achieve 82.6%, 93.94%, 76.82%, and 76.1% for QALD-7, QALD-8, QALD-9, and LC-QuAD, respectively, all of which are better than those of LP. A new slot-filling method based on Entity Type Tagger and Multi-label Model for generating SPARQL queries achieved relatively good results on the End-to-End metrics for several datasets.

A better entity matching mechanism is necessary to make the end-to-end query more efficient, especially for the part of attribute entities. An attribute instance must usually be compared to dozens or even hundreds of URIs. It requires a suitable ranking mechanism to make the query process efficient.

In addition, the Ensemble BR or CC model exhibits good performance by incorporating relationships between labels into the trained model. The scalability should be improved when a new label is added to require retraining the model to establish the relationships.

REFERENCES

[1] Z. Abbasiantaeb and S. Momtazi, "Text-based question answering from information retrieval and deep neural network perspectives: A survey," *WIREs Data Mining Knowl. Discovery*, vol. 11, no. 6, p. e1412, Nov. 2021.

[2] D. Jurafsky and J. H. Martin, *Speech and Language Processing*. Stanford, CA, USA: Stanford Univ. Press, 2019.

[3] S. Khalid, S. Wu, A. Wahid, A. Alam, and I. Ullah, "An effective scholarly search by combining inverted indices and structured search with citation networks analysis," *IEEE Access*, vol. 9, pp. 120210–120226, 2021.

[4] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer, "DBpedia—A large-scale, multilingual knowledge base extracted from Wikipedia," *Semantic Web*, vol. 6, no. 2, pp. 167–195, 2015.

[5] D. Vrandečić and M. Krötzsch, "Wikidata: A free collaborative knowledgebase," *Commun. ACM*, vol. 57, no. 10, pp. 78–85, Sep. 2014.

[6] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: A core of semantic knowledge," in *Proc. 16th Int. Conf. World Wide Web*, 2007, pp. 697–706.

[7] E. Prud'hommeaux and Andy Seaborne, *SPARQL Query Language for RDF*. W3C Recommendation, 2008.

[8] Y. Lan, G. He, J. Jiang, J. Jiang, W. Xin Zhao, and J.-R. Wen, "A survey on complex knowledge base question answering: Methods, challenges and solutions," 2021, *arXiv:2105.11644*.

[9] K. Höffner, S. Walter, E. Marx, R. Usbeck, J. Lehmann, and A.-C. N. Ngomo, "Survey on challenges of question answering in the semantic web," *Semantic Web*, vol. 8, no. 6, pp. 895–920, Aug. 2017.

[10] K. Xu, S. Zhang, Y. Feng, and D. Zhao, "Answering natural language questions via phrasal semantic parsing," in *Natural Language Processing and Chinese Computing*. Shenzhen, China: Springer, 2014, pp. 333–344.

[11] 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, Apr. 2020.

[12] 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.

[13] J.-H. Lin and E. J.-L. Lu, "An NMT-based approach to translate natural language questions to SPARQL queries," in *IT Convergence and Security: Proceedings of ICITCS 2021*. London, U.K.: Springer, 2021, pp. 3–11.

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

[15] Y.-H. Chen, E. J. Lu, and Y.-Y. Lin, "Efficient SPARQL queries generator for question answering systems," *IEEE Access*, vol. 10, pp. 99850–99860, 2022.

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

[17] M.-L. Zhang and Z.-H. Zhou, "A review on multi-label learning algorithms," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 8, pp. 1819–1837, Aug. 2014.

[18] Z.-H. Zhou, M.-L. Zhang, S.-J. Huang, and Y.-F. Li, "Multi-instance multi-label learning," *Artif. Intell.*, vol. 176, pp. 2291–2320, Jan. 2012.

[19] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains for multi-label classification," *Mach. Learn.*, vol. 85, no. 3, pp. 333–359, Dec. 2011.

[20] R. Usbeck, A.-C. N. Ngomo, B. Haarmann, A. Krithara, M. Röder, and G. Napolitano, "7th challenge on question answering over linked data (QALD-7)," in *Semantic Web Challenges: 4th SemWebEval Challenge at ESWC 2017*. Portorož, Slovenia: Springer, May 2017, pp. 59–69.

[21] 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.

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

[23] 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.*, 2017, pp. 210–218.

[24] S.-W. Chen, "The study of transition-based phrase DAG parsing to generate RDF queries for natural language questions," Nat. Chung Hsing Univ., 2023.

[25] E. J.-L. Lu, H.-Y. Kao, and T.-A. Ou, "A name-entity linker for question answering over linked data," in *Proc. TANET Taiwan Acad. Netw. Conf.* Taichung, Taiwan: TANAT, 2009.

[26] 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.

[27] N. Reimers and I. Gurevych, "Optimal hyperparameters for deep LSTM-networks for sequence labeling tasks," 2017, *arXiv:1707.06799*.

[28] K. Singh, A. S. Radhakrishna, A. Both, S. Shekarpour, I. Lytra, R. Usbeck, A. Vyas, A. Khikmatullaev, D. Punjani, and C. Lange, "Why reinvent the wheel: Let's build question answering systems together," in *Proc. World Wide Web Conf.*, 2018, pp. 1247–1256.

[29] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2014, pp. 1532–1543.

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

[31] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 26, 2013, pp. 1–9.

[32] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of English: The Penn treebank," *Comput. Linguistics*, vol. 19, no. 2, pp. 313–330, 1993.
[33] S. Vakulenko, J. D. F. Garcia, A. Polleres, M. D. Rijke, and M. Cochez, "Message passing for complex question answering over knowledge graphs," in *Proc. 28th ACM Int. Conf. Inf. Knowl. Manag.*, 2019, pp. 1431–1440.
[34] I. Abdelaziz, S. Ravishankar, P. Kapanipathi, S. Roukos, and A. Gray, "A semantic parsing and reasoning-based approach to knowledge base question answering," in *Proc. AAAI Conf. Artif. Intell.*, May 2021, vol. 35, no. 18, pp. 15985–15987.
[35] 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 2018.

YI-HUI CHEN (Member, IEEE) received the Ph.D. degree in computer science and information engineering from National Chung Cheng University. She was with the Academia Sinica, as a Postdoctoral Fellow. She was a Research Scientist with the IBM's Taiwan Collaboratory Research Center. After that, she was with 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.

JIN-DE LIN received the M.S. degree from the Department of Management Information Systems, National Chung Hsing University. His research interests include the semantic web and natural language processing.