Open Domain Question Answering over Knowledge Graphs Using Keyword Search, Answer Type Prediction, SPARQL and Pre-trained Neural Models

Christos Nikas[1,2], Pavlos Fafalios[1(✉)], and Yannis Tzitzikas[1,2]

[1] Information Systems Laboratory, FORTH-ICS, Heraklion, Greece
{cnikas,fafalios,tzitzik}@ics.forth.gr
[2] Computer Science Department, University of Crete, Heraklion, Greece

Abstract. Question Answering (QA) in vague or complex open domain information needs is hard to be adequate, satisfying and pleasing for end users. In this paper we investigate an approach where QA *complements* a general purpose interactive keyword search system over RDF. We describe the role of QA in that context, and we detail and evaluate a pipeline for QA that involves a general purpose entity search service over RDF, answer type prediction, entity enrichment through SPARQL, and pre-trained neural models. The fact that we start from a general purpose keyword search over RDF, makes the proposed pipeline widely applicable and realistic, in the sense that it does not pre-suppose the availability of knowledge graph-specific training dataset. We evaluate various aspects of the pipeline, including the effect of answer type prediction, as well as the performance of QA over existing benchmarks. The results show that, even by using different data sources for training, the proposed pipeline achieves a satisfactory performance. Moreover we show that the ranking of entities for QA can improve the entity ranking.

Keywords: Open domain question answering · Knowledge graphs · Keyword search · Answer type prediction

1 Introduction

Question answering over knowledge bases (KBQA) is an important NLP task because of the rapid growth of knowledge bases (KBs) on the web and the commercial value they bring for real-world applications [18]. In knowledge bases, where data is represented as a graph, e.g. using the Resource Description Framework (RDF), methods relying on Graph Processing and SPARQL Query Generation are adopted in order to extract the desired information [5]. At the same time, neural network-based (NN-based) Question Answering (QA) methods have received increasing attention in recent years and have already achieved very good results [1]. Although such methods require large amounts of training data, pre-trained language models [4] have become available, that can be fine-tuned on

A. Hotho et al. (Eds.): ISWC 2021, LNCS 12922, pp. 235–251, 2021.
https://doi.org/10.1007/978-3-030-88361-4_14

specific data to obtain high quality results for various tasks, such as sequence classification and extractive QA.

Nevertheless in vague or complex open domain information needs and questions, that require considering and joining facts, QA methods are not that good [19]. At the same time, there is not a single QA component per QA task that is perfect, and the performance of a QA component varies based on questions with different features; see the detailed analysis in [24]. Indeed, all the 12 baseline approaches evaluated over the QALD-2 dataset (containing natural language questions) of the DBpedia-Entity benchmark [7] achieve NDCG@10 (Normalized Discounted Cumulative Gain at rank 10) less than 0.37.[1]

Since we are interested in a general purpose and widely applicable method for open domain QA, in this paper we investigate an approach where at its core has a *keyword search* service. This allows exploiting the wealth of techniques related to text pre-processing, retrieval and language models, thus tackling some of the weaknesses of current components, like those identified in [24], related to the upper/lowercase of named entities, the implicit entity names (that NER tools usually fail to identify due to the various morphological variations), the abbreviations in named entities, and others. In addition, not all question intentions can be identified and mapped to the correct SPARQL statement (e.g. questions that can be answered by the textual descriptions in the `rdfs:comment`), therefore the exploitation of IR and NLP techniques is indispensable. In brief keyword search can provide relevant hits for any kind of information need, and there are already scalable and effective approaches for keyword search over RDF [10].

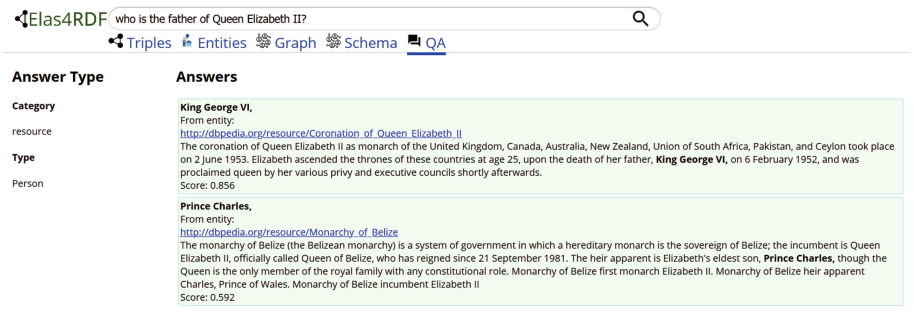

Fig. 1. Open domain QA over knowledge graphs as part of an interactive keyword search system over RDF.

As application context, we consider a *multi-perspective* keyword search over RDF, like the one presented in [17], that provides perspectives (tabs) that show the more relevant triples, the more relevant entities, graphical visualizations that show how the top-ranked triples are connected, and schema-based filterings. In such a context, the QA tab (as depicted in Fig. 1) is expected to provide a short

[1] https://github.com/iai-group/DBpedia-Entity.

and concise answer, if that is feasible. We could therefore say that we investigate an approach for open-domain QA over Knowledge Bases that could complement general purpose interactive keyword search over RDF. In such a dynamic context, we cannot expect that a training dataset is available for the knowledge base, and especially when the same approach needs to be deployed over another knowledge base.

We present a QA approach that relies on: (i) an entity search system to retrieve unstructured textual descriptions for entities, (ii) a Semantic Answer Type prediction component to predict the answer type, (iii) SPARQL to retrieve structured information that matches the predicted answer type, (iv) an entity enrichment component to expand the textual description with the information retrieved from the triplestore, and (v) a powerful language model fine-tuned for QA to extract the final answers.

In brief, given a natural language question, we first retrieve the top-k entities and their textual descriptions (through keyword search), then we get the triples only of these entities that have the predicted answer type, then we generate natural language sentences and we apply extractive QA using a pre-trained neural model, as illustrated in Fig. 2.

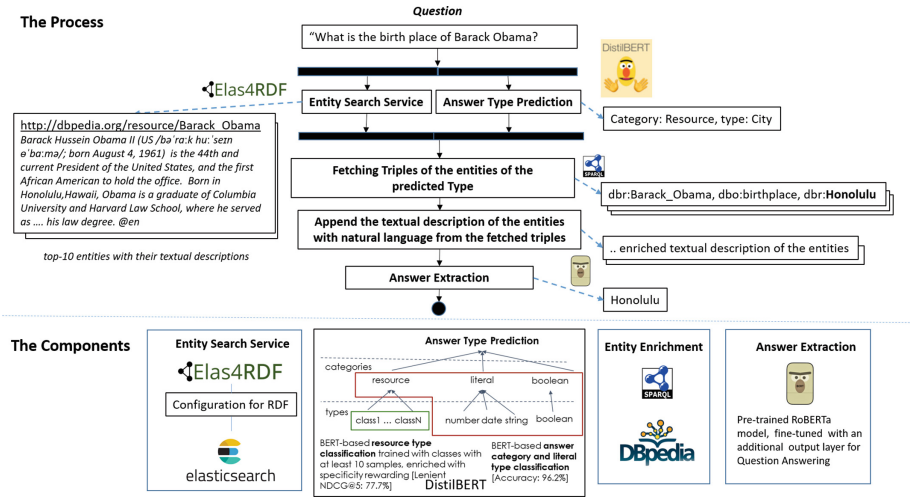

Fig. 2. The considered QA process and components.

Related research questions are: (a) How good can the QA pipeline over DBpedia be, in comparison to approaches and benchmarks over a different knowledge graph (in our case Freebase)? (b) How does Answer Type Prediction affect the quality of QA? (c) How can answers from this QA pipeline contribute to the entity retrieval task over DBpedia-Entity dataset [7], and entity ranking in general?

The results of our evaluation indicate that the answers generated by this approach provide additional value for entity search when combined with the initial entities retrieved by the search service used. Our approach can also perform well on difficult QA datasets (> 52% Accuracy), without having been trained on the specific datasets, but relying on the structured and unstructured information retrieval methods that we use. In brief, the main contribution of this work are: (a) we investigate a process for QA in the context of an keyword search access paradigm, (b) we detail the QA pipeline that comprises components for Entity Retrieval, Answer Type Prediction, Entity Enrichment, and Answer Extraction, (c) we evaluate the pipeline over multiple datasets, showcasing the value added by our approach. To the best of our knowledge, no previous work uses KBQA within an interactive search system over RDF where QA complements the other perspectives (and thus consistency with the input that feeds all perspectives is required), nor evaluated the effect of Query Answer Type prediction. The source code of our implementation and a running demo are publicly accessible.[2]

The rest of this paper is organized as follows: Sect. 2 provides an overview of the approach, Sect. 3 describes Answer Type Prediction, Sect. 4 describes Entity Retrieval and Extraction, Sect. 5 describes Answer Extraction, Sect. 6 focuses on evaluation, Sect. 7 describes related work, and finally, Sect. 8 concludes the paper and identifies issues for future research.

2 Overview of the Approach

The QA pipeline can be summarized as follows: we retrieve the top-k entities and their textual descriptions (through search), then we get the triples only of these entities that have the predicted answer type, then from these triples we generate natural language sentences for enriching the textual descriptions, and finally we apply extractive QA using neural networks. Consequently the pipeline is supported by 4 main components: Entity Search Service (for Entity Retrieval), Answer Type Prediction, Entity Enrichment, and Answer Extraction (Fig. 2).

First, given a natural language question, we retrieve a set of entities relevant to the question from DBpedia, along with a short description of each entity, using the Elas4RDF search service [10]. We query this service with the input question after removing stop words. The output of this stage is a list of entities described by their URI and a short textual description of the entity, extracted by a descriptive (for the entity) property, in our case `rdfs:comment`. The number of retrieved entities is set to 10, but it can be adjusted. A higher number of entities could yield more useful answers, but will require more time to be processed.

In parallel, we predict the answer type of the input natural language question by extending and improving a previous work on Answer Type Prediction [16] (this step is detailed in Sect. 3).

[2] Source code: https://github.com/isl/elas4rdfdemo, Demo: https://demos.isl.ics.forth.gr/elas4rdf/.

Then we expand the description of each entity with information from RDF nodes matching the predicted answer type by running SPARQL queries at real-time (this step is described in Sect. 4).

Finally, we use a RoBERTa [11] model fine-tuned on the SQuAD-2 dataset [20] to perform extractive QA for the input question using the extended description of each entity. Therefore, we obtain an natural language answer from each retrieved entity. Finally, we rank the answers using the score from the output of the model and present them on the user interface of the keyword search system (more in Sect. 5).

3 Answer Type Prediction

Here we describe how we perform Answer Type Prediction, i.e. how we predict the type of the answer of a natural language question, given the question. In comparison to [16], in our work we use two classifiers (instead of three) by integrating the *literal type prediction* classifier and the *category prediction* classifier. With this change, we simplify the approach, and reduce memory footprint (more in Sect. 3.2). Given the real-time context of out approach, we also use DistilBERT instead of BERT to achieve better response time and efficiency while maintaining high performance (more in Sect. 3.5).

3.1 Overview

The task is split in two stages: *Category prediction* and *Type prediction*. In particular, we model the problem as a two-stage classification task: in the first step the task is to predict the general category of the answer (*resource*, *literal*, or *boolean*), while in the second step the task is to predict the particular answer type (*number*, *date*, *string*, or a particular *resource class* from a target ontology).

We use the two datasets provided by SMART Task [14], one using the DBpedia ontology and the other using the Wikidata ontology. Both follow the below structure: Each question has (a) a question id, (b) a question text in natural language, (c) an answer category (*resource*/*literal*/*boolean*), and (d) an answer type (which depends on the answer category). If the category is *resource*, answer types are ontology classes from either the DBpedia ontology (∼760 classes) or the Wikidata ontology (∼50K classes). If the category is *literal*, answer types are either *number*, *date*, or *string*. Finally, if the category is *boolean*, answer type is always *boolean*. An excerpt from this dataset is shown below:


```
[ {
  "id": "dbpedia_14427",
  "question": "What is the name of the opera based on Twelfth Night?",
  "category": "resource",
  "type": ["dbo:Opera", "dbo:MusicalWork", "dbo:Work" ]
 },{
  "id": "dbpedia_23480",
  "question": "Do Prince Harry and Prince William have the same parents?",
```

```
  "category": "boolean",
  "type": ["boolean"]
 } ]
```


With respect to the size of the datasets, the DBpedia dataset contains 21,964 questions (train: 17,571, test: 4,393) and Wikidata 22,822 questions (train: 18,251, test: 4,571). The DBpedia training set consists of 9,584 resource, 2,799 boolean, and 5,188 literal questions. The Wikidata training set consists of 11,683 resource, 2,139 boolean, and 4,429 literal questions.

For question category and type prediction we use two DistilBERT sequence classification models. We choose DistilBERT instead of BERT to reduce memory footprint and time required to answer a question.

3.2 Question Category and Literal Type Prediction

A question can belong to one of the following three categories: (1) boolean, (2) literal, (3) resource. *Boolean questions* (also referred to as Confirmation questions) only have 'yes' or 'no' as an answer (e.g. *"Does the Owyhee river flow into Oregon?"*). Thus, there is no further classification for this category of questions. *Resource questions* have a specific fact as an answer (e.g. *"What is the highest mountain in Italy?"*) that can be described by a class in an ontology (e.g. http://dbpedia.org/ontology/Mountain). *Literal questions* have a literal value as answer, which can be a *number, string*, or *date* (e.g. *"Which is the cruise speed of the airbus A340?"*).

To detect question categories, we fine-tune a DistilBERT model using the Huggingface PyTorch implementation.[3] We choose a BERT-based model because we approach answer type prediction as a classification problem where each question is a sequence of words.

Since we only use three types to classify literal questions, we integrate literal type prediction into the same classifier with category prediction, following the approach of [22]. By doing this, we save computing requirements and reduce memory footprint because we avoid using a different BERT classifier for literal type prediction. Therefore, this model classifies each question in one of the following five classes: 1) boolean, 2) literal date, 3) literal number, 4) literal string, 5) resource.

To fine tune the model we used the training datasets provided for the SMART task. Specifically, we used questions from both the DBpedia and the Wikidata dataset. Since the data is imbalanced for categories (13.7% boolean, 26.6% literal, 59.4% resource), we randomly sampled questions for each class so that all classes had the same number of samples.

As we will see below, this model achieves 97.7% accuracy on our test set in this prediction task.

[3] https://huggingface.co/transformers/.

3.3 Resource Answer Type Prediction

The prediction of the answer type of questions in the *resource* category is a more fine-grained (and thus more challenging) classification problem, because of the large number of types a question can be classified to (∼760 classes on DBpedia and ∼50K classes on Wikidata). Therefore, it is not effective to train a classifier on all the ontology classes, especially for open-domain tasks.

To reduce the number of possible types for classification, we selected a subset (C) of all ontology classes, based on the number of samples of each class in the training set. This subset C contains classes that have at least k occurrences in the training set. We set $k = 10$ as this number provides a good trade-off between number of classes and performance. The choice of this parameter is described more extensively in Sect. 3.4. The final number of classes in C is 88. Since we chose to train the system on a subset of all the classes, our classifier cannot handle questions with labels that are not included in this subset. To tackle this problem, we replace their labels with the labels of super classes that belong in C. Then we fine tune a DistilBERT model on them.

Since most questions in the dataset have several answer types ordered by specificity, according to the semantic hierarchy formed in the ontology, in the fine tuning stage we use these questions multiple times, one with each of the provided types as the label. The goal is to find an answer type that is as specific as possible for the question. However, the model may classify a question to a more general answer type in the ontology. To tackle this problem, we 'reward' (inspired by [3]) the predictions of the classes that lie below the top class. The reward of a class c is measured by the depth of the class in the hierarchy, specifically, $reward(c) = depth(c)/depth_{Max}$, where $depth(c)$ is the depth of c in its hierarchy, while $depth_{Max}$ is the maximum depth of the ontology (6 for DBpedia). This means that, after applying normalization and adding the rewards on the output of the model, the top class can be a sub-class that was originally ranked below a more general class. For example, for the question *"What is the television show whose company is Playtone and written by Erik Jendresen?"* the top 5 classes that the classifier predicts are: 1) Work, 2) TelevisionShow, 3) Film, 4) MusicalWork, 5) WrittenWork. Then rewards are applied to classes that are a subclass of Work. After applying the rewards, the top 5 classes are: 1) TelevisionShow, 2) Work, 3) Film, 4) Book, 5) MusicalWork. We can see that TelevisionShow is now the top prediction, which is both correct and more specific than the previous top prediction (Work).

3.4 Tuning of the k Parameter

To find the optimal value for the parameter k, which is the minimum sample size required to include a class in the subset of classes included in the classifier, we evaluated our system using 4 different values: 5, 10, 30 and 50. Table 1 shows the number of classes included in the classifier for each different value of k and the corresponding performance. We notice that the best results are obtained using k = 10, while the results for all other cases are slightly worse.

Table 1. Performance of *Resource Answer Type Prediction* for different values of k.

Value	Classes	NDCG@5	NDCG@10
5	180	0.775	0.765
10	151	0.786	0.778
30	79	0.785	0.772
50	55	0.785	0.748

3.5 Model Selection

DistilBERT [21] is a smaller general-purpose language representation model based on BERT [4]. A DistilBERT model can be 40% smaller in size than an equivalent BERT model, while retaining 97% of its language understanding capabilities and being 60% faster. We chose this model for category classification and answer type classification because the compromise in the language understanding capabilities is not significant for us, since our models perform well enough for the required tasks. At the same time, answer type prediction is part of a QA system that runs as part of a keyword search Web application, therefore answer time speed and memory footprint are important in this context.

4 Entity Enrichment

For *Resource* and *Literal* questions, as predicted by the Answer Type Prediction step, we exploit the SPARQL endpoint of the underlying KB to find facts about the retrieved entities that match the predicted answer type. In our case, since the entity retrieval stage works over DBpedia, we selected DBpedia, however any KB could be used. Then, we generate natural language sentences from these facts and append the sentences to the entity description.

For *Resource* questions, for each entity, we retrieve all RDF triples where the subject is the entity, and the object has an RDF type that matches the top type returned by the Answer Type Prediction component, or an equivalent class, using the following query:

```
select distinct str(?pl) as ?pLabel ?a where {
    <entity uri> ?p ?a .
    ?p rdfs:label ?pl .
    <answer type> owl:equivalentClass ?eq .
    ?a rdf:type ?eq .
         FILTER(lang(?pl) = 'en' || lang(?pl) = '') }
```

For *Literal Date* questions we retrieve triples where the property that connects the entity with the candidate answer has an `rdfs:range` equal to `xsd:date`.

For *Literal Number* and *String* questions we retrieve all triples where the subject is the entity and the object is a literal. Then we check programmatically if the object is numeric or a string depending on the answer type. We follow this process because not all literal RDF Nodes have an XSD Schema data type.

From the retrieved triples we use the label of the corresponding entity, the object which is a candidate answer, and the label of the property that connects the entity with this answer. Then we generate a sentence of the form *"entity_label + property_label + object"* and append it to the textual description of the entity.

5 Answer Extraction

This stage receives a list of entity URIs and their expanded textual descriptions. For each entity in the list, we generate an answer from the expanded entity description using a RoBERTa model for extractive QA from the *huggingface* transformers library[4]. Then, we sort the answers by their score and display them on the QA perspective of the web application, along with the answer category and type.

The model that we use is fine-tuned on the SQuAD dataset provided by `deepset.ai`[5]. RoBERTa (Robustly optimized BERT approach) is a retraining of BERT with improved training methodology, using 10 times more data and compute power. We chose this model over BERT because of the increased difficulty of the extractive QA task.

A few indicative examples of Q-A pairs follow: (Q: Who did Mozart write his four horn concertos for? A: Joseph Leutgeb), (Q: What things did Martin Luther King do? A: human rights advocate and community activist), (Q: When did Charles Goodyear invent rubber? A: 1839) (Q: Who is the father of Queen Elizabeth II? A: King George VI).

6 Evaluation

In Sect. 6.1 we evaluate our approach over WebQuestions [2], a benchmark consisting of popular questions asked on the web that are answerable by Freebase, a different knowledge base than DBpedia, which our system retrieves information from, so essentially we evaluate how good our approach for open domain QA is while retrieving information from a *different source* and without having been previously trained over this specific dataset. In Sect. 6.2 we investigate how the task of Answer Type Prediction affects the effectiveness of QA. In Sect. 6.3 we evaluate the performance of our approach as a standalone QA system over the DBpedia-Entity collection [7]. In Sect. 6.4 we evaluate how answers from our QA pipeline can contribute to the entity retrieval task over the DBpedia-Entity dataset, and entity ranking in general. In Sect. 6.5 we discuss the efficiency of the system and in Sect. 6.6 we provide a summary of the evaluation results.

6.1 Experiment 1: Webquestions

WebQuestions [2] is a popular dataset for benchmarking QA engines, especially ones that work on structured knowledge bases. It is a dataset of question-answer

[4] https://huggingface.co/transformers/.
[5] https://huggingface.co/deepset/roberta-base-squad2.

pairs obtained from non-experts. It contains 6,642 questions collected using the Google Suggest API to obtain questions that begin with a wh-word and contain exactly one entity. Answers were generated using Amazon Mechanical Turk. The AMT task requested that workers answer the question using only the Freebase page of the question's entity. An example of a question-answer pair is the following:

```
Question: "What countries are part of the UK?"
Answers: "Scotland","England","Wales","Northern Ireland"
```

To evaluate our approach over this benchmark, we obtained answers from our system for all 2,032 questions in the test collection. Then, we compute the following metrics:

- *Precision*: The percentage of terms retrieved as answers by our system, that are included in the correct answers, averaged over all questions
- *Recall*: The percentage of terms in the correct answers, that are also retrieved as answers by our system, averaged over all questions
- *F1*: The harmonic mean of precision and recall
- *Accuracy*: The percentage of questions that received at least one correct answer

For reasons of performance, we limit the number of facts returned by the SPARQL endpoint to 20 (more in Sect. 6.5). We compute the evaluation scores for different sets of answers of varying confidence by considering only answers that have a score above a specific threshold t and trying different values for t. The results are displayed in Table 2.

Table 2. Evaluation results over WebQuestions.

Threshold	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Precision	7.007	16.170	18.607	21.290	23.710	25.261	28.101	31.185	37.543	**43.363**
Recall	31.263	27.712	29.016	27.894	29.078	30.882	31.279	33.465	34.506	**40.477**
F1	9.710	16.957	19.074	19.695	21.664	23.224	25.039	28.356	31.443	**39.200**
Accuracy	**53.759**	47.597	47.570	46.893	47.697	48.031	47.867	48.765	52.380	52.174

We can see that a threshold value of 0.9 yields the best values for Precision, Recall and F1. Accuracy is higher for a threshold value of 0 because (as expected) including all answers (score >= 0) leads to a higher probability that at least one correct answer will be included, however the 0.9 threshold gives a close to the optimal accuracy. Overall, our system has a satisfactory performance even though it has not been previously trained on this specific dataset, such as the systems in codalab[6] (e.g., [9]), or end-to-end neural-based models (e.g., [13]).

[6] https://worksheets.codalab.org/worksheets/0xba659fe363cb46e7a505c5b6a774dc8a.

6.2 Without Answer Type Prediction

To examine the value that is added to this QA pipeline by the answer type prediction component, we evaluate our system over the same dataset and metrics as in Sect. 6.1, but without using the answer type prediction component.

Therefore, in this case, the text provided to the extractive QA model is the textual description of each entity retrieved by the entity search system, without being expanded with facts matching the answer type, as described in Sects. 3 and 4. We report the following results using the best value for the answer score threshold (0.9) determined in Experiment 1: Precision: 37.356, Recall: 32.966, F1: 32.181, Accuracy: 48.122. We see that results are lower by 4–8 percentage points, suggesting the positive effect of answer type prediction.

6.3 Experiment 2: DBpedia Entity: QA

DBpedia-Entity is a standard test collection for entity search over DBpedia [7]. It is meant for evaluating retrieval systems that return a ranked list of entities (DBpedia URIs) in response to a free text user query. This dataset contains named entity queries, keyword queries, list queries and QA queries. We consider the subset of QA queries, which contains 140 queries from the QALD-2 challenge (Question Answering over Linked Data) [12]. These are natural language questions that can be answered by DBpedia entities, for example, "Who is the mayor of Berlin? Each entity/answer is accompanied by a score in 3-point relevance scale: *highly relevant (2)* (the entity is a direct answer to the query), *relevant (1)* (the entity can be shown as an answer to the query, but not among the top results), *irrelevant (0)*.

Other systems that report results over this benchmark use the NDCG@10 and NDCG@100 metrics because they focus on entity search. In our case, since we use this benchmark for QA, we consider Precision scores, in order to find out whether the top answers returned by our system are relevant to the query.

We evaluate the performance of our approach as a standalone QA system for the task of entity search. To do this we compute the Precision scores at the values 1, 3 and 5. The results obtained for varying values of answer score threshold are given in Table 3.

Table 3. Precision @1, @3, @5 for varying answer score threshold over DBpedia-Entity.

Threshold	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
P@1	33.573	49.331	55.432	55.641	55.938	56.190	58.857	57.241	57.273	**69.444**
P@3	27.840	42.006	48.265	47.951	46.595	50.813	52.905	51.207	52.348	**69.444**
P@5	24.543	41.008	47.147	46.836	45.768	49.716	51.905	51.207	52.348	**69.444**

The results are good in the sense that more than 69% of answers are relevant to their corresponding questions. Below (in Sect. 6.4) we also explore how this component can improve the performance of a dedicated Entity Search system by adding the set of answers to the set of entities retrieved by the search system.

6.4 Experiment 3: DBpedia Entity: QA+RANKING

We use the DBpedia Entity dataset [7] to evaluate the performance of Elas4RDF as an entity retrieval system for QA. Our goal for this experiment is to find out how the answers retrieved using this work affect the performance of Elas4RDF for QA tasks. Therefore, we use the group of queries from the DBpedia Entity collection that are Natural Language Questions (e.g. "Who is the mayor of Berlin?"). This group contains 140 of the 467 total queries in the benchmark. Over this group of queries, we compute the NDCG scores (@10 and @100) for:

- Entities retrieved by the Elas4RDF search service [10]
- Entities retrieved by the Elas4RDF search service *combined* with high scoring answers from the QA tab

To fuse the set of answers from the QA tab with the set of entities from the search service, we retrieve all entities from the search service, then we select a number (a) of answers from the QA tab and add them to the list of entities. Each entity has a score computed by the search service and each answer a score computed by the QA component. All scores are in the range scale of 0 to 1. We try two approaches to compute these scores:

I Keep the score from each entity and answer as computed by the entity search system and the QA component.
II Sum scores for entities in both rankings.

Finally, we sort the list of combined entities and answers by these scores, and we keep the top 10 or 100 results, depending on the NDCG metric that we wish to compute.

The results are displayed in Tables 4 and 5. The first row (baseline) corresponds to results for the entities returned by the QA component when no additional answers have been added. The next rows correspond to results for varying number of top answers from the QA component added to the baseline. We can see that including answers from the QA tab improves the NDCG score in all cases. The highest improvement in almost all cases occurs when the top-5 answers are added to the list of entities.

Table 4. NDCG scores over Natural Language Questions of the DBpedia Entity collection for approach I: Keep Initial Scores

	NDCG@100		NDCG@10	
Answers added	Score	Difference	Score	Difference
0 (baseline)	0.325	0	0.325	0
1	0.352	0.027	0.352	0.027
3	0.372	0.047	0.353	0.028
5	**0.384**	**0.059**	**0.354**	**0.029**
10	0.382	0.057	0.353	0.028

Table 5. NDCG scores over Natural Language Questions of the DBpedia Entity collection for approach II: Sum Scores

	NDCG@100		NDCG@10	
Answers added	Score	Difference	Score	Difference
0 (baseline)	0.325	0	0.325	0
1	0.355	0.03	0.355	0.03
3	0.375	0.05	**0.358**	**0.033**
5	**0.387**	**0.062**	0.357	0.032
10	0.386	0.061	0.356	0.031

As regards the comparison of approaches I and II, we can see that approach II obtains better results with a small difference (0.003 improvement of NDCG@100 using 5 answers). The reason for this is that approach II handles cases were an answer is returned by both the entity search system and the QA component.

Overall we can say that our QA pipeline could be considered as a method for ranking entities in the context of entity search. In comparison to a "plain" entity search, our pipeline is computationally more expensive because of the memory and time requirements added by the answer type prediction, entity enrichment and answer extraction components (see Sect. 6.5), but it can give better results in certain cases. Specifically, it can improve NDCG@100 by 6.2% points.

6.5 Efficiency

While running, the system's memory footprint is approximately 1.4 GB, and it takes up 511 MB of space to store all required models. To evaluate the time required to answer a question, we record times for each step of the pipeline as well as the overall time required to provide the final answers for all (2,032) questions in the Webquestions dataset (Sect. 6.1) and compute their average. This experiment was performed on a machine with 6 physical cores running Debian Linux. We found that the average time for the Answer Type Prediction stage is 0.1 s, for the Entity Enrichment stage 3.9 s, for the Answer Extraction stage 4.3 s and the overall average time required to provide the final answers is 8.3 s. We can see that answer type prediction is the fastest stage, because it uses a lighter language model (DistilBERT) while the other 2 stages are quite slower, because of the response time of the SPARQL queries for Entity Enrichment and the larger language model used for Answer Extraction (Table 6).

Table 6. Average time cost for each stage of the pipeline

Answer type prediction	Entity enrichment	Answer extraction
0.1 s (1.2%)	3.9 s (47%)	4.3 s (51.8%)

One could highly improve the efficiency by using a locally hosted triplestore that would provide a faster response time. Moreover one could speed up the answer extraction stage by using the RoBERTa model on a GPU. Finally, the number of returned facts could also be limited by setting a maximum response size, or using more strict SPARQL queries (e.g., by ignoring the equivalent classes), or using equivalence-aware indexes like those described in [15].

6.6 Executive Summary

We summarize the evaluation results as follows: We have shown that our approach for open domain QA can obtain satisfactory results, i.e. 54% accuracy, 39% F1 over popular QA benchmarks, something that is very interesting because it does not follow a supervised end-to-end approach trained on the same knowledge base, but makes use of different information sources than those intended by the benchmarks. We have also showed that the answer type prediction and entity enrichment stages improve Precision by 6%, Recall by 7% and F1 score by 7% (over WebQuestions). In addition we have shown that our approach can be used in combination with an entity search system to improve entity search tasks by 6% NDCG@100 (over DBpedia Entity dataset).

7 Related Work

For a survey of QA approaches over knowledge bases see [5]. In general, systems have converged to two major approaches: (i) Semantic Parsing (SP), and (ii) Information Extraction (IE); the former focuses on question understanding and therefore attempts to convert sentences into their semantic representation, such as logical forms, while the latter (IE) approach aims at identifying topic (focus) entities in the input question and then, via pre-defined templates, map the question to the KB predicates, and finally, explore the KG neighborhood of the matched entities. Our approach cannot be classified to any of these two extremes: although it starts from keyword search (that has an IE flavor), in parallel it performs Answer Type Prediction (that has a SP flavor), it enriches the textual description with SPARQL-fetched triples of the entities of the predicted type (SP and IE-flavors), and then it exploits pre-trained Neural Networks for the extraction of the final answer.

In comparison to related work, e.g. see [24] for a recent overview of QA approaches over DBpedia, the most related works are: [8] which converts the natural language question into two subqueries: SPARQL query and keyword search. That work uses a keyword index for special keywords rather than a whole knowledge graph for keyword search and produces the final answer using an algorithm to combine SPARQL results and keyword search results. Another work regarding QA and Keyword Search is SINA [23]. That system performs query preprocessing to tokenize, remove stopwords and lemmatize terms in the query, then groups keywords into segments and generates conjunctive federated SPARQL queries to retrieve answers. In contrast to our approach, that work

relies fully on a SPARQL endpoint instead of using a dataset-specific index for keyword search. However, from our experience, and as stated in [8], not all query intentions can be identified and mapped to the correct SPARQL statement.

Finally, we should note that the effect of Answer Type Prediction has been investigated in entity search ([6] shows that it improves significantly NDCG@10), however, to the best of our knowledge, no other work has investigated how it affects QA over knowledge graphs. Moreover, as mentioned in the introductory section, to our knowledge no previous work uses KBQA within an interactive search system over RDF where QA complements the other perspectives (and thus consistency with the input that feeds all perspectives is required).

8 Concluding Remarks

Since QA over knowledge graphs is hard to be adequate, satisfying and pleasing for end users, in this paper we have investigated an approach for QA in a more realistic context, i.e. in the context of an interactive search system over RDF where QA complements the other perspectives that are given to the users. We start from the entity ranking that is offered by the keyword search system, and we build on top a pipeline for QA that involves SPARQL, semantic answer type prediction, and pre-trained neural networks. We have evaluated our approach over two different datasets and showcased the value it provides for QA and entity search tasks. We have shown that for open domain QA this approach achieves satisfactory results, i.e. 54% accuracy and 39% F1 over a popular QA benchmark (WebQuestions), even if (a) no training has been performed over this particular benchmark, and (b) the method uses a different information source (DBpedia) than the one intended by the benchmark (FreeBase).

We have also showed how the Answer Type Prediction and Entity Enrichment stages, do improve Precision by 6%, Recall by 7% and F1 score by 7% (over WebQuestions). Finally, we have shown that our approach can be used in combination with an entity search system to improve entity search tasks by 6% NDCG@100 (over DBpedia Entity dataset).

Overall, the proposed pipeline can be applied over large knowledge graphs, since the process starts from an efficient and effective keyword search system, while the next steps exploit pre-trained neural network models.

As regards future research, it is worth investigating more questions from the DBpedia Entity dataset (not only QA-related), to see whether the entity ranking is improved in all cases, and to investigate methods for further improving the effectiveness of the approach without limiting its applicability.

References

1. Abbasiantaeb, Z., Momtazi, S.: Text-based question answering from information retrieval and deep neural network perspectives: a survey (2020)
2. Berant, J., Chou, A., Frostig, R., Liang, P.: Semantic parsing on Freebase from question-answer pairs. In: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pp. 1533–1544 (Oct 2013)

3. Deng, J., Krause, J., Berg, A.C., Fei-Fei, L.: Hedging your bets: optimizing accuracy-specificity trade-offs in large scale visual recognition. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition, pp. 3450–3457. IEEE (2012)
4. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding (2018)
5. Dimitrakis, E., Sgontzos, K., Tzitzikas, Y.: A survey on question answering systems over linked data and documents. J. Intell. Inf. Syst., 1–27 (2019)
6. Garigliotti, D., Hasibi, F., Balog, K.: Identifying and exploiting target entity type information for ad hoc entity retrieval. Inf. Retrieval J. **22**(3), 285–323 (2019)
7. Hasibi, F., et al.: DBpedia-entity v2: A test collection for entity search. In: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 1265–1268 (2017)
8. Hu, X., Duan, J., Dang, D.: Natural language question answering over knowledge graph: the marriage of SPARQL query and keyword search. Knowl. Inf. Syst. (2021). https://doi.org/10.1007/s10115-020-01534-4
9. Jain, S.: Question answering over knowledge base using factual memory networks. In: Procs of the NAACL Student Research Workshop. Association for Computational Linguistics, San Diego, California, June 2016
10. Kadilierakis, G., Fafalios, P., Papadakos, P., Tzitzikas, Y.: Keyword search over RDF using document-centric information retrieval systems. In: Harth, A., et al. (eds.) ESWC 2020. LNCS, vol. 12123, pp. 121–137. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-49461-2_8
11. Liu, Y., et al.: RoBERTa: a robustly optimized BERT pretraining approach (2019)
12. Lopez, V., Unger, C., Cimiano, P., Motta, E.: Evaluating question answering over linked data. Web Semant. **21**, 3–13 (2013)
13. Lukovnikov, D., Fischer, A., Lehmann, J., Auer, S.: Neural network-based question answering over knowledge graphs on word and character level. In: Proceedings of the 26th International Conference on World Wide Web, pp. 1211–1220 (2017)
14. Mihindukulasooriya, N., Dubey, M., Gliozzo, A., Lehmann, J., Ngomo, A.C.N., Usbeck, R.: SeMantic AnsweR Type prediction task (SMART) at ISWC 2020 Semantic Web Challenge. CoRR/arXiv abs/2012.00555 (2020)
15. Mountantonakis, M., Tzitzikas, Y.: Content-based union and complement metrics for dataset search over RDF knowledge graphs. J. Data Inf. Q. (JDIQ) **12**(2), 1–31 (2020)
16. Nikas, C., Fafalios, P., Tzitzikas, Y.: Two-stage semantic answer type prediction for question answering using BERT and class-specificity rewarding. In: Proceedings of the SeMantic AnsweR Type prediction task (SMART) at ISWC 2020, pp. 19–28 (2020)
17. Nikas, C., Kadilierakis, G., Fafalios, P., Tzitzikas, Y.: Keyword search over RDF: is a single perspective enough? Big Data Cogn. Comput. **4**(3), 22 (2020)
18. Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., Taylor, J.: Industry-scale knowledge graphs: lessons and challenges. Queue **17**(2), 48–75 (2019)
19. Qi, P., Lee, H., Sido, O.T., Manning, C.D.: Retrieve, rerank, read, then iterate: answering open-domain questions of arbitrary complexity from text (2020)
20. Rajpurkar, P., Jia, R., Liang, P.: Know what you don't know: unanswerable questions for squad. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pp. 784–789 (2018)
21. Sanh, V., Debut, L., Chaumond, J., Wolf, T.: DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108 (2019)

22. Setty, V., Balog, K.: Semantic answer type prediction using BERT. In: Procs of the SeMantic AnsweR Type prediction task (SMART) at ISWC 2020 Semantic Web Challenge, vol. 2774, pp. 10–18 (2020). CEUR-WS.org
23. Shekarpour, S., Marx, E., Ngonga Ngomo, A.C., Auer, S.: SINA: semantic interpretation of user queries for question answering on interlinked data. J. Web Semant. **30**, 39–51 (2015)
24. Singh, K., Lytra, I., Radhakrishna, A.S., Shekarpour, S., Vidal, M.E., Lehmann, J.: No one is perfect: analysing the performance of question answering components over the DBpedia knowledge graph. J. Web Semant. **65**, 100594 (2020)