CBench: Towards Better Evaluation of Question Answering Over Knowledge Graphs

Abdelghny Orogat  
Carleton University  
abdellghny.orogat@carleton.ca

Isabelle Liu  
Carleton University  
isabelle.liu@carleton.ca

Ahmed El-Roby  
Carleton University  
anhmed.elroby@carleton.ca

ABSTRACT

Recently, there has been an increase in the number of knowledge graphs that can be only queried by experts. However, describing questions using structured queries is not straightforward for non-expert users who need to have sufficient knowledge about both the vocabulary and the structure of the queried knowledge graph, as well as the syntax of the structured query language used to describe the user’s information needs. The most popular approach introduced to overcome the aforementioned challenges is to use natural language to query these knowledge graphs. Although several question answering benchmarks can be used to evaluate question-answering systems over a number of popular knowledge graphs, choosing a benchmark to accurately assess the quality of a question answering system is a challenging task.

In this paper, we introduce CBench, an extensible, and more informative benchmarking suite for analyzing benchmarks and evaluating question answering systems. CBench can be used to analyze existing benchmarks with respect to several fine-grained linguistic, syntactic, and structural properties of the questions and queries in the benchmark. We show that existing benchmarks vary significantly with respect to these properties, deeming choosing a small subset of them unreliable in evaluating QA systems. Until further research improves the quality and comprehensiveness of benchmarks, CBench can be used to facilitate this evaluation using a set of popular benchmarks that can be augmented with other user-provided benchmarks. CBench not only evaluates a question answering system based on popular single-number metrics but also gives a detailed analysis of the linguistic, syntactic, and structural properties of answered and unanswered questions to better help developers of question answering systems to better understand where their system excels and where it struggles.

1 INTRODUCTION

Recent years witnessed an unprecedented growth in the number of knowledge graphs (KGs) [6, 10, 17, 33, 38, 53]. These knowledge graphs contain a plethora of information that can be potentially used for question answering (QA). However, finding answers in a KG is not an easy task. The user is required to have a detailed knowledge of the KG, and a structured query language to describe their questions in a structured format that can be used to find matches in the KG. This requirement limits the ability to ask questions to power users who have the necessary skills to write syntactically and semantically correct queries to accurately represent their information needs. The number of such users represents a tiny fraction of a potentially large userbase. To overcome this challenge, a large number of QA systems that let users describe their information needs using natural language were developed [14, 19, 21, 23, 29, 31, 42, 55–58]. In fact, over 62 QA systems have been developed since 2010 [25].

As a result of the popularity of QA over KGs, several benchmarks were introduced to evaluate QA systems [7, 9, 13, 37, 41, 43–46, 50]. These benchmarks typically include questions described in natural language, answers to the questions from the KG targeted by the benchmark, and possibly structured queries that return the previously mentioned answers. To evaluate a newly developed QA system, its developers need to choose from a large number of benchmarks (at least 17 at the time of writing this paper) to evaluate their system. Without a quantitative comparison that highlights the differences between these benchmarks, choosing a subset of them to evaluate a new QA system is mainly motivated by the ease of comparison to existing systems in the literature rather than by how effective a benchmark is in evaluating a QA system. In fact, existing benchmarks differ significantly from each other with respect to the following three points:

- How the benchmarks were created: Some benchmarks are manually created by human experts based on heuristics [3, 12, 13, 15, 43–47, 49, 50]. Other benchmarks are automatically generated from the KG [7, 37, 41].
- Metadata: This includes what KGs that the benchmark target, and the number of questions in the benchmark. Most benchmarks target a limited number of KGs (All benchmarks that we are aware of target only 5 KGs). Also, the number of questions in each benchmark varies significantly. Out of the benchmarks we are aware of, the benchmark with the smallest number of questions includes 150 questions, while the benchmark with the largest number of questions includes 108,442 questions.
- Linguistic, syntactical, and structural properties: The natural language questions have linguistic properties. Their corresponding queries (if they exist in the benchmark) also have syntactical and structural properties. In this paper, we reveal that existing benchmarks vary significantly with respect to these properties, which is reflected on the reported quality scores depending on the benchmark used.

Another issue with existing benchmarks is their limited usability. Today, benchmarks are used in the following fashion: The user parses the benchmark file, extract the questions and utilizes the QA system to find answers in the targeted KG, then compare the returned answers to the answers extracted from the benchmark file to calculate multiple evaluation scores like micro, macro, and global F-1 scores (discussed in Section 7). The user then examines the questions that the QA system failed to answer correctly and debug their code to identify why the QA system struggles with these questions. In that sense, the benchmark is used as a dataset that helps in producing the aforementioned scores with a lost potential of being more informative to its user by giving more details on the

\[^1\]Out of 20 QA systems that were developed in the past 5 years, only 6 are open-sourced, and only 3 systems are accessible via a functional web interface.
fine-grained properties of the processed questions, which will help
the user better understand how the QA system behaves.

In this paper, we introduce CBench, an extensible fine-grained
benchmark suite that overcomes the aforementioned challenges
facing accurate evaluation of QA systems. CBench can be used in
two modes: Benchmark Analysis Mode and QA Evaluation Mode.
CBench can be used in the first mode to perform a fine-grained
analysis on the natural language questions and queries in a set
of benchmarks selected by the user. CBench includes a total of
17 benchmarks targeting 5 KGs, and can be easily extended with
new benchmarks and KGs. For the structured queries, CBench ana-
lyzes several syntactical properties of structured queries like the
type of the query, the operators used, and the query size. CBench
also analyzes the structural properties (shapes) of the queries. For
the natural language questions, CBench analyzes several linguis-
tic properties like the type of question, and part-of-speech (PoS)
tags of each question token of the natural language question. In
our analysis, we surprisingly quantify high-degree variations with
respect to the linguistic, syntactical, and structural properties of
questions and queries among different benchmarks in the litera-
ture, and experimentally show that these differences result in an
inconsistent assessment of QA systems. These findings motivate
further research in building better benchmarks that address as much
fine-grained properties as possible to have a good coverage of real-
world questions that the QA systems will encounter in real-world
deployments.

Until such benchmarks are available, CBench facilitates evaluat-
ing QA systems in the second mode (QA Evaluation Mode), in
which the user can choose the benchmarks they wish to use to
evaluate their QA system. Then, CBench interacts with the QA
system using a set of well-defined APIs to send/receive question-
s/answers. In addition to reporting micro, macro, and global F-1
scores, CBench also analyzes all the questions in the chosen bench-
marks and their corresponding structured queries (if available).
Specifically, CBench returns (1) a detailed analysis of the proper-
ties of the queries that the evaluated QA system processed, and (2)
linguistically-similar natural language questions for any question
of interest (e.g., a question that the QA system failed to answer).
Using the two aforementioned types of output, the QA user can
either (1) identify common properties between questions that the
QA system struggles with (e.g., most of the questions have a spe-
cific query shape), or (2) identify obvious inconsistencies in the
processed questions. For example, using CBench, we were able
to quickly identify that one of the QA systems we evaluated was
able to answer the question “What is the capital of Cameroon?”
correctly, while it incorrectly answered “What is the capital of
Canada?”, which highlights an overfitting problem in their entity
recognition and relation mapping approaches. Being able to quickly
identify commonalities or inconsistencies will help the QA system
developers to quickly identify the QA system component that they
need to improve. Based on the insights provided by CBench, the
user can also use it in a Debugging Mode within the QA Evaluation
Mode, where they can control CBench’s output questions based on
any of the linguistic, syntactical, or structural properties of all the
questions and queries in CBench to better understand how their

QA system behaves in several controlled situations. For example,
the user can specify that CBench only outputs temporal questions
whose queries have a star-shape.

Our contributions in the CBench suite are:
• To the best of our knowledge, We are the first to introduce the
concept of fine-grained analysis of questions and queries in QA
benchmarks.
• Using our fine-grained analysis, we surprisingly identify a high
degree of variations among existing benchmarks with respect
to several linguistic, syntactical, and structural properties of
the natural language questions and their corresponding queries
in the benchmarks.
• To demonstrate the effects of such variations, we evaluate six
QA systems using CBench in the QA Evaluation Mode and show
that their quality scores vary significantly.
• We give important insights to QA systems’ researchers on
benchmark selection and QA evaluation metrics.

The rest of this paper is organized as follows: Section 2 presents
the preliminaries of this paper. Section 3 discusses the architec-
ture of CBench. Section 4 gives an overview of the benchmarks in
CBench. Section 5 presents how CBench analyzes the structured
queries in the benchmarks. Section 6 presents how CBench analyzes
the natural language questions. Section 7 discusses our experiments
on six QA systems using CBench. Section 10 concludes the paper.

2 PRELIMINARIES

Knowledge Graph (KG): A knowledge graph is a directed graph
KG = (V, E) consisting of a set of vertices V that represent enti-
ties, types, and literals, and a set of labeled edges E that connect
these vertices. RDF [2] is a popular representation model for KGs,
which organizes data as a set of triples in the form (s, p, o) where s
refers to subject, p refers to predicate, and o refers to object, such
that s, o ∈ V, and p ∈ E. SPARQL [1] is the structured query lan-
guage for querying RDF. Figure 1 visualizes a sample subgraph
from DBpedia [6].

Graph Pattern (gp): A Graph Pattern is a set of triple patterns. A
triple pattern can be generated from a triple by replacing the s, p,
or o by a variable from the universal set of variables U. A query
can include multiple graph patterns.

Query (q): A query is represented as the pair (GP, SM), where GP
is the set of graph patterns in the query and SM is the set of solution

https://github.com/aorogat/CBench
modifiers. The query returns an answer set \( A \) that (1) matches the given \( GP \) from a knowledge graph \( KG \), and (2) modified to conform to the the solution modifiers \( SM \) (e.g., select, order by, distinct, limit and offset, etc.). In SPARQL, a query with type \( Select \) is used to select all, or a subset of, the variables bound in the set of subgraphs that matches \( GP \). If type is \( Ask \) instead of \( Select \), it returns a Boolean answer; true if \( GP \) can be matched (\( A \) is a non-empty set).

**Natural Language Question (nlq):** A natural language question is represented as the list of all tokens of the question excluding white spaces and punctuation marks. Formally, \( nlq = \{ token_1, token_2, \ldots, token_m \} \), where \( m \) is the number of tokens in the question.

**Benchmark (B):** A benchmark \( B = \{ NLQ, Q, G \} \) consists of a set of natural language questions \( NLQ \), an optional set of formal queries \( Q \), and a set of gold standard answers \( G \) such that \( g_i \) is the set of gold standard answers for the question \( nlq_i \), whose corresponding query is \( q_i \).

**Example 1.** Following is an example of a natural language question that can be found in a benchmark \( (nlq_i) \): "Which, companies, have, more, than, 1, million, employees, or, founded, in, Beijing". The query \( g_i \) that is associated with this question is:

```
SELECT DISTINCT ?uri WHERE {
  ?uri a dbo:Company {  
    ?uri dbo:numberOfEmployees ?n .  
    FILTER ( ?n > 1000000 )  
  } UNION {  
    ?uri dbo:foundationPlace dbr:Beijing.  
  }
}
```

The answers \( g_i \) to this query based on the subgraph in Figure 1 are \{"Lenovo", "Indian Railways"\}.

### 3 OVERVIEW OF CBENCH

Figure 2 shows the architecture of CBench, which can be used in two modes: (1) The Benchmark Analysis Mode, where CBench can be used to perform a fine-grained analysis on the structured queries and the natural language questions on a set of benchmarks selected by the user, and (2) the QA Evaluation Mode, where CBench can be used to evaluate a QA system over the user-selected benchmarks providing deeper insights on how the QA system is performing.

**Benchmark Analysis Mode:** CBench includes 17 benchmarks from which the user can choose a subset for analysis. The user can also upload their own benchmarks to be included in the analysis. The Benchmark Builder passes the selected benchmarks (and the uploaded ones, if any) to the Q-Analyzer which carries out the syntactical and structural analysis of the queries (Section 5), and the linguistic analysis (Section 6). Finally, the Q-Analyzer returns the analysis report to the user.

**QA Evaluation Mode:** Just like the previous mode, the user selects a set of benchmarks and/or upload their own to evaluate the QA system. In addition, the user provides CBench with a URL for an endpoint that CBench can query. Other configuration parameters that are used in the evaluation (e.g., thresholds for calculating quality scores) are also chosen by the user prior to evaluation. To avoid the scenario where the selected benchmarks target different versions of the same KG (discussed in Section 4), the Benchmark Builder updates the answers of the queries in the selected benchmarks through the Benchmark Updater module, which queries the used KG that will be used for the evaluation to retrieve the updated answers. The updated benchmarks are then passed to the System Evaluator. The System Evaluator carries out three tasks: (1) Communicating with the QA system to collect the answers to the questions from the selected benchmarks, (2) calculating the micro, macro, and global F-1 scores (discussed in Section 7), and (3) retrieving the fine-grained analysis of the processed questions from the Q-Analyzer. The System Evaluator then outputs an interactive report that includes the scores and the analysis of the processed questions to the user. The user can choose to focus on specific questions to view all their fine-grained properties. CBench also finds other questions that are linguistically similar to the selected questions (discussed in Section 6). Using this feature, the user is able to quickly identify either common features or inconsistencies of unanswerable or incorrectly answered questions, which will help the user to understand which components of their QA system to improve. The user can also use CBench in the Debugging Mode, in which they can group questions/queries based on specific properties to evaluate their QA system in specific scenarios. For example, the user can choose to evaluate their QA system only on aggregate questions (e.g., How many) whose queries have a star-shape to investigate how their system deals with such questions.

The details of the configurations of CBench, how to add a new benchmark, and the API’s used for communication with the QA systems can be found in the system’s repository.

### 4 BENCHMARKS IN CBENCH

Before discussing the analysis of queries and natural language questions, we give an overview of the benchmarks that are included in CBench. CBench can also be augmented with other benchmarks provided by the user. Table 1 shows the benchmarks along with the number of questions in each and the KGs they target. QALD5 is an annual evaluation campaign for question answering that started in 2011. Therefore, it includes 9 benchmarks (QALD-1 to QALD-9). There are overlaps between these versions of QALD. As each question in any of the QALD benchmarks is associated with a query, there are cases where the natural language question is in a more recent benchmark but associated with a different query. Table 2 shows the number of repeated natural language questions for each benchmark and how many of those have repeated queries (i.e., no change at all in the benchmark entry). The second column shows the total number of questions in each benchmark. While the third and fourth columns show the number of questions borrowed from previous versions, the fifth column shows how many questions were inserted for the first time in the corresponding version of QALD. For example, QALD-9 has in total 408 questions. Only 6 questions are new. The other 402 questions are borrowed from all older versions, where 231 of the 402 questions use the same natural language questions but are associated with updated queries. The remaining 171 questions use the same natural language questions.

---

5The prefix dbr is bound to http://dbpedia.org/resource/  
The prefix dbo is bound to http://dbpedia.org/ontology/  
The prefix rdf is bound to http://www.w3.org/2000/01/rdf-schema#  
6https://github.com/aorogat/CBench  
7http://qald.aksw.org
Figure 2: CBench Architecture.

Table 1: The benchmarks used in CBench. DB refers to DBpedia, FB refers to Freebase, MB refers to MusicBrainz, WK refers to Wikidata, and LS refers to LinkedSpending. Benchmarks annotated with ★ do not include queries, and benchmarks annotated with † target only single-factoid questions.

| Benchmarks | #Qs | KG   | Version |
|------------|-----|------|---------|
| QALD-1 [43]| 199 | DB, MB | 3.6     |
| QALD-2 [15]| 344 | DB, MB | 3.7     |
| QALD-3 [13]| 397 | DB, MB | 3.8     |
| QALD-4 [44]| 321 | DB    | 3.9     |
| QALD-5 [45]| 334 | DB    | 2014    |
| QALD-6 [46]| 431 | DB, LS | 10-2015 |
| QALD-7 [50]| 530 | DB, WD | 04-2016 |
| QALD-8 [49]| 315 | DB    | 10-2016 |
| QALD-9 [47]| 408 | DB    | 10-2016 |
| LC-QuAD [41]| 4,998 | DB  | 04-2016 |
| WebQuestions [9] | 5,810 | FB  | 09-2015 |
| GraphQuestions [37] | 5,166 | FB  | 06-2013 |
| SimpleQuestions★ † [12] | 108,442 | FB2M, FB5M | |
| SimpleDBpediaQA★ † [7] | 43,086 | DB  | 10-2016 |
| TempQuestions★ † [28] | 1,271 | FB  | 09-2015 |
| ComplexQuestions★ † [4] | 150 | FB    | 09-2015 |
| ComQA★ [3] | 1,121,4 | Wikipedia | |

Table 2: QALDs Overlaps. #Qs is the total number of all questions. #SU is the number of Same questions but associated with Updated queries. #SS is the number of Same questions and associated with the Same queries. #New is the number of New questions and associated with New queries.

| #Qs | #SU | #SS | #New |
|-----|-----|-----|------|
| QALD-1 | 199 | -   | -    | 199  |
| QALD-2 | 344 | 150 | 0    | 194  |
| QALD-3 | 397 | 236 | 52   | 109  |
| QALD-4 | 321 | 146 | 6    | 169  |
| QALD-5 | 334 | 27  | 193  | 114  |
| QALD-6 | 431 | 317 | 0    | 114  |
| QALD-7 | 530 | 262 | 57   | 211  |
| QALD-8 | 315 | 174 | 100  | 41   |
| QALD-9 | 408 | 231 | 171  | 6    |

and the same queries. It is interesting to see that some questions are used in some version, left in a newer version and then reused again in a more recent version.

LC-QuAD is a semi-automated question generation dataset. SPARQL templates are automatically generated and are converted into natural question templates. These general templates are manually transformed into natural language questions. GraphQuestions is a set of questions that are generated in two steps. First, generating a set of graph-structured logical patterns from the KG, then transforming them into natural questions with the help of human annotators. WebQuestions is a set of questions obtained from non-experts. These questions are collected based on suggestions from Google Suggest API. The questions with answers from Freebase are taken and annotated by Amazon Mechanical Turk workers then converted to SPARQL queries by experts. SimpleQuestions is generated by shortlisting the set of facts from Freebase that can be converted to informative questions. Then, these elected facts were sampled and passed to annotators to manually generate natural language questions whose answers are the entities in these facts. SimpleDBpediaQA is a subset of the SimpleQuestions benchmark dataset created by mapping entities and predicates from Freebase to DBpedia. TempQuestions consists exclusively of temporal questions. These questions are extracted from Free917 [16], WebQuestions and ComplexQuestions. ComplexQuestions is multi-constraints question-answer pairs that have some questions from WebQuestions (596 questions) and some manually labelled questions. ComQA is a large benchmark that includes real questions taken from the WikiAnswers platform and present various challenging aspects such as compositionality, temporal reasoning, and comparisons.
Table 3: Percentage of keyword occurrences in queries for each benchmark.

| Element       | QALD   | LC-QuAD | Web   | Graph |
|---------------|--------|---------|-------|-------|
| Select        | 91.63% | 91.52%  | 100.00% | 100.00% |
| Ask           | 8.37%  | 8.48%   | 0.00%  | 0.00%  |
| Distinct      | 76.65% | 91.52%  | 99.98% | 0.00%  |
| Limit         | 6.51%  | 0.00%   | 0.02%  | 0.00%  |
| Offset        | 3.93%  | 0.00%   | 0.00%  | 0.00%  |
| Order By      | 5.99%  | 0.00%   | 0.02%  | 0.00%  |
| And           | 51.65% | 70.67%  | 37.65% | 41.75% |
| Filter        | 10.33% | 0.00%   | 99.62% | 100.00%|
| Union         | 6.10%  | 0.00%   | 0.36%  | 0.00%  |
| Optional      | 5.37%  | 0.00%   | 0.00%  | 0.00%  |
| Not Exists    | 0.21%  | 0.00%   | 0.00%  | 0.00%  |
| Minus         | 0.21%  | 0.00%   | 0.00%  | 0.00%  |
| Aggregators   | 5.27%  | 0.00%   | 0.00%  | 20.17% |
| Group By      | 5.27%  | 0.00%   | 0.00%  | 13.74% |
| Having        | 1.34%  | 0.00%   | 0.00%  | 0.00%  |

5 ANALYSIS OF STRUCTURED QUERIES

Most of the benchmarks discussed in Section 4 include structured queries that can be used to obtain the answers to the natural language questions in the benchmark. Inspired by prior work on the analysis of query logs of endpoints [11], we discuss how the Query Analyzer processes the SPARQL queries by focusing on both their syntactically and structural properties to present them in the final report shown to the user. We use CBench’s Benchmark Analysis Mode to process all queries from the 12 benchmarks that include SPARQL queries to highlight the high-degree variations among them, then we give our insights in light of these variations. In Section 7, we use CBench’s QA Evaluation mode to experimentally show the effects of these variations on the accurate assessment of QA systems.

5.1 Syntactical Analysis of Queries

In this section, we study the syntactical properties of the queries, which include the frequency of query keywords, the number of triple patterns in the query, and the frequency usage of operators.

5.1.1 Query Keywords. We count the frequency of query keywords in the benchmarks. The results are shown in Table 3. Due to the lack of space, we combine all the queries from QALD-1 to QALD-9 and report them under QALD. In the case of repetitive questions, we consider the most recent query. After deduplication, the QALD discussed here include 959 questions. We find that even within the 9 QALD benchmarks, variations still exist. We will highlight these variations later in this section. The first block in Table 3 reports the query types. In general, the majority of queries use the Select keyword across all benchmarks (at least 91% of queries). Ask queries whose answers are either true (if a solution matches the graph pattern in the query) or false (otherwise) represent 8.37% and 8.48% of the queries in QALD and LC-QuAD, respectively. GraphQuestions and WebQuestions do not include queries that use the keyword Ask.

The second block in Table 3 includes the keywords used as solution modifiers in the queries. We ignored the Reduced modifier as it is never used in all benchmarks. We notice that the majority of the queries use the Distinct keyword in all benchmarks except GraphQuestions, which is the only benchmark that does not use any of the four solution modifiers. The Limit, Offset and Ordered-By keywords are not frequently used in QALD, and almost non-existent in other benchmarks.

The third block in Table 3 has keywords used to describe the graph patterns as described in Section 2. The And operator, which represents the conjunctions of triple patterns, is often used across all benchmarks. However, there is a large variation in the percentage of queries that use the keyword, where the minimum percentage is 37.65% (WebQuestions), and the maximum percentage is 70.67% (LC-QuAD). The Filter keyword demonstrates an interesting case, where it is used in almost all questions in WebQuestions and GraphQuestions, but with a much smaller percentage in QALD and not used at all in LC-QuAD. The remaining keywords (Union, Optional, Not Exists, and Minus) are used in a small percentage of queries across all benchmarks. But again, there is a high degree of variation across all benchmarks in their usage for these keywords.

Lastly, Aggregators (e.g., count, max, etc.) and Group-By keywords are used only in QALD and GraphQuestions. The keyword Having is used in QALD only.

With respect to the 9 QALD benchmarks, variations also exist although they come from the same organization as shown in Table 4. For example, the Distinct keyword occurs in only 38.69% of the QALD-1 queries, whereas in the other QALD benchmarks, the Distinct keyword occurs in between 63% to 91% of the queries. The Limit keyword occurrences steadily increase from 3.17% in QALD-1 to 5.64% in QALD-9. The Offset keyword is approximately never used in QALD-1 to QALD-3, but occurs more frequently in the other QALD benchmarks (from 4% to 8%). The Filter and Optional keywords occur frequently in QALD-1 and QALD-2 (32.29% and 54.44% of the queries, respectively), while this percentage is significantly declined in other QALD benchmarks (does not exceed 11%).

5.1.2 Number of Triple Patterns. The Query Analyzer also counts the number of triple patterns in all the graph patterns of the queries as a measure of the size of the queries in the benchmarks. The total
number of triple patterns has been computed and categorized from 1 to 11+ triple patterns for every benchmark. Figure 3 shows that the queries with a low number of triple patterns (from 1 to 3) are dominant in all benchmarks except the GraphQuestions benchmark, in which there are no queries with 1 triple pattern. Noticeably, the earlier versions of QALD and the GraphQuestions benchmarks include longer queries when compared to the other benchmarks.

5.1.3 Query Operators. We also study the co-occurrences of the Filter, And, Union, and Optional operators in the queries. The results can be found in Table 5. The first block of the table shows the queries with graph patterns that have triple patterns with no operators (with a single triple pattern without filter), with Filter only, with And only, or with both Filter and And operators. The next row has their subtotal (conjunctive patterns with filters or CPF). Most of the queries (from 89.77% to 100.00%) in all benchmarks are CPF queries. However, the distribution of queries using the combinations of the two operators varies.

The second block in the table shows the co-occurrences of the Optional operator with different types of CPF queries. This operator is only used in the QALD benchmarks with an increase of +4.13% in the relative size.

Finally, The third block of the table shows that the Union operator is not used in the GraphQuestions and LC-QuAD, and is rarely used in WebQuestions, while it is more frequently used in QALD. For the QALD benchmarks, there are other combinations that are rare and therefore not included in this table. For example, (O,U), (O,F,U), (A,O,U), etc., Table 6 shows the results of each QALD version separately.

5.2 Structural Analysis

In addition to the syntactical analysis of the queries in the benchmarks, we also analyze the structural shapes of the queries of the following types of queries: 1. Conjunctive queries that can use only the And operator, denoted by CQ. 2. Conjunctive queries that can use both And and Filter operators, denoted by CQF. 3. Conjunctive queries that can use And, Filter and Optional operators, denoted by CQOF.

Cbench identifies eight different shapes of queries. Figure 4 illustrate these shapes\(^6\). The Single-Edge shape has only one edge. The Chain shape with length \( n \) is a series of edges \( \{x_0, x_1\}, \{x_1, x_2\}, \ldots, \{x_{n-1}, x_n\} \). The Cycle shape is like the Chain shape except that the first node in the chain is the same as the last node. The Chain-Set shape is a set of one or more unconnected chains. The Tree shape can have any connected nodes keeping only one path between any nodes.

---

\(^{6}\)Examples of questions, their corresponding queries, and their structural shapes can be viewed in the repository.
Table 6: The frequency of the operators used in QALDs queries: Filter (F), And (A), Optional (O), and Union (U).

| Operators | QALD-1 | QALD-2 | QALD-3 | QALD-4 | QALD-5 | QALD-6 | QALD-7 | QALD-8 | QALD-9 |
|-----------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| None      | 7.14%  | 10.37% | 35.53% | 44.41% | 48.43% | 53.50% | 63.06% | 53.05% |
| F         | 0.00%  | 0.37%  | 0.00%  | 0.00%  | 0.00%  | 0.00%  | 0.00%  | 0.00%  |
| A         | 50.60% | 35.19% | 50.14% | 39.92% | 40.46% | 39.76% | 32.96% | 32.17% | 36.04% |
| A, F      | 3.57%  | 3.70%  | 7.74%  | 6.05%  | 3.37%  | 3.61%  | 2.87%  | 4.06%  |
| CPF       | 61.31% | 49.63% | 93.41% | 90.73% | 90.80% | 91.57% | 90.07% | 98.09% | 93.15% |

Table 7: Cumulative shape analysis of \( CQ \), \( CQ_F \), \( CQ_O \) across all benchmarks.

| Shape   | QALD | LC-QuAD | Web | Graph |
|---------|------|---------|-----|-------|
|         | CQ   | CQ_F   | CQ_O| CQ   | CQ_F | CQ_O| CQ   | CQ_F | CQ_O|
| Single-Edge | 50.82% | 46.88% | 45.13% | 29.35% | 29.35% | 29.35% | 40.00% | 63.29% | 63.29% |
| Chain   | 83.77% | 80.79% | 80.20% | 72.93% | 72.91% | 72.91% | 80.00% | 90.98% | 90.98% |
| Chain-Set | 83.90% | 81.60% | 80.97% | 72.93% | 72.91% | 72.91% | 80.00% | 90.98% | 90.98% |
| Star    | 14.84% | 15.86% | 16.59% | 27.05% | 27.06% | 27.06% | 20.00% | 8.61%  | 8.61%  |
| Tree    | 99.75% | 97.80% | 97.90% | 100.00% | 100.00% | 100.00% | 99.74% | 97.74% | 97.74% |
| Forest  | 100.00% | 99.54% | 99.56% | 100.00% | 100.00% | 100.00% | 99.98% | 99.98% | 99.98% |
| Cycle   | 0.13%  | 0.12%  | 0.11%  | 0.00%  | 0.00%  | 0.00%  | 0.11%  | 0.11%  | 0.11%  |
| Flower  | 99.87% | 98.15% | 98.23% | 100.00% | 100.00% | 100.00% | 99.98% | 99.98% | 99.98% |

Figure 4: The different shapes recognized by CBench

Table 7: Cumulative shape analysis of \( CQ \), \( CQ_F \), \( CQ_O \) across all benchmarks.

Two nodes. The Star shape is a special case of the Tree shape where there exists exactly one node with more than 2 neighbors. The

Figure 4: The different shapes recognized by CBench

This query corresponds to the NLQ: Which building after the Burj Khalifa has the most floors? As shown in the query, there are two chains. The first chain (a single triple pattern) finds the number of

This query corresponds to the NLQ: Which building after the Burj Khalifa has the most floors? As shown in the query, there are two chains. The first chain (a single triple pattern) finds the number of
The only exception is the GraphQuestions benchmark, which does not include any Single-Edge queries. It is interesting to see that a high percentage of the queries have the Chain shape. With the exception of GraphQuestions, this shape includes at least 72% of queries in all benchmarks. This observation along with the information from Figure 3 indicate that a high percentage of the queries have a Chain shape with a relatively short length of the chain. In contrast, the Chain-Set shape distribution has almost the same percentage as that of the Chain shape, which indicates that queries that have the Chain-Set shape, but not the Chain shape are rare in all benchmarks. The same observation applies to the Tree and Forest shapes, indicating rare occurrences of Forest shapes that are not trees. The Star shape distribution ranges between 8.61% and 30.7% across all benchmark. The Cycle shape is poorly represented in all benchmarks as every benchmark has not include any Single-Edge queries. For the different versions of QALD, Tables 8, 9, and 10 show the distribution of the discussed shapes for CQ, CQF, and CQOF queries respectively.

5.3 Insights on Benchmark Selection (1)

It is out of the scope of our study to determine which benchmark is quantitatively and qualitatively best in evaluating QA systems. However, given the previous fine-grained properties of queries in the discussed benchmarks, and until further research on benchmarking QA over KGs is done, we can give insights on what the user can expect when using the discussed benchmarks in QA evaluation. In this section, and without loss of generality, we discuss evaluating QA on the two most common KGs in the literature (DBpedia and Freebase), excluding simple-factoid questions.

QA over DBpedia: QALD-1 to QALD-9, and LC-QuAD target DBpedia with a mix of simple and complex queries. If the user intends to evaluate their QA system over DBpedia, we recommend using all benchmarks from QALD in addition to LC-QuAD, which is not a common practice in the literature. Based on the previously discussed properties, we notice that the QALD benchmarks have better coverage of all possible different cases of using query keywords (Table 3) and combination of operators (Table 5), while the LC-QuAD queries exclusively use only the Distinct along with conjunctions of triple patterns. Also, the QALD queries include a non-negligible number of queries that have more than three triple patterns (differs from one QALD benchmark to another), whereas 100% of the queries in LC-QuAD have at most three triple patterns (Figure 3). This is an indication that the QALD benchmarks include more complex queries, which may prove more challenging for QA systems. This is demonstrated by the more complex query shapes in QALD that do not exist in LC-QuAD (i.e., Forest, Cycle, and Flower). However, since the QALD benchmarks are manually created, they include 959 questions in total, whereas LC-QuAD includes almost 5000 questions. This indicates better linguistic and vocabulary coverage in LC-QuAD. Indeed, further investigations reveal that LC-QuAD’s queries span 6 times more resources (entities) and 2 times more predicates than QALD.

QA over Freebase: The two benchmarks targeting Freebase that include queries are WebQuestions and GraphQuestions. We recommend using GraphQuestions if the user intends to evaluate their QA system over Freebase. While the differences are minor in terms of the operators used in the benchmarks (Table 5), other syntactical and structural properties showcase the differences. GraphQuestions include aggregate queries, while WebQuestions do not (Table 3). Also, WebQuestions queries tend to be shorter (over 60% of queries are single-triple-pattern and almost 100% have no more than three triple patterns), whereas GraphQuestions have longer and more complex queries (more than 35% of queries have more than three triple patterns reaching 10 triple patterns) as shown in Figure 3. This is further showcased by the percentage of simpler query shapes for both benchmarks, where most of the queries in WebQuestions are chain queries, while GraphQuestions have higher percentage of more complex shapes. WebQuestions can be considered a better alternative to evaluate simple questions over Freebase because it includes single-triple-pattern queries, while GraphQuestions do not include such queries.

We will discuss our insights on benchmark selection with respect to the benchmarks that do not include queries in Section 6.

6 ANALYSIS OF NATURAL LANGUAGE QUESTIONS

SPARQL queries are used to represent the graph patterns, whose matches are the desired answers in the KG. Therefore, analyzing SPARQL queries gives deeper insights into the structure of the questions in the benchmarks. However, not all benchmarks include SPARQL queries that correspond to the natural language questions (as previously shown in Table 1). In Cbench, we also analyze the natural language questions via the NLQ Analyzer to provide linguistic-based insights on the questions in the benchmarks. The analysis of the natural language questions in Cbench focuses on the type of the natural language questions (Section 6.1). This analysis gives the user insights on benchmark selections (Section 6.2). In addition, to support the user with linguistic-based insights on the performance of the evaluated QA, Cbench provides the user with the most linguistically-similar questions to any question of interest. In order to support this feature, Cbench converts all natural language questions in the benchmarks to their vector space (Section 6.3). This feature is demonstrated in Section 7.

6.1 Type of Questions

In Cbench, the questions are categorized into:
### Table 8: Cumulative shape analysis of $C_Q$ across all versions of QALD benchmarks.

| Shape   | QALD-1 | QALD-2 | QALD-3 | QALD-4 | QALD-5 | QALD-6 | QALD-7 | QALD-8 | QALD-9 |
|---------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| Single-Edge | 13.64%  | 24.56% | 42.91% | 53.92% | 53.15% | 55.71% | 63.03% | 66.78% | 60.53% |
| Chain   | 50%    | 54.39% | 78.89% | 92.16% | 90.94% | 91.64% | 89.89% | 91.1%  | 91.81% |
| Chain-Set | 50%    | 54.39% | 78.89% | 92.16% | 90.94% | 91.64% | 90.16% | 91.1%  | 91.81% |
| Star    | 47.73% | 42.11% | 19.03% | 7.35%  | 8.27%  | 7.8%   | 9.31%  | 8.56%  | 7.89%  |
| Tree    | 98.86% | 100%   | 99.65% | 99.51% | 99.21% | 99.72% | 99.73% | 100%   | 99.71% |
| Forest  | 98.86% | 100%   | 100%   | 100%   | 100%   | 100%   | 100%   | 100%   | 100%   |
| Cycle   | 0%     | 0%     | 0.35%  | 0.49%  | 0.39%  | 0.28%  | 0%     | 0%     | 0.29%  |
| Flower  | 100%   | 100%   | 100%   | 100%   | 99.61% | 100%   | 99.73% | 100%   | 100%   |

### Table 9: Cumulative shape analysis of $C_Q_F$ across all versions of QALD benchmarks.

| Shape   | QALD-1 | QALD-2 | QALD-3 | QALD-4 | QALD-5 | QALD-6 | QALD-7 | QALD-8 | QALD-9 |
|---------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| Single-Edge | 11.65%  | 21.64% | 38.04% | 49.11% | 49.63% | 53.17% | 59.7%  | 63.93% | 57.02% |
| Chain   | 49.51% | 51.49% | 75.77% | 88.84% | 87.5%  | 89.15% | 86.65% | 89.1%  | 89.81% |
| Chain-Set | 50.49% | 51.49% | 76.99% | 89.73% | 88.24% | 89.68% | 87.91% | 89.51% | 89.81% |
| Star    | 47.57% | 40.3%  | 19.33% | 8.48%  | 9.93%  | 8.99%  | 10.33% | 9.18%  | 8.82%  |
| Tree    | 98.06% | 95.52% | 96.63% | 97.32% | 97.43% | 97.48% | 98.1%  | 98.69% | 98.35% |
| Forest  | 99.03% | 96.27% | 97.55% | 97.77% | 97.79% | 98.68% | 97.48% | 98.69% | 98.62% |
| Cycle   | 0%     | 0%     | 0.31%  | 0.45%  | 0.37%  | 0.26%  | 0%     | 0%     | 0.28%  |
| Flower  | 99.03% | 96.27% | 97.55% | 97.77% | 97.79% | 98.68% | 97.48% | 98.69% | 98.62% |

### Table 10: Cumulative shape analysis of $C_Q_O_F$ across all versions of QALD benchmarks.

| Shape   | QALD-1 | QALD-2 | QALD-3 | QALD-4 | QALD-5 | QALD-6 | QALD-7 | QALD-8 | QALD-9 |
|---------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| Single-Edge | 7.74%   | 11.55% | 38.3%  | 49.33% | 49.82% | 53.3%  | 57.04% | 64.05% | 57.14% |
| Chain   | 42.58% | 51%    | 76.9%  | 89.78% | 88.28% | 89.68% | 87.91% | 89.51% | 89.84% |
| Chain-Set | 43.87% | 51%    | 76.9%  | 89.78% | 88.28% | 89.68% | 87.91% | 89.51% | 89.84% |
| Star    | 47.57% | 40.3%  | 19.33% | 8.48%  | 9.93%  | 8.99%  | 10.33% | 9.18%  | 8.82%  |
| Tree    | 96.13% | 96.02% | 96.66% | 97.33% | 97.44% | 97.48% | 98.1%  | 98.69% | 98.35% |
| Forest  | 99.03% | 96.27% | 97.55% | 97.77% | 97.79% | 98.68% | 97.48% | 98.69% | 98.62% |
| Cycle   | 0%     | 0%     | 0.3%   | 0.45%  | 0.37%  | 0.26%  | 0%     | 0%     | 0.28%  |
| Flower  | 97.42% | 96.41% | 97.57% | 97.78% | 97.8%  | 98.68% | 97.61% | 98.69% | 98.63% |

- **Wh-questions**: the type of questions that starts with a wh-pronoun (i.e., *What*, *When*, *Where*, *Who*, *Whom*, *Which*, and *Whose*). For example, *Where was the first ford motor company located?* (WebQuestions) is a wh question. This excludes questions starting with the keyword *Why*, which targets non-factoid questions that are out of scope for QA over KGs.
- **How-questions**: Questions starting with the keyword *how* followed by many or an adjective, such as *How many different currencies are used in the places governed by the president of France?* (LC-QuAD) or *How tall is Michael Jordan?* (QALD-9). Other non-factoid How questions are not included.
- **Yes/No questions**: Questions that can be answered with *yes* or *no*: *Is Michelle Obama the wife of Barack Obama?* (QALD-9).
- **Requests**: Direct requests, such as *Can you name all the states of the US?* (GraphQuestions). Such a question can be considered as a rephrasing of a *What* question. However, we put these questions in a different category because they are usually handled differently in QA systems.
- **Topicalized questions**: Questions in which an entity or prepositional phrase is topicalized for the purpose of emphasis: *Adobe pdf supports how many computing platforms?* (GraphQuestions). Table 11 shows the percentage of questions of each question type across all benchmarks. We break the wh-questions into their sub-types. For wh-questions, we also include many questions that start with a preposition followed by a wh-word, such as *in which* or *to where*, because the single preposition can be easily moved to the end without changing the rest of the sentence. From these results, we observe that there is no dominant question type across the benchmarks and the distribution of query types vary widely.

### 6.2 Insights on Benchmark Selection (2)

In Section 5.3, we discussed our insights on benchmark selection based on our analysis of benchmarks that include queries.
section, we enrich this discussion on these benchmarks and add new insights on the benchmarks that do not include queries. Again, and without loss of generality, we discuss evaluating QA on the two most common KGs in the literature (DBpedia and Freebase), excluding simple-factoid questions.

**QA over DBpedia:** We previously mentioned that both QALD and LC-QuAD need to be used to evaluate a QA system over DBpedia. This recommendation is solidified by our observations in this section. Both benchmarks vary in terms of the types of questions, where each benchmark covers the types that are not covered by the other. For example, LC-QuAD does not include **When**-questions but QALD does, and QALD does not include **Whose**-questions but LC-QuAD does.

**QA over Freebase:** Among the benchmarks that target Freebase, **GraphQuestions** is the most comprehensive being the only benchmark that covers all question types. In contrast, the other benchmarks (**WebQuestions**, **TempQuestions**, **ComQA**, and **ComplexQuestions**) miss between 4 and 6 types. Another advantage of using **GraphQuestions** is that it includes different utterances of the same question, which further challenges a QA system. For example, “What celebrities are imitated by Will Ferrell?”, “Who did Will Ferrell impersonate?”, etc. It is interesting to notice that **GraphQuestions** include yes/no questions. However, the queries and answers do not reflect this type of questions by not using the SPARQL’s `ASK` keyword and having non-binary answers. For example, the question "*Are there any digital cameras with the maximum aperture reaching 5.2?*" is answered by a list of these cameras instead of yes/no answer. Prior to CBench, the user needed to adjust for this scenario by changing the answer in the benchmark to either Yes or No. CBench handles this scenario for the user. **TempQuestions** focuses on questions that include temporal aspects. So, it can be used as an auxiliary benchmark to another main benchmark (e.g., **GraphQuestions**). The number of questions in **ComplexQuestions** (150) and the number of types not included (6) are the main disadvantages of using it.

### 6.3 Linguistically-Similar Questions

Each token in the question is assigned a tag based on both the token itself and its context in the question. This process is known as Part-of-Speech (PoS) tagging. There have been numerous works in the literature on PoS tagging [18]. State-of-the-art approaches use a bidirectional recurrent neural network (BiLSTMs) and a subsequent conditional random field (CRF) decoding layer in combination with word embeddings that are trained over a large corpus of text [5, 32]. In CBench, we utilize a pre-trained model [26] to annotate each token in a question with its predicted PoS tag. However, CBench can be used with any PoS tagger. We define the `tag dictionary td = {t₁, t₂, …, tₙ}` as the set of all possible tags (of size $n$) that can be assigned to a question token. We use two types of tags: (1) The Universal POS (UPOS)$^7$, and (2) the more comprehensive Penn Treebank tag set [54]. The simple UPOS tags are the common representations for the word types, such as **NOUN**, **VERB**, and **ADJ**, and do not code for any morphological features (i.e. the structure of words such as stems, root words, prefixes, and suffixes). On the other hand, the Penn Treebank tags are more specific. For example, the **VERB** tag is further divided into **MD** - auxiliary, **VB** - base form, **VBD** - past tense, **VBG** - present participle, **VBN** - past participle, **VBP** - non-3rd person singular present, and **VBZ** - 3rd person singular present. We denote the tag dictionary for UPOS as $td_{UPOS}$, where $|td_{UPOS}| = 17$, and the tag dictionary for Penn Treebank as $td_{penn}$, where $|td_{penn}| = 36$. In section 7, we qualitatively evaluate both choices of tag types in CBench.

Due to the richness of natural language, it is not expected that a sequence of PoS tags will reoccur frequently in the limited set of questions in the benchmarks. Therefore, for this linguistic feature (PoStags), we use a function $f_{tag}$ that embeds the previously defined natural language representation (list of tokens) into a fixed-size vector space whose number of dimensions $n$ is equal to the size of the tag dictionary used. Algorithm 1 illustrates this function. The function uses the PoS tagger to tag each token in the question, then updates the frequency of the tag in the final vector representation. This representation not only captures the definition and the context

---

7https://universaldependencies.org/u/pos/
Algorithm 1: Vectorization of natural language questions using PoS tagging

\[
\text{input} : \text{List of tokens that represent the natural language question tokens, tag dictionary } td, \text{ PoS tagger }\text{tagger} \\
\text{output}: \text{Fixed-size vector representation } pos\_freq[|td|] \\
1 \text{Initialize } pos\_tag = [1] = 0; \\
2 \text{for token in tokens do} \\
3 \quad pos\_freq[\text{indexOf}\text{(tagger(token))}++]
4 \text{end} \\
5 \text{Return } pos\_freq;
\]

of each token but also indirectly captures the length of the question by counting the frequencies of the occurrences of the tags.

We use the obtained vector representation to calculate how syntactically similar a pair of questions is. In CBench, we use the euclidean distance measure to represent how dissimilar two questions are. Formally, between two vector representations of questions \( p \) and \( q \) is calculated as\( d(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \), where \( n = |td| \). This definition of distance is used to find the \( k \) most syntactically close questions to a question of interest (e.g., a question that is incorrectly answered). The \( k \) questions are shown to the user marked with whether they are correctly or incorrectly answered. To quickly find the \( k \) nearest questions to a question of interest, we pre-compute a distance matrix between each pair of questions in the benchmark. This one-time process takes less than a minute.

It is worth noting that in addition to PoS tagging, we also implemented a similar approach based on dependency parse trees of the questions in the benchmark to map the questions to their corresponding vectorized representations before calculating the pair-wise euclidean distances. We did not observe major changes in the output (discussed in Section 7).

7 EVALUATION

In this section, we experimentally evaluate six QA systems over different benchmarks using CBench. We show how the variations in the previously discussed properties of the questions/queries affect the evaluation of QA systems (Section 7.1). We then discuss our insights on evaluating the QA systems using CBench (Section 7.2). We also use CBench to perform a fine-grained analysis of the questions processed by the QA systems (Section 7.3). Finally, we qualitatively evaluate the different settings for highlighting linguistically-close questions to a question of interest (Section 7.4).

Evaluation Environment: All experiments were run on a machine having an Intel(R) Core(TM) i9-9900K CPU@3.60GHz and 16 MB cache, 32 GB RAM, running on Ubuntu 18.04.

QA Systems: We evaluate six QA systems [20, 22, 27, 30, 36, 40]. Three systems are remotely located and accessed via web services and three systems are run locally on the same computer as CBench. All systems are evaluated over the most recent versions of DBpedia and Wikidata.

- WDAqua [20]: WDAqua is accessed remotely and targets DBpedia and Wikidata. It ignores the syntax of the questions and focuses only on the semantics of the words in the question.
- gAnswer [27, 59]: gAnswer is accessed remotely and targets DBpedia only. It is based on finding a semantic graph to represent the question’s intention and then maps the problem to a subgraph matching problem over the KG.
- QAnary [35, 36]: QAnary is a modular QA framework that breaks the QA pipeline into independent modules that can be easily replaced to improve the overall quality of QA. Although there are many open-source modules are shared, most of them are not functioning. We selected three working components for this experiment which are NED-tagme (for the entity recognition module), Diambiguation-Property-OKBAQA (for the relation mapping module) and Query Builder (for the query generation module). This QA pipeline is evaluated locally over DBpedia.
- QAasparql [30]: QAasparql uses machine learning to classify the question into one of three types: list, count or Boolean questions. After that, it detects entities and relations and builds the queries based on the question type. The generated queries are then ranked by a module based on Tree-LSTM [39] to get the final query that is used to retrieve the system answer. QAasparql is oped-sourced and is evaluated locally over DBpedia.
- AskNow [22]: AskNow first normalizes the NLQ into an intermediary canonical syntactic form, named Normalized Query Structure, and then renders it into SPARQL queries. This QA system is open-sourced and is evaluated locally over DBpedia.
- AskPlatypus [40]: AskPlatypus is a multilingual QA system over Wikidata that is mainly based on query templates and slot filling. It is evaluated remotely using questions targeting Wikidata from QALD-7 and QALD-8.

Benchmarks: The benchmarks included in CBench are shown in Table 1. For the fine-grained query analysis, we exclude the benchmarks that do not include SPARQL queries.

Evaluation Metrics: We evaluate the QA system using three metrics: Micro, macro, and global F-1 scores. First, we define precision, recall, and F-1 for an individual question \( q_i \). If we assume that the set of answers in the gold standard for \( q_i \) (correct answers) is \( A_qi \), and the set of answers from the QA system for this question is \( A_i \), then we define the precision as \( \mu_i = \frac{|G_qi \cap A_qi|}{|G_qi|} \), recall as \( \nu_i = \frac{|G_qi \cap A_qi|}{|A_qi|} \), and F-1 as \( F_qi = \frac{2 \mu_i \nu_i}{\mu_i + \nu_i} \). If there are \( n \) questions in the benchmark, then micro F-1 is calculated as follows:

\[
F_{\mu} = \frac{\sum_{i=1}^{n} |G_qi \cap A_qi|}{\sum_{i=1}^{n} |A_qi|}, \quad R_{\mu} = \frac{\sum_{i=1}^{n} |G_qi \cap A_qi|}{\sum_{i=1}^{n} |G_qi|}, \quad F_{\mu} = \frac{2 F_{\mu} R_{\mu}}{F_{\mu} + R_{\mu}}.
\]

The macro F-1 is defined by calculating the F-1 scores per question and averaging the values for all questions. Formally, \( F_2 = \frac{\sum_{i=1}^{n} F_qi}{n} \). The global F-1 score evaluates the overall quality of the QA system in terms of the question it can answer correctly. If we assume that the set of questions in a benchmark is \( Q \), the set of questions that are processed by the QA system (non-empty answers) is \( S \), and the set of questions that is answered correctly is \( C \), then we define the global precision as \( P_G = \frac{|C|}{|S|} \), global recall as \( R_G = \frac{|C|}{|Q|} \), and global F-1 as \( F_G = \frac{2 P_G R_G}{P_G + R_G} \). The latter definition of the global scores is strict because a question is considered to be answered correctly if and only if the answer is correct.
correctly only if \( F_{qi} = 1 \). Therefore, in the literature, the notion of what is considered a correctly answered question is relaxed, which is referred to as partially correct answered questions. A question is considered partially correctly answered if \( 0 < F_{qi} < 1 \). In this paper, we also study the effect of changing the \( F_{qi} \) value used to consider a question to be partially correctly answered on the reported \( F_G \).

### 7.1 Evaluation of QA Systems

Table 12 shows the six systems and their scores for the 3 metrics used in CBench with \( F_{qi} > 0 \) for considering a question to be considered partially correctly answered. The effect of changing the value of the \( F_q \) threshold will be discussed later in this section. gAnswer experiments on QALD-7 and QALD-8 are excluded since they include questions that target Wikidata, which is not supported by gAnswer. The table shows that although the QALD benchmarks come from the same source, the six systems have considerably large variations in their scores. We calculate the average and standard deviation for all scores. The standard deviation values range from 0.0 to 0.15, which are considered to be high for variations in F-1 scores. When adding the LC-QUAD scores to the results, we notice a slight increase in the standard deviation of the global and macro scores and a slight decrease in the macro score. We will address the differences between the F-1 scores in Section 7.2.

Even though Freebase is deprecated since 2015, several benchmarks still target evaluating question answering over it. The following experiment is inspired by the recent trend of migrating Freebase to Wikidata [34]. We evaluate WDAqua using the benchmarks that originally target Freebase, but over Wikidata. The challenge to do this experiment is to guarantee that a chosen question from any of the benchmarks can be answered using Wikidata. We randomly sample questions from all the benchmarks and manually examine if the question can be answered using Wikidata by identifying the entities in the questions and the answers in the benchmark in Wikidata and finding a path that connects them, which resembles how a query should be written to retrieve the answers. We continue sampling until we find 200 Wikidata-answerable questions for each benchmark (with the exception of ComplexQuestions, which includes only 150 questions). The results for this experiment are shown in Table 13. WDAqua performs considerably better for questions from WebQuestions and ComplexQuestions compared to the other benchmarks. This especially affects the standard deviation of \( F_G \) (0.12), which is significant for F-1 scores.

We also investigate the effects of using different thresholds for the \( F_{qi} \) for a question to be considered partially correct. For this analysis, we evaluate WDAqua and gAnswer over QALD-9 (the benchmark both systems perform best in). Figure 6 shows that the two systems are significantly affected by the change of the \( F_{qi} \) value used. Although the two systems start at nearly the same value for \( F_G \), where a question is considered partially correct if \( F_{qi} > 0 \), WDAqua is more sensitive to stricter thresholds. Overall, there is a drop in \( F_G \) of 0.12 in WDAqua, where as gAnswer’s score drops by 0.09. Both drops are significant for F-1 scores, in general. The drop in WDAqua’s score is higher than that of gAnswer because WDAqua produces less-quality answers for individual questions compared to gAnswer. Therefore, a more restrictive quality threshold results in such a drop. It is also noticeable that the first step (from \( F_{qi} \geq 0 \) to \( F_{qi} \geq 0.1 \)) has the most significant impact as \( F_G \) drops by 0.04 for both systems, which is considered significant given the slightly stricter threshold.

### 7.2 Insights on Evaluation Metrics and Benchmarks

CBench includes the three evaluation metrics that have been used in the literature: Micro F-1 (\( F_p \)), Macro F-1 (\( F_G \)), and Global F-1 (\( F_C \)). It is interesting to see that previous works on QA over KG usually use only a subset of the three metrics. It is also common to use only the Global F-1. Our evaluation suggests that using a subset of the metrics is misleading in evaluating QA systems. Table 12 shows that the choice of a metric changes the order of the superior system. For example, if we consider the Global F-1 scores and use any of the QALD benchmarks for evaluation, gAnswer is superior to WDAqua. However, if we consider the Micro F-1 scores, WDAqua is superior. This means that the quality of the answer of individual questions is better in WDAqua (better Micro F-1 scores). However, gAnswer is able to answer more questions correctly or partially correctly. Interestingly, the same observation applies for the same QA system in some cases. For example, when evaluating WDAqua using the benchmarks QALD-1 to QALD-5, the Micro F-1 score is larger than the Macro F-1 score. However, using QALD-6 to QALD-9, the Macro F-1 score is larger. This means that the questions WDAqua struggles with in QALD-6 to QALD-9 have long answers. This results in a worse hit on the micro scores than on the macro scores (average of scores of individual questions). The order also changes based on the benchmark used, suggesting that the choice of a set of benchmarks in evaluation can be misleading in evaluating QA systems. For example, if we consider the Global F-1 scores and use QALD-7 to evaluate QAsparql and AskNow, we notice that AskNow is superior. However, if we use QALD-8, a benchmark from the same authors but one year younger, QAsparql is superior. These variations pose questions on how the benchmarks are created. It is interesting to see that these variations do not exist only for the automatically-generated benchmarks, but also for benchmarks that are manually created. This leads us to conclude that more research effort needs to be put to better understand how to sufficiently represent a good balance between all the combinations of the properties that we proposed in our work to have good coverage of as many...
Table 12: Evaluation of QA Systems over benchmarks targeting DBpedia/Wikidata. Benchmarks annotated with ★ include questions that target Wikidata.

| Basis | WDAqua[20] | gAnswer[27, 59] (TM+DP+QB) | Qanary[35, 36] | QA sparql[30] | AskNow[22] | Ask Platypus[40] |
|-------|------------|-----------------------------|---------------|--------------|-----------|-----------------|
|       | $F_G$ | $F_\mu$ | $F_\Sigma$ | $F_G$ | $F_\mu$ | $F_\Sigma$ | $F_G$ | $F_\mu$ | $F_\Sigma$ | $F_G$ | $F_\mu$ | $F_\Sigma$ | $F_G$ | $F_\mu$ | $F_\Sigma$ |
| QALD-1 | 0.31 | 0.27 | 0.14 | 0.44 | 0.18 | 0.24 | 0.00 | 0.00 | 0.00 | 0.02 | $\approx$0.00 | 0.01 | 0.12 | $\approx$0.00 | 0.07 | - | - |
| QALD-2 | 0.32 | 0.17 | 0.16 | 0.41 | 0.08 | 0.21 | 0.00 | 0.00 | 0.00 | 0.03 | $\approx$0.00 | 0.01 | 0.14 | $\approx$0.00 | 0.10 | - | - |
| QALD-3 | 0.21 | 0.23 | 0.11 | 0.28 | 0.11 | 0.16 | 0.05 | $\approx$0.00 | 0.02 | 0.12 | 0.01 | 0.06 | 0.19 | $\approx$0.00 | 0.13 | - | - |
| QALD-4 | 0.21 | 0.17 | 0.12 | 0.30 | 0.13 | 0.16 | 0.03 | $\approx$0.00 | 0.01 | 0.16 | 0.02 | 0.08 | 0.13 | 0.05 | 0.08 | - | - |
| QALD-5 | 0.31 | 0.19 | 0.18 | 0.36 | 0.10 | 0.20 | 0.04 | $\approx$0.00 | 0.02 | 0.23 | 0.01 | 0.12 | 0.29 | 0.11 | 0.09 | - | - |
| QALD-6 | 0.36 | 0.15 | 0.24 | 0.39 | 0.09 | 0.25 | 0.05 | $\approx$0.00 | 0.02 | 0.29 | 0.01 | 0.17 | 0.30 | 0.09 | 0.09 | - | - |
| QALD-7★ | 0.39 | 0.19 | 0.29 | - | - | - | 0.07 | 0.02 | 0.06 | 0.30 | 0.14 | 0.17 | 0.37 | 0.14 | 0.15 | 0.15 | $\approx$0.00 | 0.08 |
| QALD-8★ | 0.43 | 0.17 | 0.33 | - | - | - | 0.09 | 0.01 | 0.04 | 0.46 | 0.12 | 0.30 | 0.33 | 0.10 | 0.13 | 0.11 | $\approx$0.00 | 0.06 |
| QALD-9 | 0.43 | 0.20 | 0.32 | 0.44 | 0.10 | 0.30 | 0.08 | $\approx$0.00 | 0.07 | 0.32 | 0.02 | 0.19 | 0.26 | 0.07 | 0.08 | - | - |

Mean | 0.33 | 0.19 | 0.21 | 0.36 | 0.12 | 0.20 | 0.05 | $\approx$0.00 | 0.03 | 0.21 | 0.04 | 0.12 | 0.24 | 0.06 | 0.10 | 0.13 | $\approx$0.00 | 0.07 |

Std | 0.08 | 0.04 | 0.09 | 0.06 | 0.04 | 0.04 | 0.03 | $\approx$0.00 | 0.03 | 0.15 | 0.05 | 0.09 | 0.09 | 0.05 | 0.03 | 0.03 | $\approx$0.00 | 0.01 |

LC-QuAD | 0.20 | 0.03 | 0.15 | - | - | - | 0.02 | 0.01 | 0.01 | 0.46 | 0.14 | 0.34 | 0.16 | 0.01 | 0.11 | - | - |

Mean | 0.32 | 0.18 | 0.20 | 0.36 | 0.12 | 0.20 | 0.04 | 0.01 | 0.03 | 0.24 | 0.05 | 0.15 | 0.23 | 0.06 | 0.10 | 0.13 | $\approx$0.00 | 0.07 |

Std | 0.09 | 0.06 | 0.09 | 0.06 | 0.04 | 0.04 | 0.03 | 0.01 | 0.02 | 0.16 | 0.06 | 0.11 | 0.09 | 0.05 | 0.03 | 0.03 | $\approx$0.00 | 0.01 |

Table 13: Evaluation of QA Systems over benchmarks targeting Freebase. ComplexQuestion is annotated with ★ because it includes 150 questions only.

| Benchmark | WDAqua [20] (200 Questions) |
|-----------|-------------------------------|
|           | $F_G$ | $F_\mu$ | $F_\Sigma$ |
| WebQuestions | 0.22 | 0.12 | 0.12 |
| TempQuestions | 0.13 | 0.05 | 0.06 |
| ComplexQuestions★ | 0.32 | 0.11 | 0.19 |
| Mean | 0.17 | 0.06 | 0.09 |
| Std | 0.12 | 0.05 | 0.07 |

Figure 7: The properties of the Chain-shaped queries of the questions from QALD-9 answered by WDAqua. $T$ represents the number of triple patterns.

7.3 Fine-Grained Evaluation

In light of the results in Section 7.1, we analyze the queries of the questions processed by the QA systems. Due to the lack of space, the experiments in this section focus on WDAqua, over Chain-shaped and Star-shaped queries of QALD-9, using a threshold $F_q > 0.8$ to consider a question to be partially correct. We highlight the effectiveness of using CBench’s Debugging Mode to obtain the fine-grained results for these specific category of queries. Other settings (shapes, benchmarks, thresholds, and gAnswer) show similar results. It is also worth noting that these specific categories can be navigated to through the interactive report shown to the user if the user decides to evaluate the QA system using all the questions in a benchmark since we do a two-level grouping of all questions processed by the QA system when presenting them to the user. The first level of grouping is by shape, and the second level is by combinations of query properties.

Figures 7 and 8 show the fine-grained analysis for the aforementioned categories (Chain-shaped and Star-shaped queries, respectively). For each shape, each combination of properties shows how many questions were answered correctly or incorrectly by the QA system. Figure 7 shows that there are 17 different combination of properties for all the Chain queries in QALD-9. Most of the Chain cases as possible that a QA system could face in practice. Until such research is done, we recommend taking a comprehensive approach in evaluating QA systems by considering all evaluation metrics over the benchmarks that we recommended using in Sections 5.3 and 6.2. Such a comprehensive approach is now facilitated by CBench.
queries are either conjunctive queries that use the keywords select and distinct and have one triple pattern (56 questions are correctly answered and 97 are incorrectly answered), or conjunctive queries that use the same two keywords and have two triple patterns (28 questions are correctly answered and 49 are incorrectly answered). The user can choose any of the categories in the figure to examine individual questions, in which case, they can use the natural language analysis features of CBench (will be qualitatively evaluated in Section 7.4). It is worth noting that the Chain shape subsumes the Single-Edge shape, which is actually represented in this figure through the top five combinations of properties (i.e., $T = 1$).

Figure 8 shows the Star queries of QALD-9. These are considered to be more complex queries, with at least three triple patterns. The figure shows that WDAqua answered only 8 out of 38 questions whose query shape is Star. It is also worth noting that the union of the results in the two figures is the expected results for the Tree-shaped queries. The user can also navigate through the figures representing different shapes through the subsumption hierarchy of shapes.

**Debugging the QA System Using Fine-Grained Analysis:**

The main three components in any QA system are (1) Entity Recognizer, (2) Relation Mapper and the (3) Query Builder. For example, given the question *What is the capital of Canada?* discussed in Table 15, a good QA system’s Entity Recognizer would recognize the token *Canada* as the corresponding resource (i.e. dbo:capital in DBpedia). The Relation Mapper maps the sequence of tokens is the capital of to the corresponding predicate (i.e. dbo:capital in DBpedia). Lastly, the Query Builder produces the following formal query that returns the answers from the targeted KG: SELECT `?x { dbo:capital ?x}. These modules can be either dependent/independent on/of each other. The Query Builder mainly depends on a predefined/learned set of query templates as well as the entities and relations detected by the Entity Recognizer and the Relation Mapper. CBench can help the QA developers to learn more about the query templates that the QA does not handle using its fine-grained analysis. For example, Figure 7 shows that WDAqua fails to answer all the queries with properties like (CQ, select-distinct-limit-offset-and, $T=2$) and (CQ$_F$, select-distinct-filter-and, $T=2$). This indicates that the query builder of WDAqua may not be generating the query templates: SELECT DISTINCT `?x{ ... } OFFSET m LIMIT n and SELECT DISTINCT ?x{ ... FILTER(...)}. It is possible, however, that there is an issue with the Entity Recognizer and Relation Mapper for all the questions that were incorrectly answered. A more obvious example is when the fine-grained analysis shows that some questions whose queries have specific properties were answered correctly and some were not. The user can choose any of these questions for further investigations (discussed next).

### Table 14: Closest 5 question to the incorrectly answered question "What currency should I take to Dubai?", based on UPOS and Penn Treebank tag. The marks ✓/✗ denote correctly/incorrectly answered questions.

| Linguistically Closest Questions - UPOS |
|----------------------------------------|
| **Answer** | **Question** |
| ✓ | What currency should I take to Mauritius? |
| ✓ | What currency should I take to Jamaica? |
| ✓ | What currency should I take to Morocco? |
| ✓ | ✗ | What currency should I take to Mexico? |
| ✗ | ✗ | What kind of money should I take to Jamaica? |
| | | |

### Table 15: Closest 5 question to the incorrectly answered question "What is the capital of Canada?", based on UPOS and Penn Treebank tag. The marks ✓/✗ denote correctly/incorrectly answered questions.

| Linguistically Closest Questions - UPOS |
|----------------------------------------|
| **Answer** | **Question** |
| ✗ | ✗ | What is the title of Kakae? |
| ✓ | ✗ | What is the currency of Rhodesia? |
| ✓ | ✗ | What is the capital of Cameroon? |
| ✓ | ✗ | What is the capital of Venezuela? |
| ✗ | ✗ | What is the origin of Xynisteri? |
| | | |
7.4 Qualitative Evaluation of Linguistically-Similar Questions

In this section, we assume that the user navigates through the fine-grained analysis figures and investigates a question of interest that is not correctly answered by the QA system. For benchmarks that do not include SPARQL queries, the user can directly choose any question from the benchmark. CBench finds the $k$ linguistically closest questions to this chosen question based on the approach discussed in Section 6. This feature is especially beneficial in debugging the Entity Recognizer and the Relation Mapper components in the QA system. In this experiment, we use gAnswer as the QA system. Other configurations do not reflect major changes to the top-$5$ questions. Changing the value of the thresholds for $F_{q_i}$ to consider a question to be partially correctly answered results in possible changes in the label of the correctness of the question.

Table 14 shows the linguistically-closest $5$ questions to the incorrectly answered question “What currency should I take to Dubai?”. There is considerable overlap between the two tag dictionaries (4 out of 5 questions match). In both cases, we note that almost all of the similar questions are correctly answered, unlike the question of interest, possibly highlighting that the question is correctly answered if the entity is a country (one-hop connection to currency in the KG) rather than a city (two-hops connection to currency).

Table 15 shows a different interesting scenario for the incorrectly answered question “What is the capital of Cameroon?”, where for the top part (using UPOS), most of the similar questions are also incorrectly answered. We note that all the questions have the template “What is the <place_holder> of <place_holder>?”. This indicates that the QA system struggles with this type of questions with only few exceptions. However, the question “What is the capital of Cameroon?” is answered correctly. This indicates that the QA system either could not identify Canada as an entity, or capital of as the predicate dbo:capital. Indeed, investigating gAnswer reveals that the Relation Mapper (referred to in gAnswer as the Relation Recognition step) mistakenly mapped the relation mention dictionary to the sparsity of the relation in the question to the type predicate due to the sparsity of the KG-Related Issues: In our work, we encountered some interesting cases where the query in the benchmark is unnatural in the sense that it would have been written in a different way but for the representation of the answer in the KG. For example, the following query (from QALD-9) is associated with the question “What airlines are part of the SkyTeam alliance?”

```
SELECT DISTINCT ?uri WHERE {
  ?uri a dbo:Airline {
    ?uri dbo:alliance dbr:SkyTeam
  } UNION {
    ?uri dbo:Alliance dbr:SkyTeam
  }
}
```

Normally, the query should only include one graph pattern that consists of two triple patterns. However, due to the co-existence of the two predicates in the query above (dbo:alliance versus dbo:Alliance), the query in the benchmark had to be written in that way. This issue in DBpedia was later corrected and the predicate dbo:Alliance was removed.

8 DISCUSSION

In light of our work, we would like to highlight the following observations that we believe should be taken into consideration for future research work in the area of QA over KGs:

- Variations Among Benchmarks: Although all benchmarks target the same problem, there are high-degree variations with respect to several linguistic features in the natural language questions, and syntactical and structural features in the queries (Sections 5 and 6). These variations pose questions on how the benchmarks are created. It is interesting to see that these variations do not exist only for the automatically-generated benchmarks, but also for benchmarks that are manually created. More research effort needs to be put to better understand how to sufficiently represent a good balance between all the combinations of the features that we proposed in our work to have good coverage of as many cases as possible that a QA system could face in practice.
- Correctness of Benchmarks: We noticed that the automatically-generated benchmarks may include some erroneous entries, which may affect the reported accuracy of the evaluated QA system. For example, it is interesting to notice that GraphQuestions include yes/no questions. However, the queries and answers do not reflect this type of questions by not using the SPARQL’s ASK keyword and having non-binary answers.
- Staleness: KGs are continuously evolving, while benchmarks are static by nature. Most of the KGs targeted by the benchmarks in CBench now have more recent versions than those that were used to generate/create the benchmark. In our work, we proposed a simple solution by updating the benchmark entries based on the queried KG. However, this approach will not work if the KG’s evolution touches its ontology or vocabulary. In fact, we noticed such behaviour in our benchmarks. Some queries no longer produce any results because of the known DBpedia changes. It is interesting to also notice that some other queries in earlier QALD benchmarks are no longer syntactically correct due to changes in the standard query language (SPARQL). This observation is especially concerning for benchmarks that do not include SPARQL queries, where it is not even possible to automatically update the answers in the benchmark and will require daunting human efforts to make the benchmark useful.
- KG-Related Issues: In our work, we encountered some interesting cases where the query in the benchmark is unnatural in the sense that it would have been written in a different way but for the representation of the answer in the KG. For example, the following query (from QALD-9) is associated with the question “What airlines are part of the SkyTeam alliance?”

```
SELECT DISTINCT ?uri WHERE {
  ?uri a dbo:Airline {
    ?uri dbo:alliance dbr:SkyTeam
  } UNION {
    ?uri dbo:Alliance dbr:SkyTeam
  }
}
```

Normally, the query should only include one graph pattern that consists of two triple patterns. However, due to the co-existence of the two predicates in the query above (dbo:alliance versus dbo:Alliance), the query in the benchmark had to be written in that way. This issue in DBpedia was later corrected and the predicate dbo:Alliance was removed.

9 RELATED WORK

Retrieving data from databases using non-structured languages has been a topic of interest since the emergence of databases [24]. More recent works focused more on information retrieval (IR)-based QA systems, which were designed to handle free text. Then for the last two decades, many QA over KGs systems emerged due to the increasing amount of new structured data on the web that is in the form of KGs [25]. Moreover, hybrid QA systems that retrieve data from both free text and KGs have also emerged (e.g. [8, 48]).

Due to the popularity of QA over KGs, several evaluation datasets emerged [3, 4, 7, 9, 12, 13, 15, 16, 28, 37, 41, 43–47, 49, 50] targeting multiple popular KGs like DBpedia [6], Freebase [10], Wikidata [53],
We experimentally show that these variations affect the reported question answering system. Based on multiple features, this analysis can be used by the user to analyze the processed questions and their corresponding queries traditionally known as quality scores, but it also gives a fine-grained analysis provided by CBench helps the user identify strong and weak points of their QA system rather than just viewing single-numbered scores.

10 CONCLUSION

In this paper, we highlight the high-degree variations in the benchmarks used to evaluate question answering over knowledge graphs. We experimentally show that these variations affect the reported scores for multiple question answering systems. To overcome the effects of such variations, we introduce CBench, a fine-grained benchmark suite that comes with a prepacked set of popular benchmarks that target multiple popular knowledge graphs. CBench is easy to use and extensible. It does not only provide the user with the traditionally known quality scores, but it also gives a fine-grained analysis of the processed questions and their corresponding queries based on multiple features. This analysis can be used by the user to quickly identify the strong and weak points of their evaluated question answering system.

REFERENCES

[1] SPARQL 1.1 query language. http://www.w3.org/TR/sparql11-query/, 2013.
[2] RDF 1.1 concepts and abstract syntax. http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/, 2014.
[3] A. Abujabal, R. Saha Roy, M. Yahya, and G. Weikum. ComQA: A community-sourced dataset for complex factoid question answering with paraphrase clusters. In Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HL), 2019.
[4] A. Abujabal, M. Yahya, M. Riedewald, and G. Weikum. Automated template generation for question answering over knowledge graphs. In Proceedings of the International Conference on World Wide Web (WWW), 2017.
[5] A. Akbik, D. Blythe, and R. Vollgraf. Contextual string embeddings for sequence labeling. In Proceedings of the International Conference on Computational Linguistics, 2018.
[6] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. DBpedia: A nucleus for a web of open data. In Proceedings of the International Semantic Web Conference (ISWC) 2007.
[7] M. Azmy, P. Shi, J. Lin, and I. Ilyas. Farewell freebase: Migrating the simple-questions dataset to dbpedia. In Proceedings of the International Conference on Computational Linguistics, pages 2093–2103, 2018.
[8] R. Beaumont, B. Grau, and A.-L. Ligozat. SemGraphQA@ QALD5: LIMSI participation at QALDS@ CLEF. In CLEF Working Notes Papers, 2015.

[9] J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on Freebase from question-answer pairs. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), 2013.
[10] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 2008.
[11] A. Bosu, W. Martens, and T. Timm. An analytical study of large SPARQL query logs. Proceedings of the VLDB Endowment, 11(2), 2017.
[12] A. Bordes, N. Usunier, S. Chopra, and J. Weston. Large-scale simple question answering with memory networks. arXiv preprint arXiv:1506.02075, 2015.
[13] E. Cabrio, P. Cimiano, V. Lopez, and S. Walter. QA4L: Multilingual question answering over linked data. In CLEF Working Notes Papers, 2013.
[14] E. Cabrio, J. Coajan, A. P. Aprosio, B. Magmani, A. Lavelli, and F. Gandon. QA4K: an open domain QA system based on relational patterns. In Proceedings of the International Semantic Web Conference (ISWC), 2012.
[15] E. Cabrio, J. Coajan, B. Magmani, F. Gandon, and A. Lavelli. QAK@ qald-2. 2012.
[16] Q. Cai and A. Yates. Large-scale semantic parsing via schema matching and lexicon extension. In Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL), 2011.
[17] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka, and T. M. Mitchell. Toward an architecture for never-ending language learning. In Proceedings of the AAAI conference on artificial intelligence, 2010.
[18] J. D. Choi, J. Tetrauld, and A. Stent. B-depends: Dependency parser comparison using a web-based evaluation tool. In Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL) and the International Joint Conference on Natural Language Processing (IJCNLP), 2015.
[19] W. Cui, Y. Xiao, H. Wang, S. Song, S. W. Hwang, and W. Wang. KBQA: Learning question answering over QA corpora and knowledge bases. Proceedings of the VLDB Endowment (PVLDB), 5(2017).
[20] D. Diefenbach, K. Singh, and P. Marek. Wdaqua-core40: A question answering component for the research community. In Semantic Web Evaluation Challenge, 2017.
[21] X. Dong and A. Halevy. Indexing datasets. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 2007.
[22] M. Dubey, S. Daigupta, A. Sharma, K. Hoffner, and J. Lehmann. Asknow: A framework for natural language query formalization in sparql. In European Semantic Web Conference (ESWC), 2016.
[23] A. Fader, L. Zettlemoyer, and O. Etzioni. Open question answering over curated and extracted knowledge bases. In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2014.
[24] B. F. Green Jr, A. K. Wolf, C. Chomsky, and K. Laughery. Baseball: an automatic question-answering system: In The western joint IRE-AIEE-ACM computer conference, 1961.
[25] K. Hoffner, S. Walter, E. Mars, R. Usbeck, J. Lehmann, and A.-C. Ngonga Ngomo. Survey on challenges of question answering in the semantic web. Semantic Web, 8(6), 2017.
[26] M. Honnibal and M. Johnson. An improved non-monotonic transition system for dependency parsing. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), 2015.
[27] S. Hu, L. Zou, J. X. Yu, H. Wang, and D. Zhao. Answering natural language questions by subgraph matching over knowledge graphs. IEEE Transactions on Knowledge and Data Engineering (TKDE), 39(5), 2017.
[28] Z. Jia, A. Abujabal, R. Saha Roy, J. Strötgen, and G. Weikum. TempQuestions: A benchmark for temporal question answering. In Proceedings of the International Conference on World Wide Web (WWW), 2018.
[29] F. Kaufmann, A. Bernstein, and R. Zumstein. Queris: A natural language interface to query ontologies based on clarification dialogs. In Proceedings of the International Semantic Web Conference (ISWC), 2006.
[30] S. Liang, K. Stockinger, T. M. de Fantas, M. Aninizova, and M. Gil. Querying knowledge graphs in natural language. Journal of Big Data, 8(1), 2021.
[31] V. Lopez, M. Fernández, E. Motta, and N. Stierer. PoweAqua: supporting users in querying and exploring the semantic web. Semantic Web, 3(3), 2011.
[32] X. Ma and E. Hovy. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL), 2016.
[33] C. Matuszek, M. Witbrock, J. Cabral, and J. DeOliveira. An introduction to the Multi-Grid dataset for complex factoid question answering with paraphrase clusters. In Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics, 2015.
[34] V. Lopez, M. Fernández, E. Motta, and N. Stierer. PoweAqua: supporting users in querying and exploring the semantic web. Semantic Web, 3(3), 2011.
[35] K. Singh, A. Both, A. Sethupat, and S. Shekarpour. Why reinvent the wheel: Let’s build question answering systems together. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), 2013.
[36] A. Khikmatullaev, D. Punjani, C. Lange, et al. Why reinvent the wheel: Let’s build question answering systems together. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), 2013.
[37] Y. Su, H. Sun, B. Sadler, M. Srivatsa, I. Guz, Z. Yan, and X. Yan. On generating characteristic-rich question sets for QA evaluation. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), 2016.

[38] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In Proceedings of the International World Wide Web Conference (WWW), 2007.

[39] K. S. Tai, R. Socher, and C. D. Manning. Improved semantic representations from tree-structured long short-term memory networks. arXiv preprint arXiv:1503.00075, 2015.

[40] T. P. Tanon, M. Dias de Assuncao, E. Caron, and F. Suchanek. Platypus – A Multilingual Question Answering Platform for Wikidata. Technical report, 2018.

[41] P. Trivedi, G. Maheshwari, M. Dubey, and J. Lehmann. Lc-quad: A corpus for complex question answering over knowledge graphs. In International Semantic Web Conference (ISWC), 2017.

[42] C. Unger, L. Bühmann, J. Lehmann, A.-C. Ngonga Ngomo, D. Gerber, and P. Cimiano. Template-based question answering over RDF data. In Proceedings of the International World Wide Web Conference (WWW), 2012.

[43] C. Unger, P. Cimiano, V. Lopez, and E. Motta. Question answering over linked data (QALD-1). In Workshop on Question Answering Over Linked Data (QALD-1), 2011.

[44] C. Unger, C. Forascu, V. Lopez, A.-C. N. Ngomo, E. Cabrio, P. Cimiano, and S. Walter. Question answering over linked data (QALD-4). In CLEF Working Notes Papers, 2014.

[45] C. Unger, C. Forascu, V. Lopez, A.-C. N. Ngomo, E. Cabrio, P. Cimiano, and S. Walter. Question answering over linked data (QALD-5). In CLEF Working Notes Papers, 2015.

[46] C. Unger, A.-C. N. Ngomo, and E. Cabrio. 6th open challenge on question answering over linked data (qald-6). In CLEF Working Notes Papers, 2016.

[47] R. Usbeck, A.-C. N. Ngomo, F. Conrads, M. Röder, and G. Napolitano. 8th challenge on question answering over linked data (qald-8). Joint Proceedings of the International Workshop on Benchmarking Linked Data and Natural Language Interfaces for the Web of Data (NLIWoD), 2018.

[48] R. Usbeck, A.-C. N. Ngomo, B. Haarmann, A. Kritthara, M. Röder, and G. Napolitano. 7th open challenge on question answering over linked data (qald-7). In Semantic Web Challenges, 2017.

[49] R. Usbeck, M. Röder, M. Hoffmann, F. Conrads, J. Huthmann, A.-C. N. Ngonga Ngomo, C. Demmler, and C. Unger. Benchmarking question answering systems. Semantic Web, 10(2), 2019.

[50] R. Usbeck, M. Röder, and A.-C. N. Ngonga. Evaluating entity annotators using gerbil. In European Semantic Web Conference (ESWC), 2015.

[51] D. Vrandečić and M. Krötzsch. Wikidata: a free collaborative knowledgebase. Communications of the ACM, 57(10), 2014.

[52] G. Zenz, X. Zhou, E. Minack, W. Siberski, and W. Nejdl. From keywords to semantic queries: Incremental query construction on the semantic web. Journal of Web Semantics, 7(3), 2009.

[53] W. Zheng, J. X. Yu, L. Zou, and H. Cheng. Question answering over knowledge graphs: question understanding via template decomposition. Proceedings of the VLDB Endowment (PVLDB), (11), 2018.

[54] Q. Zhou, C. Wang, M. Xiong, H. Wang, and Y. Yu. Spark: Adapting keyword query to semantic search. In Proceedings of the International Semantic Web Conference (ISWC), 2007.

[55] L. Zou, R. Huang, H. Wang, J. X. Yu, W. He, and D. Zhao. Natural language question answering over RDF: a graph data driven approach. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 2014.