A comparative study of question answering over knowledge bases

Khiem Vinh Tran[a,b], Thanh Tam Nguyen[c], Hao Phu Phan[d], Ngan Luu-Thuy Nguyen[a,b], Jun Jo[e] and Quoc Viet Hung Nguyen[e]

[a] *University of Information Technology, Ho Chi Minh City, Vietnam*
[b] *Vietnam National University, Ho Chi Minh City, Vietnam*
[c] *Leibniz Universiät, Hannover, Germany*
[d] *National Research University Higher School of Economics, Russia*
[e] *Griffith University, Australia*

ARTICLE INFO

Keywords:
Question answering
Knowledge base
Knowledge graph
Query processing

ABSTRACT

Knowledge bases are popular sources of information, but non-expert users may find it difficult using structured queries to access them. Question answering over knowledge bases (KBQA) has become a popular approach to overcome these difficulties. However, most KBQA systems still confront significant difficulties in converting natural languages (NL) to query languages (e.g. SPARQL). Moreover, they often answer simple questions just for one triple, but complicated inquiries require complex questions with sub questions or multiple functions. As a result, it is difficult to choose a KBQA system for a particular scenario. This article provides a thorough examination of KBQA systems and challenges them with a comprehensive setting of question type, query language, and knowledge domain. Besides evaluating a KBQA system using a sizable collection of evaluation metrics, this study also provides a detailed analysis of the linguistic properties of both question and answer in order to better assist the developers of question answering systems in better understanding where their system excels and where it struggles. Especially, we also develop a new multilingual corpus about COVID-19 for future KBQA research.

1. Introduction

Question Answering (QA) is a long-standing discipline within the field of natural language processing (NLP), which is concerned with providing answers to questions posed in natural language on data sources, and also draws on techniques from linguistics, database processing, and information retrieval. While the first research efforts on QA were carried out in the 1960s, it has still received a great deal of interest during the past few years. The major reasons for this include substantial advancements in voice recognition and NLP, as well as the widespread availability of knowledge bases (KB) in a wide range of disciplines (Tanon et al., 2020; Rebele et al., 2016; Zheng et al., 2021).

According to (Weikum, 2021), over the last 15 years, knowledge bases, also known as knowledge graphs, have been automatically constructed from web data, and have become a key asset for search engines and other use cases. The number of knowledge graphs (KGs) has increased at an unprecedented rate in recent years (Auer et al., 2007; Bollacker et al., 2008; Vrandečić and Krötzsch, 2014). These KGs include a wealth of information that may possibly be utilized for QA. Finding answers for a question in a KG, on the other hand, is not always straightforward. The user needs to have a thorough understanding of the KG as well as a structured query language in order to express their queries in a structured manner that can be utilized to locate matches in the KG. These criteria restrict the ability to ask questions to expert users who possess the required skills to construct queries that are syntactically and semantically valid and that properly reflect their information requirements. The number of users who fall into this category is a small percentage of a potentially huge user base. In order to address this issue, a significant number of QA systems that allow users to express their information requirements using natural language have been created. According to (Jurafsky and Martin, 2019), factoid question answering has two main approaches - information retrieval (IR) based QA and knowledge-based QA. In the first approach, many datasets and systems have been established recent years with the advancement of machine reading comprehension (MRC) task. One of the most famous datasets is SQuAD (Rajpurkar

17520634@gm.uit.edu.vn (K.V. Tran); thanhtamlhp@gmail.com (T.T. Nguyen); ffan@edu.hse.ru (H.P. Phan); nganlt@uit.edu.vn (N.L. Nguyen); j.jo@griffith.edu.au (J. Jo); quocviethung.nguyen@griffith.edu.au (Q.V.H. Nguyen)
ORCID(s): 0000-0001-7128-2058 (K.V. Tran); 0000-0002-2586-7757 (T.T. Nguyen); 0000-0003-3803-3396 (H.P. Phan); 0000-0002-9687-1315 (Q.V.H. Nguyen)

Table 1
Comparison between existing benchmarks on QA over KBs

Benchmarks	Our	(Orogat et al., 2021)	(Costa and Kulkarni, 2018)
New data	✓	×	×
Multi-domain	✓	×	×
Cross-Query Language	✓	×	×
Multi-language	✓	✓	×
Code	✓	✓	×
Large KBs	✓	✓	✓

et al., 2016) conducted in English. MRC is also become famous in multilingualism such as Chinese (He et al., 2018), Arabic (Romeo et al., 2019), and Vietnamese (Nguyen et al., 2020). The second approach is question answering over knowledge base (KBQA) with precision of question answering over knowledge graph (KG). Many studies have been conducted with KB such as (Shin et al., 2019; Li and Madden, 2019; Qiao and Hu, 2020) and recently is WabiQA (Noraset et al., 2021), which is KBQA system in Thai language.

As a consequence of the important of QA over KBs, many benchmarks have been developed recently. These benchmarks usually contain questions expressed in plain language, responses to the questions from the KB targeted by the benchmark, and potentially structured queries that retrieve the answers already indicated. To assess a freshly created QA system, its developers must select datasets with developed in a large KB in order to evaluate their system. Most of benchmarks choose DBPedia as a KB for evaluation because many QA datasets are built on top of this KB. However, they are restricted to this specific set of KBs and most benchmarks cover only a small number of KBs. Additionally, the number of questions in each of the existing benchmarks is substantially different, rendering it difficult for users to select a practice guideline. For example, one benchmark has a modest number of questions (100 questions) (Siciliani et al.) while another has 5,000 questions (Trivedi et al., 2017).

Another issue with existing benchmarks is their lack of diversity in terms of query languages. Today, several benchmarks make use of SPARQL-only datasets. In order to find answers in the desired KB, the user first parses the question, retrieves the SPARQL query, and then utilizes the QA system. In order to generate various evaluation scores such as micro, macro, and global F-1 scores, the provided answers are compared to the answers retrieved from the benchmark file. When the QA system does not properly answer all of the questions, the user goes back through the questions and debugs the code to figure out why the QA system is having trouble with certain questions. If we have any datasets that are queryable using SQL or another query language, we may compare the efficacy of each query language.

An additional factor to consider is the domain of datasets; with current datasets, numerous benchmarks are not domain specific but deal with datasets in a number of domains. The problem of COVID-19 has resulted in the development of biomedical datasets with the goal of assisting the world in overcoming this horrific epidemic. As a result, it is urgent that quality assurance over KB in the biomedical domain should be addressed in benchmarks. In order to address this issue, this study analyses five KBQA systems with both biomedical and nonspecific data.

In this article, we provide the results of a benchmarking analysis that was conducted to assess QA over KG. The performance of a broad variety of state-of-the-art QA methods, baselines, and of datasets with cross-domain characteristics, is evaluated using this framework. This framework is also used to conduct in-depth studies on synthetic datasets to derive insights into the performance behaviour of the benchmarked techniques, and in-depth analyses on many existing datasets to foresee the future of QA over KGs. On the basis of these findings, we offer a series of recommendations for the selection of suitable QA methods for certain application scenarios.

At the expense of conventional comparative studies, our research incorporates novel empirical comparisons between three baselines that are considered to be state-of-the-art. Our benchmark is distinct from previous benchmarks in this field, as shown by the discrepancies in Table 1. The contributions of our benchmark framework can be summarized as follows:

- To illustrate the consequences of such differences, we analyze five KBQA systems with both biomedical and generic data in the QA evaluation and demonstrate that the quality ratings of the systems differ considerably.
- We provide valuable insights for academics working on QA systems, such as benchmark selection and QA assessment measures. To the best of our understanding, we are the first to encompass ideas of data analysis and knowledge bases (KBs) in QA benchmarks.

- We examine and compare the performance of the most successful KQBA systems on existing datasets. Afterwards, we present COVID-KGQA, a new knowledge graph question answering corpus that includes the most questions about COVID-19. In addition, we also evaluate the empirical study of these systems and datasets with the effects of differences in terms of a variety of linguistic aspects of the natural language questions in the benchmarks.
- We provide valuable performance guidelines to academics working on KBQA in terms of benchmarks and QA systems selection. Alternatively, we suggest some future attempts that may be useful in conjunction with the improvement of the studies.
- In order to facilitate study, we have made the source code and dataset (which we generated) accessible. Our findings are reliable and reproducible, and the source code is publicly available [1].

The remainder of our paper is organized as follows. In §2 lists background and literature review on question answering over knowledge base. Next, in §3 we formulate the problem and introduce key concept of KBQA. Then, we discuss in §4 different representative techniques used in this benchmark. §5 introduces the setup used for our benchmark, including the component-based design, datasets, metrics, and evaluation procedures. §6 reports the experimental results. A summary of the findings, practical guidelines and future directions are provided in §7. Finally, §8 summarizes our works and concludes the paper.

2. Background

In order to build an question-answering over knowledge base (KBQA) system, it is necessary to integrate methods from the fields of semantic web (SW), machine learning (ML) and natural language processing (NLP) . In this part, we briefly cover the concepts related to KBQA.

Semantic Web (SW). SW is part of World Wide Web consisting of a set of standards, proposed by the World Wide Web Consortium (W3C), that are used to store and query KB. As (Berners-Lee et al., 2001), Semantic Web is not a distinct web from the existing web, but rather an extension of the current web. One of the most significant features of the Semantic Web is the ability for computers to analyze data and information on their own without human intervention. In order to assist in realizing the full potential of the present Web, a number of important technologies, primarily the Resource Description Framework (RDF) (Cyganiak et al., 2014) and the Web Ontology Language (OWL) (Antoniou and Van Harmelen, 2004), have been introduced to the community. These techniques have enabled a growing number of papers containing uniformly arranged data to be published on the Web in a simple manner. Meanwhile, SPARQL (Prudhommeaux, 2008) is a query language that may be used to search for and manipulate RDF-formatted data that has been collected from a variety of sources. In addition, the development of standardized technologies for the Semantic Web has facilitated the incorporation of semantics into existing texts as well as the connecting of common data across different application areas and even geographical regions. This allows for the creation of a web of data in which the data is connected using typed connections, which is known as Linked Data (Bizer et al., 2008). Because it makes typed assertions, it can link arbitrary entities all across the globe, and it permits complex queries to be submitted in SPARQL. The detail of RDF, OWL, SPARQL and Linked data are give below.

Resource Description Framework (RDF). RDF is a representation language for describing information on the World Wide Web. The World Wide Web Consortium (W3C) published the initial RDF specification in 1997 (Lassila and Swick, 1997), and it later became a W3C recommendation in 1999 (Brickley et al., 1999). Currently, the most recent version is RDF 1.1, which was released in 2014 (Cyganiak et al., 2014) by W3C.

SPARQL. SPAQRL Protocol and RDF Query Language (SPARQL) is a structured language (Prudhommeaux, 2008), similar to SQL, that is used for accessing and manipulating RDF graphs. The current version of SPARQL deployed by W3C is SPARQL 1.1 (Harris et al., 2013)

Web Ontology Language (OWL). OWL (McGuinness et al., 2004) is a Semantic Web language meant to convey rich and sophisticated knowledge about things, groups of things, and relationships between things. As a computational logic-based language, OWL makes it possible for computer programs to make use of knowledge expressed in OWL. In 2009, the W3C OWL Working Group produced the current version of OWL, which is commonly referred to as OWL 2 (Hitzler et al., 2009). A Second Edition of OWL was published in 2012 (Consortium et al., 2012).

[1] https://github.com/tamlhp/kbqa

Table 2
Example of Question Answering over Knowledge Graphs

Answer Type	Sample Question	Query	Answer
Union of Power	What is the nick name of Baghdad? (QALD-9)	PREFIX dbr: http://dbpedia.org/resource/ PREFIX foaf: http://xmlns.com/foaf/0.1/ SELECT ?nm WHERE {dbr:Baghdad foaf:nick ?nm}	The City of Peace
	Which company which assembles its cars in Broadmeadows, Victoria? (LC-QUAD)	SELECT DISTINCT ?uri WHERE { x http://dbpedia.org/property/assembly http://dbpedia.org/resource/Broadmeadows,_Victoria . ?x http://dbpedia.org/ontology/parentCompany ?uri . ?x http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://dbpedia.org/ontology/Automobile}	http://dbpedia.org/resource/Ford_Motor_Company
Number	How many awards has Bertrand Russell? (QALD-9)	PREFIX dbr: http://dbpedia.org/resource/ PREFIX dbp: http://dbpedia.org/property/ SELECT (COUNT(?Awards) AS ?Counter) WHERE {dbr:Bertrand_Russell dbp:awards ?Awards}	5
	How many shows does HBO have? (LC-QUAD)	SELECT DISTINCT COUNT(?uri) WHERE {ri http://dbpedia.org/ontology/company http://dbpedia.org/resource/HBO . ?uri http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://dbpedia.org/ontology/TelevisionShow}	38
Boolean	Was Margaret Thatcher a chemist? (QALD-9)	ASK WHERE {ttp://dbpedia.org/resource/Margaret_Thatcher http://dbpedia.org/ontology/profession http://dbpedia.org/resource/Chemist}	true
	Was Tim Gunn a guest on The Broken Code? (LC-QUAD)	ASK WHERE {ttp://dbpedia.org/resource/The_Broken_Code http://dbpedia.org/property/guests http://dbpedia.org/resource/Tim_Gunn}	true

[1] RDF Syntax: https://www.w3.org/1999/02/22-rdf-syntax-ns

Linked Data. Linked data (Bizer et al., 2011) refers to a collection of best practices for publishing, sharing, and connecting data, information, and knowledge on the Web. One of the primary goals of linked data is to establish relationships between data, which can then be used to construct a Web of Data. RDF as a standard data format lays a solid foundation for information unification on the conventional Web. DBpedia (Auer et al., 2007) is a typical large-scale Linked Dataset on the web which, essentially, makes the content of Wikipedia available in RDF. The knowledge graph on DBpedia not only connects the concepts represented inside Wikipedia, but also with other external information available on the Web.

3. Problem Statement

Question Answering over Knowledge Bases (KBQA) is the term used for the task of retrieving the answer from executing query matching with natural language question over a knowledge base. Formally, we can define the task of KBQA as follow. Let KB be a knowledge base, Q is a question, q is the matching query and A is an answer extracted by matching query q executed from given question Q over KB. According to (Chakraborty et al., 2021), the set of all possible answers can be in the form as follows. The first is the union of the power set P of entities E and literals L in KB: $P(E \cup L)$. The second is the number of outcomes set for all potential functions of aggregation $F : P(E \cup L) \mapsto \mathbb{R}$. The third is a Boolean set of answer for yes/no questions. Figure 1 gives the illustration of an illustration of the question answering over knowledge bases process and Table 2 gives some examples.

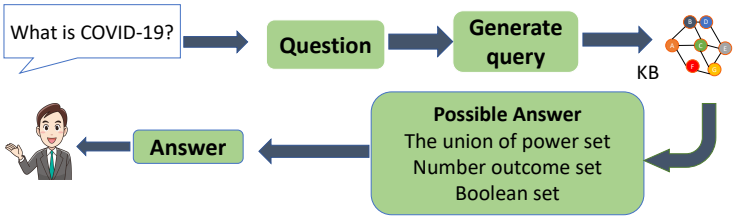


Figure 1: Question answering over knowledge bases.

In other words, KBQA involves determining if a natural language question can be addressed correctly and concisely in knowledge bases. The fundamental goal of KBQA is to comprehend the true semantics of a natural language query and extract it in order to match it with the whole semantics of a knowledge base. However, owing to the varied semantics of natural language queries in the actual world, this is precisely a significant problem.

4. Representative methods for question answering over knowledge bases

In recent years, there have been an increasing number of off-the-shelf methods to KBQA in a variety of applications. It becomes fascinating to compare and evaluate them in order to help users make informed decisions. KBQA can be

divided into 4 main approaches: (i) *embedding-based* – converts a query into a logic form, which is then executed against KBs in order to discover the appropriate responses; (ii) *subgraph matching* – constructs the query subgraph using a semantic tree, (iii) *template-based* – converts user utterances into structured questions via the use of semantic parsing, and (iv) *context-based* – comprehends the questions with different contexts.

4.1. Embedding techniques

Embedding techniques take advantages of semantic and syntactic information included in a question as well as a database schema to produce a SQL logic form (parsing tree) A sub-task of semantic parsing attempts to translate a natural language text into a formal semantic representation, such as SQL queries, logic forms, and code creation. A popular method of generating SQL from questions in the literature is to use a SQL structure-based sketch with many slots and frame the issue as a slot filling task by including some kind of pointing/copying mechanism. Question-to-SQL is implemented using a variety of methods (Finegan-Dollak et al., 2018; Yu et al., 2018)

TREQS. The underlying concept of this model is to convert healthcare-related questions posed by physicians into database queries, which are then used to obtain the response from patient medical records. Given that questions may be linked to a single table or to many tables, and that keywords in the questions may not be correct owing to the fact that the questions are written in healthcare language, using the language generation method (Zhang et al., 2019), this model is able to address the problems of general-purpose applications. Using a language generation model, this translate-edit model produces a query draft, which is subsequently edited in accordance with the table schema. Three components in the Translate-Edit Model for Question-to-SQL (TREQS (Wang et al., 2020)) generation are listed below:

- *Sequence-to-Sequence Framework*: For Question-to-SQL generation task, as part of the TREQS framework, a question encoder (single-layer bidirectional LSTM) is combined with an SQL decoder. The input of encoder is word embeddings of input tokens. Then they are converted to a sequence of encoder hidden state $h^e = (h^e_1, h^e_2, ...h^e_J)$ with e indicating that the hidden states are obtained from the encoder and the concatenation of the hidden states of forward and backward LSTM is $h^e_j = \overrightarrow{h^e_J} \oplus \overleftarrow{h}^e_{J-j+1}$. During each decoding step t, the decoder uses the encoder hidden states and the word embedding of the preceding token as input and produces a decoder hidden state h^t_d. The encoder is equipped with word embeddings and decoder from the same W_{emb} matrix. The hidden and cell states of the decoder LSTM are initialized using.

$$h^d_0 = tanh\Big(W_{e2dh}(\overrightarrow{h^e_J} \oplus \overleftarrow{h^e_1}) + b_{e2dh}\Big) \tag{1}$$

$$c^d_0 = tanh\Big(W_{e2dc}(\overrightarrow{h^e_J} \oplus \overleftarrow{c^e_1}) + c_{e2dh}\Big) \tag{2}$$

  W_{e2dh}, W_{e2dc} are the weighted matrices, b_{e2dh}, b_{e2dc} are vectors and both are learnable parameters.

- *Temporal Attention on Question:* At every phase of decoding t the decoder does not only accept its hidden inner state and token as input but also concentrates on the important elements of the problem for the current generation.

  The temporal attetion strategy (Nallapati et al., 2016) demonstrated the effectiveness of solving the problem as described above and can better prevent repeated attendance on the same section of the question than the standard attention models (Luong et al., 2015; Bahdanau et al., 2014). To accomplish this purpose, the model first creates a function to align the current hidden state of the decoder with each hidden state of the encoder:

$$s^e_{tj} = (h^e_j)^\intercal W_{align} h^d_t \tag{3}$$

  with W_{align} are parameter. With the aim to avoid repetitive attention, in the preceding decoding steps, the model penalize the tokens that have acquired high attention scores with the following normalization rule:

$$s^{temp}_{tj} = \begin{cases} exp(s^e_{tj}), \text{if } t = 1 \\ \frac{exp(s^e_{tj})}{\sum_{k=1}^{t-1} exp(s^e_{kj})}, \text{if } t > 1 \end{cases} , \alpha_{tj} = \frac{s^{temp}_{tj}}{\sum_{k=1}^{J} s^{temp}_{tk}} \tag{4}$$

where s_{tj}^{temp} represents the new alignment score with temporal dependence, and a_{tj} represents the attention weight at the current decoding step. Lastly, using the temporal attention mechanism, the model produces the following context vector for the input question:

$$z_t^e = \sum_{j=1}^{J} \alpha_{tj} h_e^j. \tag{5}$$

- *Dynamic Attention on SQL:* In order to perform well on this task, this model incorporates a dynamic attention mechanism to the decoder ((Shi et al., 2021), (Shi et al., 2019)), which allows it to dynamically attend on the previously created tokens. This is necessary because when generating the condition values, it is possible that the decoder will need to take into consideration not only the previously generated token, its own hidden states, and encoder context vector, but also additional factors such as the encoder. For encoders with $t > 1$, the alignment scores (denoted by $s_{t\tau}^d$, with $\tau \in \{1, ..., t-1\}$) on previously produced tokens may be computed in the same way as the alignment scores for encoders with $t > 1$. The formula calculates the attention weight for each token as

$$\alpha_{t\tau}^d = \frac{exp(s_{t\tau}^d)}{\Sigma_{i=1}^{t-1} exp(s_{ti}^d)} \tag{6}$$

and the formula of decoder-side context vector is:

$$z_t^d = \Sigma_{\tau=1}^{t-1} = \alpha_{t\tau}^d h_\tau^d \tag{7}$$

- *Controlled Generation and Copying:* The aim of this task is tackling these problems with the MIMIC-SQL dataset, as follows. The first problem is SQL queries have strict templates. The second problem is that table headers in queries are aggregated and condition columns, which normally don't show precisely in questions. The final problem is because some terms in the question are out of vocabulary (OOV), so the precondition values cannot be ideally obtained through questions.

  With the aim of solving the above problems, for the purpose of task token creation, the decoder integrates the generation network with the pointer network. The reason for this is because the pointer network has the capability of copying OOV tokens from the source and context sequences to the destination sequences. When using OOV words in the TREQS model, the main job is to generate the words in the vocabulary and set placeholders for them ([PH]). Basically, generating condition values in SQL queries is the unique used task. At step t, with the target of generating a token, the first job is computing the probability distribution on a vocabulary $\mathcal{V}$ as follows:

$$\tilde{h}_t^d = W_z\left(z_t^e \oplus z_t^d \oplus h_t^d\right) + b_z \tag{8}$$

$$P_{\mathcal{V},t} = softmax\left(W_{emb}(W_{d2v}\tilde{h}_t^d + b_{d2v})\right) \tag{9}$$

  After that, the probability of generating a token y_t when conjunction with the pointer is determined by this formula.

$$P(y_t) = \mathrm{p}_{gen,t} P_{gen}(y_t) + (1 - \mathrm{p}_{gen,t}) P_{ptr}(y_t) \tag{10}$$

  with $\mathrm{p}_{gen,t}$ is probability of using a generation network for token generation, and in this formula, $P_{gen}(y_t)$ is the probability of the network generation is computed by

$$P_{gen}(y_t) = \begin{cases} P_{\mathcal{V},t}(y_t) & y_t \in \mathcal{V} \\ 0 & otherwise \end{cases} \tag{11}$$

  and the probability of pointer network $P_{ptr}(y_t)$ is established as follows:

$$P_{ptr}(y_t) = \begin{cases} \Sigma_{j:x_j=y_t} \alpha_{tj}^e & y_t \in \mathcal{X} \cap \mathcal{V} \\ 0 & otherwise \end{cases} \tag{12}$$

with $\mathcal{X}$ on this formula is the set of all tokens in a question. Additionally, $p_{gen,t}$ is calculated by

$$p_{gen,t} = \sigma(W_{gen} z_e^t \oplus h_d^t \oplus E_{y_{t-1}} + b_{gen}) \tag{13}$$

In terms of the loss function, this model employs cross-entropy loss with the goal of increasing the log-likelihood of observed sequences to the greatest extent possible (ground-truth),

$$\mathcal{L} = -log P_\theta(\hat{y}|x) = \Sigma_{t-1}^{T} log P_\theta(\hat{y}_t|\hat{y}_{<t}, x) \tag{14}$$

with $\hat{y}$ is a ground-truth SQL sequence in the training data and θ is weight matrices W and biases b.

- *Placeholder Replacement:* During this task, once a query has been generated, this model replaces the position of each [PH] with a token in the original question. The replacement probability for a [PH] at a given time step t is calculated as follows:

$$P_{rps}(y_{t'}) = \begin{cases} \Sigma_{j:x_j=y_{t'}} \alpha^e_{t'j} & y_{t'} \in \mathcal{X} - \mathcal{V} \\ 0 & otherwise \end{cases} \tag{15}$$

A mask is applied on the attention weights, which may make use of the semantic link between previously created words and their nearby OOV words. The possible case is if at the step $t-1$, the model attends word x_i, it will have a high likelihood of attending the nearby words of x_i at step t. Any attention-based Seq2Seq model may be employed by this meta-algorithm.

TREQS++. TREQS++ or TREQS+Recover is the additional method with the added step as described below.

Recover Condition Values with Table Content: This step is necessary because the use of the translate-edit model may not provide complete confidence that all of the queries will be executed since the condition values may not be precise in certain cases. With the goal of retrieving the precise condition values from the projected ones, this model employs a condition value recovery method in order to address this issue.

In summary, methods in this paradigm operate by converting natural language inquiries into logic forms (Berant and Liang, 2014) that can describe the semantics of the whole query. Next, the results of the parsing are used to create structured queries (for example, SPARQL) that are used to search knowledge bases and get the responses. To generate semantic representations, (Hakimov et al., 2015) use a combinatory categorical grammar with handmade lexical items and lambda-type calculus expressions, as well as a combinatorial categorical grammar with handcrafted lexical items. As a result, the input utterances must be consistent with the grammatical rules. (Dubey et al., 2016) makes use of a logical model. Users can submit English-language questions to a target RDF knowledge base. They are first normalized into a canonical intermediate syntax known as normalized query structure, and then translated into SPARQL queries. (Yih et al., 2016) demonstrates that learning from labelled semantic parsers results in substantial improvements in overall performance. (Hamon et al., 2017) answer questions based on linked biological data utilizing a four-step procedure that starts with the linguistic and semantic annotation of the input question. The concept of developing a generic (pipeline) architecture for QA on linked data in order to foster developer collaboration was pioneered by (Singh et al., 2017), which previously combined building blocks in tailored systems, allowing for a semantic description of both quality assurance components and requirements. (Singh et al., 2018) is a platform that embraces component reuse by aggregating numerous key components for the purpose of solving QA chores and enabling the building of diverse QA pipelines.

4.2. Subgraph matching techniques

Subgraph matching constructs the query subgraph using a semantic tree, while others deviate significantly from this approach by building the subgraph from the entity. A natural language phrase may have many interpretations, each of which corresponds to a different set of semantic elements in the knowledge graph. After the semantic tree is located, the semantic relationships must be extracted from it before the semantic query graph is constructed. Several techniques (Maheshwari et al., 2019; Jin et al., 2019) use this approach.

gAnswer. gAnswer (Hu et al., 2017) is the most advanced subgraph matching method to convert natural language questions into query graphs that include semantic information. When the system has completed this process, it may convert query graphs into normal SPARQL queries, which will be performed in graph databases in order to provide

responses to users. For semantic disambiguation, the system uses a novel data-driven approach. The system retains various plans for entity and predicate mappings when creating query graphs and performs semantic disambiguation in the query execution phrase based on entity and predicate matches while generating the query graphs (incorrect mappings). The answers to gAnswer are obtained using a graph data-driven solution that is divided into two phases: offline and online. During the offline phase, a graph mining technique is used to determine the semantic equivalence of relation terms and predicates. After that, a paraphrase dictionary is constructed in order to document the semantic equivalence that has been discovered. The online phase is divided into two stages: the question understanding and question evaluation. During the question understanding step, a semantic query graph is constructed to reflect the user's intent by extracting semantic relations from the dependency tree of the natural language question using the previously constructed paraphrase dictionary, which is then used to construct the semantic query graph. Following that, a subgraph of the knowledge graph is chosen that, via subgraph isomorphism, corresponds to the semantic query network. In the question evaluation step, the final response is delivered depending on the subgraph that was chosen.

4.3. Template-based techniques

The usage of templates is critical in question answering (QA) over knowledge graphs (KGs), where user utterances are converted into structured questions via the use of semantic parsing. Using templates has the advantage of being traceable, and this may be used to create explanations for the users, so that they can understand why they get certain responses. These systems (Abujabal et al., 2017, 2018) use this approach.

TeBaQA. TeBaQA (Vollmers et al., 2021) is the most advanced template-based technique. Based on isomorphic graph patterns, TeBaQA (Vollmers et al., 2021) is used to create templates that can be reused across many SPARQL queries. TeBaQA can be illustrated as having the five following steps:

Preprocessing is the first step in which all queries are run through to eliminate semantically unnecessary terms and generate a collection of meaningful n-grams. The Graph-Isomorphism Detection and Template Classification phase makes use of the training sets to train a classifier based on a natural language question and an SPARQL query by analyzing the basic graph pattern for graph isomorphisms and then applying the classifier to the natural language question and the SPARQL query. Fundamentally, structurally identical SPARQL queries convey syntactically equivalent questions, according to this theory. When a query is asked, it is categorized into a ranked list of SPARQL templates, which is then shown on the screen. In the case of Information Extraction, TeBaQA collects all essential information from the question, such as entity names, relations between entities, and classes, and then determines the response type using an index set that is not dependent on the KG of the data being analyzed. This step involves inserting the extracted information into top-level template elements, determining what kind of SPARQL query to use, and adding query modifiers. It is next necessary to run the SPARQL queries that were generated and compare their responses to the anticipated response type. In the following ranking, all of the information, as well as the natural language question and the returned responses are taken into consideration. A two-step process is used to compile this ranking. In the first step, TeBaQA filters the results according to 1) the anticipated response type of the question in contrast to the actual answer type of the query and 2) the cardinality of the result set. Second, the quality of the remaining SPARQL queries is evaluated using TeBaQA.

4.4. Context-based techniques

Context-based systems attempt to comprehend questions from various perspectives, such as question analysis, classification of questions, answer path, context of answers, and type of answers. In §2, we have already introduced about this techniques.

QAsparql. QAsparql (Liang et al., 2021) is a model using five steps as listed below, to translate questions to SPARQL queries. Figure 2 shows an overview of QAsparql model.

- *Question Analysis* is the first part of QAsparql system. To be more specific, the syntactic characteristics of each question component are used to perform the following tasks: tokenize the question; find the appropriate part of speech tags for each of these tokens; recognize named entities; identify relationships between tokens; and calculate the dependency label of each question component. The questions are lemmatized in order to reduce the inflectional forms of a word to a common base form.. Furthermore, dependency parsing is also generated for question classification and query ranking tasks.

- *Question type classification* is the second part of QAsparql system. In this task, this model divides questions into three categories such as List, Count and Boolean. Some characteristics of each type of questions are listed below.

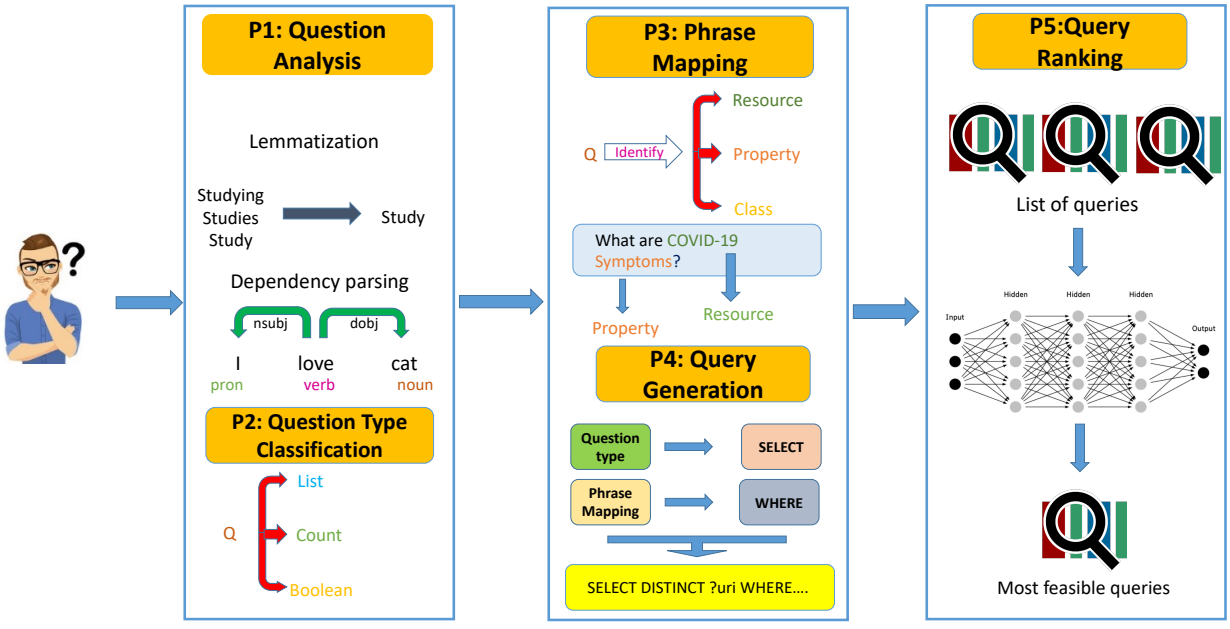


Figure 2: Overview of QAsparql model

The first one is List, with the beginning of Wh-questions or verb such as list and show and the answer usually is the list of resources. The next one is Count with the appearance with the word COUNT in SPARQL query and usually begins with How-questions and the answer of this type is number, sometimes is List due to the cases. The last one is Boolean with the exist of keyword ASK in SPARQL query with regular cases are YES/NO questions and the answers of this type are True or False. The approach of this model is firstly, the questions are lemmatized and next, TF-IDF (Baeza-Yates et al., 1999) technique is used for word embedding task. The formula of TF-IDF is $tfidf(t,d,C) = tf(t,d) \times idf(t,C)$, where tf is term frequency - number of term occurrences in a document, i.e. $tf(t,d) = \frac{f_{t,d}}{\Sigma_{t' \in d} f_{t',d}}$. In that, $f_{t,d}$ is number of times term t appears in a document d and $\Sigma_{t' \in d} f_{t',d}$ is total number of terms t in the document d. And $idf(t,C)$ is inverse document frequency - how much information the term provides in corpus C is calculated as $idf(t,C) = \log \frac{|C|}{|C_t|'}$ where $|C|$ is the number of documents in the corpus and $|C_t|'$ is $\{d \in C : t \in d\}$ the number of documents containing term t. Finally, Random Forrest (Breiman, 2001) model is used for questions classified into three types - List, Count and Boolean and SPARQL query, constructed based on the result of question classification.

- *Phrase mapping*. The main purpose of this task is to build the final queries using information based on knowledge graph. The information required to help SPARQL queries construction is divided into three types: resources, properties, and classes. Resources are literal and physical or abstract entities represented by any Internationalized Resource Identifier (IRI). For example, the IRI of city 'Ho Chi Minh' is `https://dbpedia.org/page/Ho_Chi_Minh_City` and string literal "VND" is currency code of VN in DBPedia. PProperties are a kind of resources used for illustrating attributes or relationships of other resources. An example of this type is `https://dbpedia.org/property/currencyCode`, which is the currency code of country. Finally is classes, classes are also kind of resources and are recognized by IRIs and may have associated with properties. For instance, `https://dbpedia.org/page/Vietnam` is a subclass of `https://dbpedia.org/ontology/country`

- *Query generation*. The main task of this step is deciding the SELECT clause and WHERE clause in an SPARQL query. Moreover, the purpose of query generation is constructing the set of triples of the graph pattern in the form <subject, predicate, object> triples and this is the basis for organizing an SPARQL query. At the Phrase mapping step, we also have the mapped resources, properties and classes which are used to generate these triples. Afterward, WHERE clause is established.

- *Query ranking*. In the Query Generation, this model creates many candidate queries for each question. Therefore, in this step, this model will rank these queries and select which are the most suitable queries. The approach of this model is based on Tree-structured Long-Short Term Memory (Tree-LSTM) (Tai et al., 2015) which is based on LSTM architecture (Hochreiter and Schmidhuber, 1997) and a new version is (Zaremba and Sutskever, 2014). On Tree-LSTM, the authors propose two extensions of LSTM - Child-Sum Tree-LSTM and the N-ary Tree-LSTM.

  The new version of LSTM (Zaremba and Sutskever, 2014) is

$$\text{LSTM} : h_t^{l-1}, h_{t-1}^l, c_{t-1}^l \rightarrow h_t^l, c_t^l \tag{16}$$

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} T_{2n,4n} \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix} \tag{17}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot g \tag{18}$$

$$h_t^l = o \odot \tanh(c_t^l) \tag{19}$$

  where $W_{s \rightarrow q}$ represents the weight matrix from s to q, t indexes the time-step, $b_{1 \rightarrow q}$ are the biases leading into q, $\sigma()$ is the logistic function, x_t is the input, i_t is the input gate , f_t is the forget gate and z_t is the input to the cell (which we call the hidden), c_t is the cell , o_t is the output gate, and h_t is the output of this module. Note also that the weight matrices from cell to gate vectors are diagonal $W_{c \rightarrow s}$, where s is one of the gates i, f, or o, depending on the cell type.

  Tree-LSTM units (indexed by j) are similar to conventional LSTM units in that they include an input gate (i_j), an output gate (o_j), an input and output cell (c_j), as well as an input and output hidden state (h_j). Unlike the conventional LSTM unit, Tree-LSTM units' gating vectors and memory cell updates are reliant on the states of many child units, which is a significant advantage over conventional LSTM units. Furthermore, instead of a single forget gate, the Tree-LSTM unit includes one forget gate f_{jk} for each child k, resulting in a total of four forget gates. Because of this, the Tree-LSTM unit is able to integrate information from each child in a targeted manner. Examples include learning to emphasize semantic heads in a semantic relatedness test or learning to maintain the representation of sentiment-rich children in a sentiment classification task using a Tree-LSTM model. Each Tree-LSTM unit, like the conventional LSTM unit, accepts an input vector x_j as input in our apps represents a word in a phrase as a vector representation of that word. The input word for each node is determined by the tree structure that was used to construct the network. For example, in a Tree-LSTM over a dependency tree, each node in the tree receives as input the vector corresponding to the head word, while in a Tree-LSTM over a constituency tree, the leaf nodes get as input the vectors corresponding to the relevant word vectors. Let c_j represent the set of children of node j in a given tree; the set of children of node j is denoted by Child-Sum Tree LSTM transition equations consist of the following terms and conditions:

$$\tilde{h}_j = \sum_{k \in C(j)} h_k, \tag{20}$$

$$i_j = \sigma(W_i x_j + U_i \tilde{h}_j + b_i) \tag{21}$$

$$f_{jk} = \sigma(W_f x_j + U_f h_k + b_f) \tag{22}$$

$$o_j = \sigma(W_o x_j + U_o \tilde{h}_j + b_o) \tag{23}$$

$$u_j = \tanh\left(W_u x_j + U_u \tilde{h}_j + b_u\right) \tag{24}$$

$$c_j = i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k \tag{25}$$

$$h_j = o_j \odot \tanh\left(c_j\right) \tag{26}$$

  In tree structures with a branching factor of at most N and children that are ordered, i.e., can be indexed from 1 to N, the N-ary Tree-LSTM may be used to find the shortest path between them. The hidden state and memory cell of each node j's kth child should be written as h_{jk} and c_{jk} for that node. The following are the N-ary Tree-LSTM

equations:

$$i_j = \sigma\Big(W_i x_i + \sum_{l=1}^{N} U_{li} h_{jl} + b_i\Big) \tag{27}$$

$$f_{jk} = \sigma\Big(W_f x_j + \sum_{l=1}^{N} U_{fkl} h_{jl} + b_f\Big) \tag{28}$$

$$o_j = \sigma\Big(W_o x_j + \sum_{l=1}^{N} U_{lo} h_{jl} + b_o\Big) \tag{29}$$

$$u_j = \tanh\Big(W_u x_j + \sum_{l=1}^{N} U_{lu} h_{jl} + b_u\Big) \tag{30}$$

$$c_j = i_j \odot u_j + \sum_{l=1}^{N} f_{jl} \odot c_{jl} \tag{31}$$

$$h_j = o_j \odot \tanh(c_j) \tag{32}$$

This model uses Tree-LSTM with the goal of mapping question and candidate queries to the sequences vector and compute the similarity of vectors.The output is the choice of candidate queries which are the highest similarity with input question.

QAsparql*. While carrying out a resource and property mapping task utilizing EARL(Dubey et al., 2018), we find that there is a significant difference between the two methods, as described in more detail below. Force gold 1 is the default method in QAsparql; however, we also uncover Force gold 2, which is a different mapping algorithm from the default algorithm. In empirical evaluation, we have discovered that Force gold 2 produces superior results to Force gold 1. In the §6 we will show the effectiveness of QAsparql (with Force gold 1) and QAsparql*(use Force gold 2). The algorithms for the two approaches are given below.

Algorithm 1 Force gold 1

Input: Golden List of Entity $\mathcal{G}$, List of Entity $\mathcal{E}$,
Output: List of Intersect entities $\mathcal{I}$, which are List of Entity $\mathcal{E}$ appears in Golden List $\mathcal{G}$

```
procedure FORCE GOLD 1
    I ← ∅
    I_i ← Item ∈ E
    U ← List of Uri ⊆ E
    U_i ← Uri ∈ U
    G_i ← Golden item ∈ G
    U_G ← List of Uri ⊆ G_≥
    U_Gi ← Uri ∈ U_G
    for I_i ∈ E do
        for U_i ∈ U do
            for G_i ∈ G do
                if U_Gi ∈ U_G then
                    I ← I∪ (G_i)
    return I
```

The hop count and connection count of each candidate node are determined using these methods. This information is sent to a classifier, which subsequently scores and ranks the features. After creating three types of things (Entity, Golden item, and the URI of each item), as seen in Algorithm 1, the next step is to turn them into a list. As a result, we have three different listings. List of Entities, List of Golden Item, and List of URI are some examples of lists. The primary goal of this method is to determine which entity URI is associated with the golden item URI. Algorithm 2 does the same job as Algorithm 1, but it employs a different approach to accomplish the goal. In the first phase, it determines which items in the list correspond to the golden item and moves those that do not into a separate list called the 'not found list'. After that, it will look for entities again, this time using surface form, in order to ensure that no entity is missed. Following that, it will remove the items that were not included, and a list of the items will be returned.

In summary, methods in this paradigm attempt to automatically convert natural language questions to structured queries. Then they access the knowledge base to acquire a set of possible answers. Finally, characteristics of the

Algorithm 2 Force gold 2

Input: Golden List of Entity $\mathcal{G}$, List of Entity $\mathcal{E}$, Surface Form $\mathcal{S}$ of Entity $\mathcal{E}$
Output: List of items $\mathbb{I}$

procedure FORCE GOLD 2
  $NF \leftarrow \emptyset, \mathbb{I} \leftarrow \emptyset$
  $I_i \leftarrow Item \in \mathcal{E}, G_i \leftarrow$ Golden item $\in \mathcal{G}, S_i \leftarrow$ Surface form $\in G_i$
  $U_i \leftarrow$ Uri $\in G_i, UI_i \leftarrow$ Uri $\in \mathcal{E}$
  for $G_i \in \mathcal{G}$ **do**
    $idx \leftarrow$ Closest string between Surface form from S_i and List of item $\mathcal{S}$
    if $idx \neq$ -1 **then**
      if Fist element of $U_i \notin UI_{idx}$ **then**
        Element of length of $UI_{idx} - 1 \leftarrow$ First element of U_i
      $\mathcal{S} \leftarrow \mathcal{S} \setminus S_{idx}$
    else
      $NF \leftarrow NF \cup G_i$
  $NF_i \leftarrow$ Item $\in NF$
  for $NF_i \in NF$ **do**
    if length of $\mathcal{S} > 0$ **then**
      $idx \leftarrow$ First key of surfaces
      First element of $UI_{idx} \leftarrow$ First element of U_i
      $\mathcal{S} \leftarrow \mathcal{S} \setminus S_{idx}$
    else
      $\mathcal{E} \leftarrow \mathcal{E} \cup NF_i$
  $keys \leftarrow$ List of key of $\mathcal{S}$
  $keys \leftarrow$ Sort all element of keys with descending order
  for $idx \in$ keys **do**
    $\mathcal{E} \leftarrow \mathcal{E} \setminus I_{idx}$
  return $\mathbb{I} \leftarrow \mathcal{E}$

Table 3
Characteristics comparison between KBQA techniques.

Technique	Features	Query language	Paradigm	Number of steps	Domain	Natural language
TREQS	Deep learning	SQL, SPARQL	Embedding	5	Single	Single
TREQS + Recover	Deep learning	SQL, SPARQL	Embedding	6	Single	Single
gAnswer	Hand-crafted	SPARQL	Subgraph matching	4	Multiple	Multiple
TeBaQA	Hand-crafted	SPARQL	Template-based	5	Multiple	Multiple
QAsparql	Hybrid	SPARQL	Context-based	5	Multiple	Multiple
QAsparql*	Hybrid	SPARQL	Context-based	5	Multiple	Multiple

question and candidates are retrieved in order to score them with the goal of determining the correct response. (Yao and Van Durme, 2014) extracts linguistic information from a question to create a question feature graph. Then a topic graph which composed with topic nodes and other relative nodes is created in Freebase, with each node representing a potential answer. Finally, feature exacted from the topic graph and candidate answers are merged to determine the correct answer. To extract hand-crafted features for queries, this technique uses dependency parse results. (Dong et al., 2015) proposed a multicolumn convolutional neural network for understanding and learning the distributed representations of questions from three distinct perspectives: answer path, answer context, and answer type. (Xu et al., 2016) offered a neural network-based relation extractor for retrieving candidate answers from Freebase and subsequently inferring these responses from Wikipedia. To be more exact, the approach entails subdividing the original query using a collection of syntactic patterns. (Hao et al., 2017) proposed a model for dynamically representing questions and their associated scores based on the various potential answer features via the cross-attention method. Additionally, they make use of the global knowledge contained inside the underlying knowledge base, with the goal of incorporating this information into the depiction of the answers. As a result, it may reduce the out-of-vocabulary issue, allowing the cross attention model to more precisely depict the query.

4.5. Summary

In summary, we use six representative systems — TREQS, TREQS+ Recover, gAnswer, TeBaQA, QAsparql and QAsparql* across domains, natural languages, and query languages. Table 3 compares the key characteristics of each technique studied in this benchmark.

5. Benchmarking framework

In this section, we introduce our benchmarking framework. We first present an overview of the architecture and the role of each module. After that, we describe the datasets and KBs that were utilized in the benchmark. In addition, we provide explanations of the metrics that were used to evaluate the various KBQA methods. Finally, we discuss the implementation details and reproducibility environment of our benchmarking tool in terms of evaluating the outcomes of knowledge base over question answering.

5.1. Framework

A major aim of this research is to develop a flexible but powerful tool to aid in the comparison and analysis of various KBQA methods and techniques of substitution. In order to do this, we have created a framework that makes use of the original performance analysis of these methods. Figure 3 depicts the component-based architecture of the framework, which is comprised of three layers: the data access layer, the computation layer, and the application layer. The data access layer abstracts the underlying data items and feeds them to the higher l levels of the application stack. The application layer communicates with users in order to receive customizable parameters and to display the results of computations performed by the computing layer. The computational layer is comprised of two major components: the evaluation module and the algorithm module, which work together to perform the experiment and assessment. We anticipate that future research will be able use our methodology to more readily compare their ideas with current state of the art methods. It is adaptable and extendable in the manner described, since new techniques and measurements can be simply incorporated into the systems.

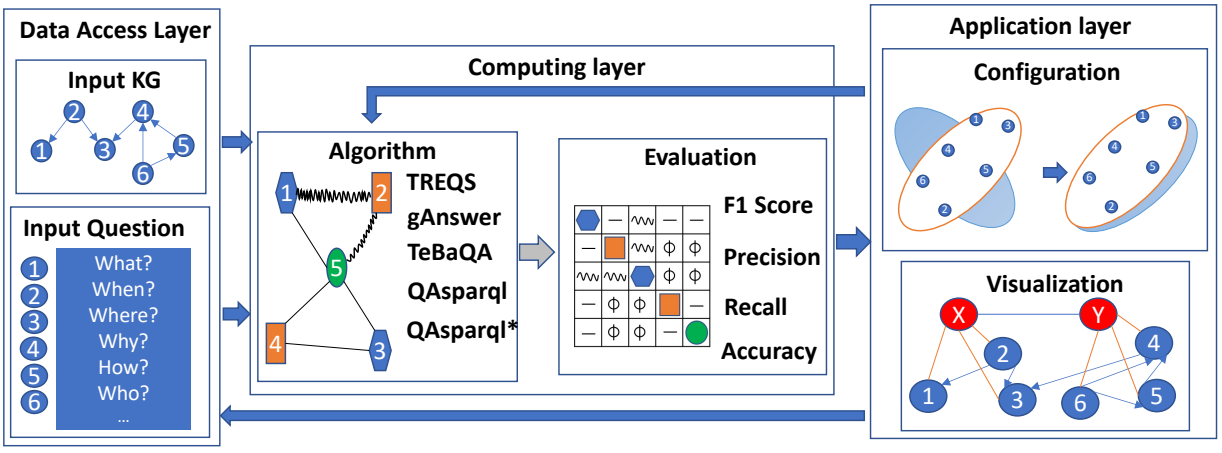


Figure 3: Benchmarking framework

5.2. Knowledge bases

DBPedia. DBpedia (Auer et al., 2007) is a knowledge graph that was created by extracting structured data from Wikipedia and displaying it. The DBpedia introduced an ontology consisting initially of 492 classes with 53,827 predicates which has increased to 768 classes and 60,231 predicates, and other well-defined concepts are included in the ontology. Entries of the same type show a high degree of similarity amongst themselves. When, humans supply only information such as name, date of birth, and nationality for all ideas, these is referred to as "domain-specific facts." The fact that the majority of the attributes were obtained solely from infoboxes is also a disadvantage of the ontology. These, on the other hand, differ even when dealing with the same type of objects. DBpedia likewise suffers from a lack of internal interlinking. It makes use of the same grouping mechanism as Wikipedia, namely, aggregation based on Wikipedia categories, to organise its information. The key-value pairs in the Wikipedia infoboxes serve as the primary source for DBPedia extraction. An ontology of sorts of infoboxes is created using a crowd-sourced approach, and keys used in those infoboxes are mapped to attributes in the ontology created using that process.

Freebase. Freebase (Bollacker et al., 2008) is a tuple database that is both practical and scalable, and it is used to organise common human information. Using Freebase, you may generate, organise, and preserve data in a collaborative

Table 4
Knowledge bases comparison

KBs	DBpedia	Freebase	Wikidata
Number of triples	411 885 960	3 124 791 156	748 530 833
Number of instances	20 764 283	115 880 761	142 213 806
Number of entities	4 298 433	49 947 799	18 697 897
Number of classes	736	53 092	302 280
Number of relations	2819	70 902	1874
No. of unique predicates	60 231	784 977	4839

environment. In its present state, Freebase includes more than 125,000,000 tuples, over 4000 types, and more than 7000 attributes. A graph-query API that uses the Metaweb Query Language (MQL) as a data query and manipulation language provides read and write access to Freebase to the general public. MQL offers an easy-to-use object-oriented interface to the tuple data in Freebase, and it is intended to make the development of collaborative, Web-based data-oriented applications much easier than before.

Wikidata. As with Freebase, Wikidata (Vrandečić and Krötzsch, 2014) is a cooperatively updated knowledge graph administered by the Wikimedia foundation, which also hosts the many language versions of Wikipedia. Wikidata, like Freebase, is cooperatively edited. After its shutdown in 2016, the data it contained is held in Wikidata. Wikidata is unusual in that it allows for the addition of provenance information to each axiom — for example, the source and date for a city's population statistic.

5.3. KBs statistics

According to (Färber and Rettinger, 2018), the statistical results of comparisons of knowledge graphs (KGs) are shown in the table 4.

Triples. All of the KGs under consideration are very big. DBPedia is the smallest KG, whereas Freebase is the biggest KG in terms of the number of triples. Freebase is the largest KG in terms of triples. It seems that there is a relationship between the method used to build up a KG and its size. For example, automatically generated KGs tend to be bigger since the constraints of integrating new information become less onerous as the KG grows. A significant effect is made by datasets that have been imported into the knowledge bases, such as MusicBrainz into Freebase. These datasets have a significant impact on the number of triples and facts in the knowledge base. In addition, the way data is modelled has a significant effect on the number of triples. Consider the following example: If n-ary relations are represented in N-Triples format (as in the case of Wikidata), more intermediary nodes must be modeled, which results in many more triples than with simple statements. Last but not least, the number of languages that are supported has an impact on the number of triples.

Classes. The number of classes varies greatly across the KGs, ranging from 736 (DBpedia) to 53K (Freebase) and 300K (Wikidata). Despite having a large number of classes, Wikidata also has the highest proportion of classes that are actually utilized, when compared to other databases (i.e., classes with at least one instance). The reason for this may the fact that heuristics are used in the selection of suitable Freebase categories as Wikidata classes. Wikidata, on the other hand, includes a large number of classes, but only a tiny percentage of them are actually utilized at the instance level. It should be noted, however, that this is not always a problem.

Relations and Predicates. When searching through the KGs, it is typical to discover relations that are only seldom used: just 5% of the Freebase relations are used more than 500 times, and about 70% are never used at all. A quarter of the DBpedia ontology's relations are used more than 500 times in DBpedia, whereas half of the ontology's relations are used just once.

Instances and Entities. Freebase has by far the most entities of any database on the internet today. Due to the fact that each statement is instantiated, Wikidata exposes a disproportionately large number of instances in comparison to entities (in the sense of instances that represent real world objects), resulting in approximately 74M instances that are not entities.

Domain. Despite the fact that all of the KGs under consideration are cross-domain, the domains are not evenly distributed across the KGs. In addition, the domain coverage varies significantly across the KGs. The domains that are well represented in the KGs are strongly influenced by the datasets that have been incorporated into the KGs. As a

result of the importation of MusicBrainz facts into Freebase, the domain of media has a high knowledge representation (77%) in Freebase. The domain of persons has the greatest number of entries in DBpedia, which is most likely owing to the use of Wikipedia as a data source.

5.4. Datasets

We construct a sizable collection of benchmarking datasets that cover different domains and different languages.

Multi-domain datasets. Multi-domain attempts to improve performance by distributing it over several domains, and has been successfully used in a variety of areas.

- *Generic*: Generic data are data gathered from a wide range of sources, including mathematics, physics, computer science, and a variety of other fields. When it comes to system development, life testing is more appropriate during the latter stages of the process, when problems such as reliability growth and system substantiation are important. LC-QUAD (Trivedi et al., 2017) is Large-Scale Complex Question Answering Dataset. It consists of 5000 questions and answers pair with the intended SPARQL queries over knowledge base in DBPedia. LC-QUAD 2.0 (Dubey et al., 2019) is the new version of LC-QUAD with several updates as follows. First, the number of questions is 30000. Second, all questions consist of paraphrased versions via crowdsourcing tasks. The last section includes queries with multiple user intentions as well as inquiries requiring SPARQL string operations. Furthermore, in addition to the pure DBPedia knowledge base as the target knowledge base, LC-QUAD 2.0 also offers a new version of DBPedia, which is DBPedia based on Wikidata.

- *Biomedical*: Biomedical data is data about (or that might fairly be considered to be about) human health. Using a dataset with a particular area, such as biomedical, may assist us in reducing the size of a dataset with a specified goal and domain. A large-scale healthcare Question-to-SQL dataset, MIMICSQL (Wang et al., 2020), was created by utilizing the publicly available real world Medical Information Mart for Intensive Care III (MIMIC III) (Johnson et al., 2016; Goldberger et al., 2000) dataset to generate 10,000 Question-to-SQL pairs. MIMICSQL* (Park et al., 2020) is the modified version of MIMICSQL which improves on the disadvantages of MIMICSQL as their tables are unnormalized and simpler than the tables used in actual hospitals. MIMICSPARQL is an SPARQL-based version of MIMICSQL* with the aim to compare the performance between SQL and SPARQL. COVID-KGQA includes our additional datasets with more 1000 questions about COVID-19. The details of our dataset are described in the next subsection

Multilingual-language datasets. Multilingual datasets are datasets which are supported many languages. With multilingual datasets, a model's performance may be tested over a wide range of languages, and the differences between each language can be determined.

Question Answering over Linked Data (QALD) is a set of evaluation campaigns on question answering over linked data that began in 2011 and lasted through 2020. QALD-9 (Ngomo, 2018) is the most recent edition, and it has 408 questions that were collected and selected from prior challenges. It is accessible in eleven different languages including English, Spanish, German, Italian, French, Dutch, Romanian and Farsi. The XML format is used for QALD-1 to QALD-5 and JSON format is for version from 6 to 9. COVID-KGQA is also multilingual dataset with appearance of two languages such as English and Vietnamese.

Our new benchmarking data. In addition to the dataset provided above, which was covered in the preceding section, we also produced a bilingual dataset about COVID-19 to further enhance the diversity of biomedical field and multilingual datasets. Using the latest version of DBPedia as a starting point, we developed corpus for COVID-19 QA over knowledge graph DBPedia (COVID-KGQA) with more than 1,000 bilingual question-answer pairs. The format of data we use is approximately the same as the QALD dataset (Ngomo, 2018) with a simplified version from CBench (Orogat et al., 2021) which will help the evaluation progress through QA systems to be much more straightforward. An example of our data is illustrated in Figure 4.

5.5. Data analysis

The benchmark datasets are shown in Table 5, along with the number of questions contained in each and the KGs they target. QALD (mentioned above) now contains nine benchmarks (QALD-1 to QALD-9). The challenge organizers provided both a training dataset and a test dataset for each task. When these datasets were first created, they comprised at a minimum the NL question, an SPARQL query, and the appropriate responses. Later, more information such as

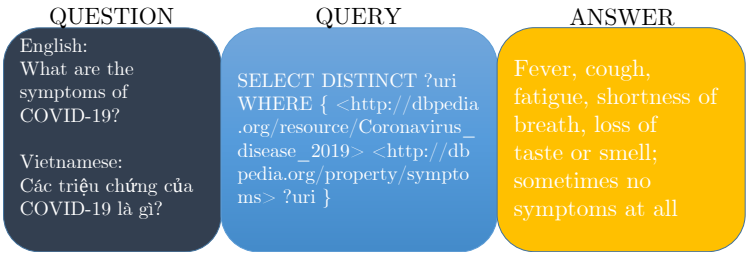


Figure 4: Example of our dataset

Table 5
List of datasets

Dataset	Questions	KB	Domain
MIMICSQL	10,000	Generated Table	Biomedical
MIMICSQL*	10,000	Generate Table	Biomedical
MIMICSPARQL	10,000	Generated KG	Biomedical
COVID-KGQA	1,000	DBPedia	Biomedical, Multi-language
LC-QUAD	5,000	DBPedia	Generic
QALD-9	408	DBPedia	Generic, Multi-language
QALD-8	315	DBPedia, Wikidata	Generic, Multi-language
QALD-7	530	DBPedia, Wikidata	Generic, Multi-language
QALD-6	431	DBPedia, LinkedSpending	Generic, Multi-language
QALD-5	334	DBPedia	Generic, Multi-language
QALD-4	321	DBPedia	Generic, Multi-language
QALD-3	397	DBPedia, MusicBrainz	Generic, Multi-language
QALD-2	344	DBPedia, MusicBrainz	Generic, Multi-language
QALD-1	199	DBPedia, MusicBrainz	Generic, Multi-language

Table 6
Biomedical datasets comparison

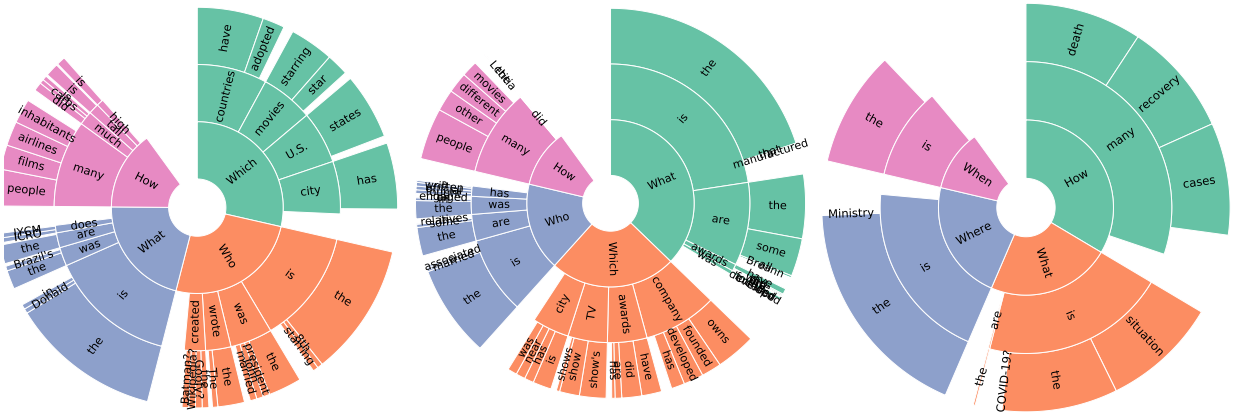
Dataset	MIMICSQL	MIMICSQL*	MIMICSPARQL
# of patients	46,520	100	100
# of tables	5	9	9
# of Question-SQL pairs	10,000	10,000	10,000
Average NL question length (in words)	16.45	16.45	16.45
Average SQL query length	21.14	44,68 (21.04)	28.98 (27.28)

keywords, response type, and information about necessary aggregation functions was added to the datasets, as well as a new knowledge base other than DBpedia and hybrid question answering based on RDF and free text.

LC-QuAD is a dataset for semi-automated question creation. SPARQL templates are created and translated into natural question templates in a computer-assisted manner. These broad templates are manually converted into questions in natural language by a team of experts. After being released in 2017, the LC-QuAD 1.0 dataset has been made publicly available. It was announced in January 2019 that LC-QuAD 2.0 would be released. There are two datasets in total: a test dataset and a training dataset. While LC-QuAD 1.0 only supported SPARQL queries over pure DBPedia (2016 version), LC-QuAD 2.0 supports SPARQL queries over Wikidata and the Wikidata-migrated DBpedia (2018 version).

The MIMICSQL dataset, which includes the Question-SQL pairings as well as the logical format for slot filling procedures, was created expressly for the purpose of this Question-to-SQL creation process. The Question-SQL pairs for the MIMICSQL dataset were gathered through a combination of human and machine generation. MIMIC-SPARQL uses Generated KG from the table that including in the original dataset. The process of converting from table to KG is illustrated below. The nine tables of MIMICSQL* were broken down into three categories, and each column was assigned to either an entity, a literal, or a relation. This simplified the process of extracting triples from the nine tables. Entities are defined as the primary or foreign key columns of a table, while literals are defined as the non-key

With regard to SQL queries, Table 6 indicates that the query length is almost twice that of SPARQL, which is based on the statistics of the dataset. The reason for this is because a single JOIN in SQL needs 11 tokens (including the operators "=" and "."), but just three tokens in SPARQL (i.e. subject, predicate, object). It is this inherent distinction between relational tables and a knowledge graph in terms of how to link information that gives rise to the syntactic difference between SQL and SPARQL (joining tables and hopping triples). Figure 5 depicts a distribution of the trigrams that contain the first letters of the questions' inquiries. Additionally, in order to have a better understanding of the lexical diversity of questions in the datasets, we identify the trigram patterns that appear the most frequently in the questions. We can see that the dataset contains a large number of different language constructs, with the following types of questions appearing most frequently; How, What, Who, Which appearing the most frequently in QALD-9 (Figure 5a), How, Who, Which, What appearing the most frequently in LC-QUAD (Figure 5b), and When, Where, What, Who appearing the most frequently in COVID-KGQA (Figure 5c). Furthermore, the questions 'What is', 'Who is', 'How many', and 'Which countries' account for the majority of all types of questions in QALD-9; 'How many', 'Who is', 'Which TV', and 'What is', in LC-QUAD, and 'When is', 'Where is', 'What is', and 'How many' in COVID-KGQA.



(a) QALD-9 (b) LC-QUAD (c) COVID-KGQA

Figure 5: Distribution of the most popular question prefixes. (a) is QALD-9, (b) is LC-QUAD, (c) is COVID-KGQA (Our)

5.6. Measurements

F1-score. Let Q denotes the number of questions in benchmark , G is the number of answers processed by the system for given question Q, and A is the number of corrected answers extracted by executing query q from question Q over knowledge graph KG. The precision of the answers shows how many of them are accurate. Recall shows how many of the returned responses correspond to the gold standard. Precision (P) and Recall (R) are defined as follows:

$$P = \frac{|\,A\,|}{|\,G\,|}$$

Precision is defined as the percentage of relevant questions that are allocated to a category out of all the questions that are assigned by the system. The high accuracy score indicates that the number of questions that are irrelevant but are classified as relevant instances by the classifier (false positives), is extremely minimal. However, Precision, does not take into account how many of relevant instances in a category are correctly allocated, making it impossible to assess overall categorization performance.

$$R = \frac{|\,A\,|}{|\,Q\,|}$$

In contrast to Precision, Recall is a measure of how many questions in a category are properly allocated to the right category. High Precision indicates that an algorithm assigns more relevant instances to a category than irrelevant

instances (quality), but high Recall indicates that an algorithm allocated the majority of the relevant instances to a category (quantity).

In order to assess performance, the trade-off between Precision and Recall must be considered, and using only one of them is not sufficient to evaluate the performance. As a result, the accuracy will rise due to the limited number of real positive assignments and the large number of irrelevant instances that are allocated to a category, among other factors. As a result of the small number of real positive assignments, there will be a significant number of false negative assignments, resulting in a low Recall score for the assignment. The F1 measure is intended to overcome the limitations of both measures, Recall and Precision, by measuring performance as the harmonic mean of both measurements, Recall and Precision, as well as the performance of the F1 measure. When compared to the individual performance metrics Recall and Precision, the F1 measure is more trustworthy in terms of gauging overall performance. As (Hripcsak and Rothschild, 2005), F1 is defined as follows:

$$F1 = \frac{1 + \beta^2}{\beta^2 * (1/P + 1/R)}$$

β weighs whether precision or recall is more important, and equal emphasis is given to both when $\beta = 1$. In trials, most researchers choose $\beta = 1$ since it gives no benefit to precision or recall. In this case, we use $\beta = 1$ for our experiment evaluation. This metric is used by many experiments through benchmark (Orogat et al., 2021), system (Liang et al., 2021) and corpus (Usbeck et al., 2017).

Execution accuracy. Execution accuracy (Acc_{EX}) is computed as the accuracy of the response retrieved through SQL/SPARQL (Wang et al., 2020).

$$Acc_{EX} = N_{EX}/N$$

where N represents the total number of Question-SQL pairings in the MIMICSQL database, and N_{EX} indicates the number of created SQL queries that have the potential to provide accurate responses. Execution accuracy can also include questions which are created with erroneous SQL queries but which result in valid query results. For that reason, we use another metric - Logical form accuracy, which eliminates the drawback of execution accuracy in the execution.

Logical form accuracy. The Logical form accuracy (Acc_{LF}) is used for if a string match exists between a produced SQL query and a ground truth query (Park et al., 2020). In order to compute Acc_{LF}, we must compare the produced SQL/SPARQL with the true SQL/SPARQL, token by token.

$$Acc_{LF} = N_{LF}/N$$

where N_{LF} counts how many requests are completely matched to the ground truth query.

Computation time. Another measure for KBQA methods is the amount of time it takes to complete a task. The processing speed of various QA applications is often constrained, as are the computer resources available to the program. As a consequence, the amount of time it takes to complete the process becomes critical. The fairness of our benchmark is ensured by the fact that all algorithms are built and tested on the same standard (i.e., they are all tested on the same benchmarks or setup).

5.7. Implementation details

Hyperparameter configuration. Table 7 summarizes the main configurations of parameters which we use to to conduct experiment on the first three datasets. These one will help reproduce the results better.

Reproducibility Environment. The experiments with the first three MIMIC datasets are conducted on an Intel(R) Xeon(R) CPU @ 2.30GHz system with 35.5 GB Ram and Tesla P100 graphics cards. Four other datasets are carried out using Intel(R) Xeon(R) CPU E5-2620 @ 2.10GHz with 125 GB Ram and Nvidia Geforce GTX 1080 graphics cards. All algorithms were evaluated using the same standards (same benchmarks and settings) in order to guarantee fairness.

6. Experiments and results

As a further step, we go through our findings from when we applied the benchmark to the KBQA benchmark system algorithms. The primary aim of the experiments is not only to compare the performances, but also to investigate the impacts of data features on the behavior of the performance. In this section, we provide the empirical findings.

Table 7
Configuration of parameters

Dataset	Method	Batch size	Step decay	Step size	Learning rate	Random state	Epoch	Batch id
MIMICSQL	TREQS (Dev)	16	0.8	2	0.0005	42	6	400
	TREQS + Recover (Dev)	16	0.8	2	0.0005	42	6	400
	TREQS (Test)	16	0.8	2	0.0005	42	6	400
	TREQS + Recover (Test)	16	0.8	2	0.0005	42	6	400
MIMICSQL*	TREQS (Dev)	16	0.8	2	0.0005	1234	7	300
	TREQS + Recover (Dev)	16	0.8	2	0.0005	1234	7	300
	TREQS (Test)	16	0.8	2	0.0005	1234	7	300
	TREQS + Recover (Test)	16	0.8	2	0.0005	1234	7	300
MIMICSPARQL	TREQS (Dev)	48	0.1	2	0.0005	1234	12	166
	TREQS (Test)	48	0.1	2	0.0005	1234	12	166

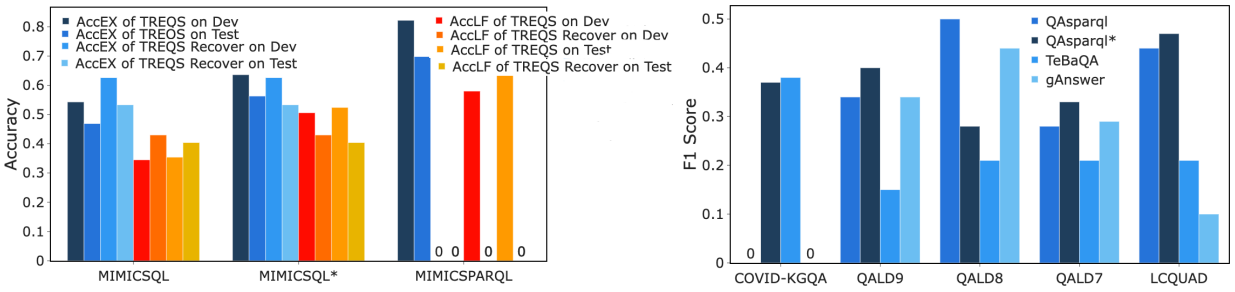
Table 8
End-to-end comparison on biomedical datasets

Dataset	Method	Acc_{EX}	Acc_{LF}	Time (s)
MIMICSQL	TREQS (Dev)	0.543	0.345	0.161
	TREQS + Recover (Dev)	**0.626**	**0.43**	
	TREQS (Test)	0.469	0.354	
	TREQS + Recover (Test)	**0.533**	**0.404**	
MIMICSQL*	TREQS (Dev)	**0.636**	**0.506**	0.311
	TREQS + Recover (Dev)	0.626	0.43	
	TREQS (Test)	**0.563**	**0.524**	
	TREQS + Recover (Test)	0.533	0.404	
MIMICSPARQL	TREQS (Dev)	**0.822**	0.580	0.25
	TREQS (Test)	0.698	**0.641**	

6.1. End-to-end comparison

As part of this investigation, we looked at the overall performance of knowledge bases when it comes to question answering on three biomedical datasets, four common existing datasets and one new dataset. The findings are presented in Table 8 and Table 9. Among the top performers showed in Table 8, TREQS and TREQS + Recover are the most accurate on the first three biomedical datasets, with the highest accuracy on the fourth. TREQS + Recover achieves a better result on MIMICSQL, but it does not perform as well as TREQS on MIMICSQL* when it comes to total performance on both the development and test datasets. TREQS + Recover is unable to operate with MIMICSPARQL at this time due to the design of TREQS + Recover, which is to Recover Condition Values with Table Content, which means that it is currently unable to perform with a knowledge graph. Figure 6a illustrates the visualization of our results on first three biomedical datasets and Figure 7 shows the statistic of average time with 100 random questions. For the most cases, Acc_{EX} and Acc_{LF} perform better on the development set than on the test set with TREQS technique. However, the results are different with TREQS; Acc_{LF} performs better on the test set. Regarding time, MIMICSPARQL is the quickest in nearly all situations, whereas MIMICSQL* is the slowest. The potential explanation is that MIMICSQL* is more sophisticated and normalized than MIMICSQL, resulting in a longer calculation time. Additionally, our research is the first to discover that despite the fact that MIMICSPARQL performs better than others in terms of performance, it also performs better in terms of time.

Table 9 compares the four systems and their scores for the four metrics - Precision, Recall, F1 and Time used in our paper with five datasets(four generic dataset and one additional biomedical dataset) and Figure 6b shows visualisation of them in terms of F1 . Our system has been chosen alongside two new systems that are up to date and scheduled to be launched in 2021. These systems performed outstandingly in the previous comparisons (Liang et al., 2021; Vollmers et al., 2021). The unique aspect of this discovery is that altering the entity and resource algorithm, as stated in §4.4, improves performance over and beyond the default method used before. This is referred to as QAsparql* in order to differentiate it from the default. gAnswer is our next chosen system, which is state of the art in the QALD-9 challenge and is performed well on QALD-7, QALD-8, and LC-QUAD. In terms of datasets, we select four existing datasets in addition to a large number of additional datasets that are split into the LC-QUAD and QALD series, series, which are



(a) On biomedical KBs

(b) On generic KBs

Figure 6: Accuracy comparison

Table 9
Performance comparison of on multilingual datasets

Dataset	QAsparql				QAsparql*				TeBaQA				gAnswer			
	P	R	F1	Time	P	R	F1	Time	P	R	F1	Time	P	R	F1	Time
COVID-KGQA* (Our)	0.00	0.00	0.00	**0.26**	0.29	0.52	0.37	**2.69**	**0.38**	**0.38**	**0.38**	13.2	0.00	0.00	0.00	40.99
QALD-9 (Train)	0.4	0.45	0.42	5.89	0.33	0.71	0.45	32.37	0.17	0.18	0.16	8.5	0.39	0.43	0.39	13.03
QALD-9 (Test)	0.32	0.36	0.34	9.56	0.3	0.6	0.4	27.45	0.16	0.15	0.15	8.15	0.33	0.37	0.34	13.30
QALD-8 (Train)	0.35	0.49	0.41	5.16	0.31	0.67	0.42	7.9	0.21	0.22	0.21	7.52	**0.45**	0.51	**0.46**	12.93
QALD-8 (Test)	**0.57**	0.44	0.5	3.56	0.2	0.44	0.28	7.35	0.21	0.22	0.21	8	0.42	**0.54**	0.44	**12.07**
QALD-7 (Train)	0.33	0.55	0.42	6.42	**0.38**	0.77	**0.51**	5.69	0.2	0.21	0.2	7.92	0.43	0.5	0.44	13.09
QALD-7 (Test)	0.21	0.44	0.28	1.74	0.24	0.5	0.33	7.3	0.22	0.23	0.21	**7.72**	**0.29**	0.35	0.29	13.04
LC-QUAD (Test)	0.34	**0.62**	**0.44**	5.41	0.31	**0.98**	0.47	13.75	0.21	0.22	0.21	16.45	0.09	0.11	0.1	60.45

the most frequently used datasets in the DBPedia. We choose QALD versions 7 to 9 from the QALD series since the target KG of these datasets are from version 2016, which is suited with a system that is nearly as current as the QALD series, and this will help these systems perform better. Our COVID-KGQA uses the latest version of DBPedia.

For the COVID-KGQA dataset, we used four systems to evaluate the effectiveness of the existing system with our new bench-mark corpus in COVID-19. gAnswer is deployed via gStore with version 2016-10 of DBPedia as the guidance for gAnswer systems, but with other systems, we run experiments with the latest version of DBPedia. In this version, the knowledge graph doesn't have any information about COVID-19. For that reason, gAnswer achieves the worst performance at both F1 and Time in our corpus. On the other hand, the best performance is TeBaQA and QAsparql, despite of the fact that although TeBaQA is higher than QAsparql 0.01 in F1, QAsparql is better than TeBaQA in some fields, as discussed in §6.3. Although TeBaQA has the best performance in our new benchmarking dataset, this system is worse with other datasets where the performance range is only 0.15 to 0.21. The possible reasons for this, is that QALD is a more difficult benchmark that includes a large number of questions that need sophisticated queries with more than two triples to be answered. A more in-depth examination showed that questions from QALD-9 often needed sophisticated templates that were either absent from the training queries or had very limited assistance in the training set. The template-based approach in TeBaQA demonstrates the weakness compared to the Information retrieval-based technique as QAsparql and QAsparql* and Subgraph matching as gAnswer except in LC-QUAD. QAsparql*, as our new finding approach, accomplishes the best performance in almost all corpora except COVID-KGQA (with shorter than approximately 0.01) and QALD-8. This demonstrates that improving the algorithm in entity matching can help the system get a better result.

6.2. Running time

Figure 7 shows the results of the experiment. In the first three datasets, we randomly select 100 questions and compare the time performance with these datasets. Overall, MIMICSPARQL with a knowledge graph approach achieves the best result compared to two other corpora. When compared to MIMICSPARQL, MIMICSQL* has almost twice the query length. A single JOIN in SQL needs 11 tokens (including the operators "=" and "."), while only three tokens are required for a single hop in SPARQL (i.e., subject, predicate, and object). A further difference between SQL and SPARQL is that the model must understand the hierarchy between a table and its columns as well as the relationships between tables. It is this inherent distinction between relational tables and a knowledge graph in terms of

how to link information that gives rise to the syntactic difference between SQL and SPARQL (joining multiple tables versus hopping across triples). Thus, the graph-based method performs much better than the table-based approach. (Park et al., 2020). We compare the running time of four systems in five other datasets. We test all questions in the test set of five existing corpora. In COVID-KGQA, gAnswer reaches a peak and becomes the highest time in overall. In brief, the explanation is as follows: gAnswer is installed through gStore with the default knowledge graph version set to 2016-10, but our dataset is about COVID-19, which was introduced in 2019, thus gAnswer takes longer to query but returns no results.

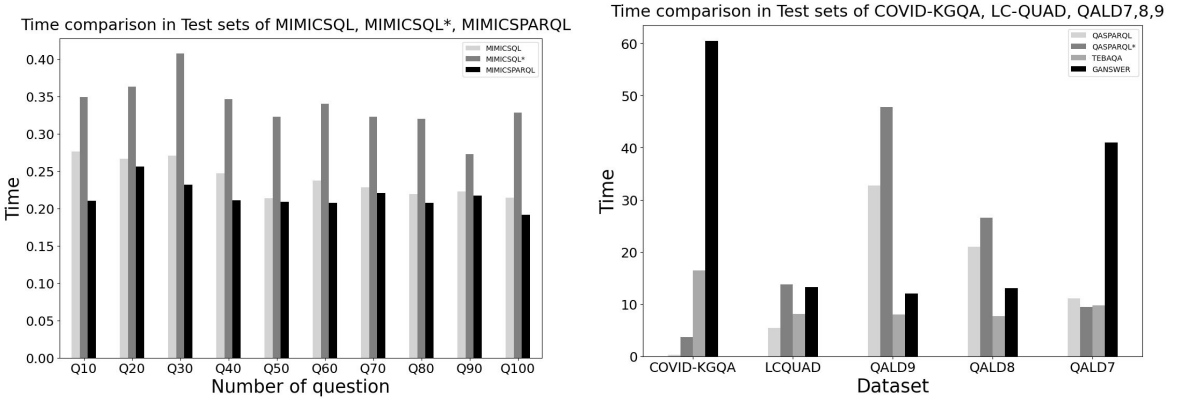


Figure 7: End to end time comparison on Test set

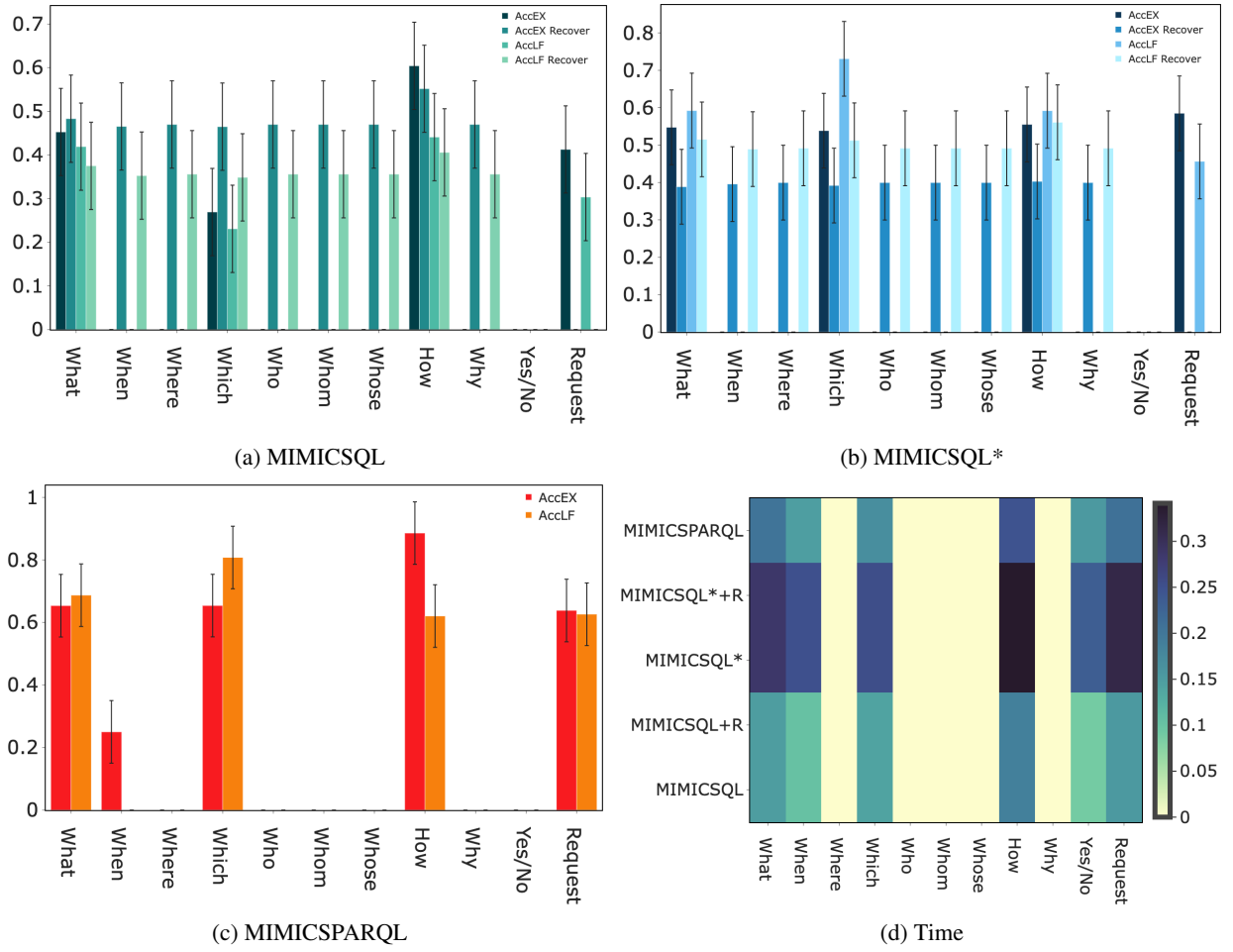
6.3. Influence of Question Taxonomy

Over the course of this investigation, the impacts of question type on question answering are investigated using knowledge graph methods. From these experiments, we conduct our studies on test set of LCQUAD and COVID-KGQA because the quantity of question of them is the highest and almost the same (1000 questions). Figure 8 depicts the results of MIMIC datasets, and Figure 9, Figure 10, wwhich are conducted with four systems in COVID-KGQA and LCQUAD, as stated in §6.3. According to (Pomerantz, 2005), the wh-question taxonomy comprises six types: Who, What, When, Where, Why, and How. Additionally, throughout the data analysis process, we discover that the dataset contains other types such as Whose and Whom. On the other hand, certain inquiries are not Wh-questions, such as Yes/No questions, which contain phrases starting with the words Can, May, Might, Am, Is, Are, Was, Were, Will, Do, Did, Does, and some request questions, such as Count, Provide, Give, Tell, Specify, Find, Get, Familiarize, Let, and Look. As a result, we undertake this experiment using 11 distinct sorts of questions.

For the MIMICSQL dataset, Figure 8a depicts the AccEX, AccLF, and Time of TREQS and TREQS+Recovery for 11 question categories, with regard to each sort of query. The plot depicts the AccEX, AccEX Recovery, AccLF, AccLF Recovery, and Time for each question type across the whole test set in this dataset. It is apparent that TREQS+Recover outperformed TREQS for all question categories on this dataset in AccEX when using the Question-to-SQL techniques TREQS and TREQS+Recover. AccLF outperforms all other variables except What, How, and Request. Similarly, MIMICSQL* and MIMICSPARQL results are depicted in Figure 8b and Figure 8c. In MIMICSQL*, the exception is TREQS+Recovery, which is inferior to TREQS in terms of What, Which, How, and Request. TREQS outperformed other types of question with AccLF at What, Which, How, and Request, but other types outscored TREQS+Recover. Time as shown in Figure 8d and is roughly the same across three datasets when the lowest values are at Where, Who, Whom, Whose, and the greatest values are at How.

Regarding the F1 in COVID-KGQA and LCQUAD results, these are illustrated in Figure 9 and Figure 10. With respect to specific question categories, TeBaQA surpassed both of them. For COVID-KGQA, QAsparql* outperforms in the categories of What, When, Where, and Which, whereas TeBaQA hits a high in the category of How. When compared to other SOTA baseline techniques, gAnswer and QAsparql perform the poorest in this corpus. For the LCQUAD, QAsparql outperformed TeBaQA at What, Who, and Request, while QAsparql outperformed TeBaQA at How, Request. Aside from that, QAsparql*, which is the most remarkable baseline, outperformed at the following questions: Where, Which, and Yes/No. With regards to Time, overall, for COVID-KGQA, the sorts of questions Where

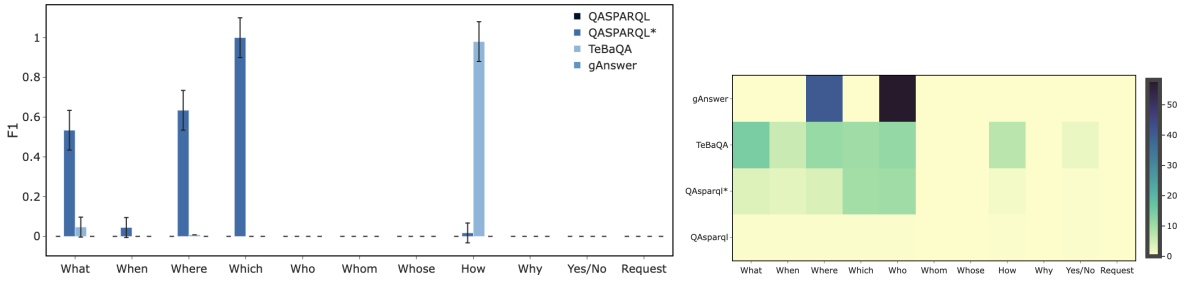
and Who are the most time-consuming across all methods, with gAnswer being the slowest of the bunch. In most situations, our newly found approach, QAsparql*, outperforms QAsparql in terms of time savings in most cases.



(a) MIMICSQL (b) MIMICSQL*

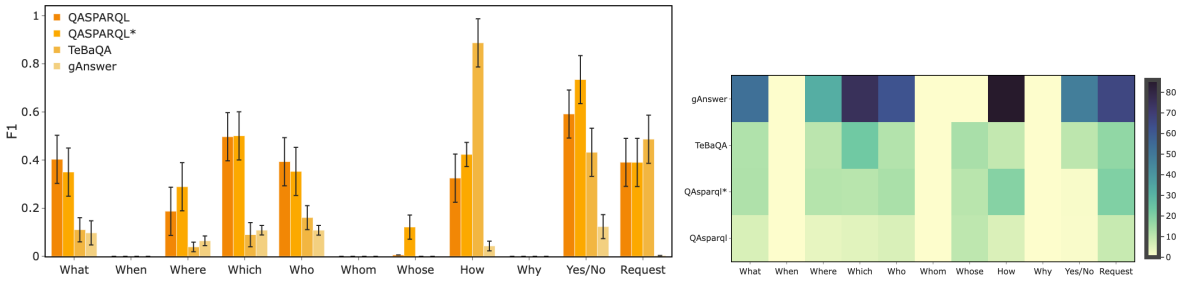
(c) MIMICSPARQL (d) Time

Figure 8: Comparison on MIMICSQL, MIMICSQL*, MIMICSPARQL with influence of question type from test set, the underscore on the charts represent value 0



(a) F1 (b) Time

Figure 9: Comparison on COVID-KGQA with influence of question type from test set, the underscore on the charts represent value 0



(a) F1 (b) Time

Figure 10: Comparison on LCQUAD with influence of question type from test set, the underscore on the charts represent value 0

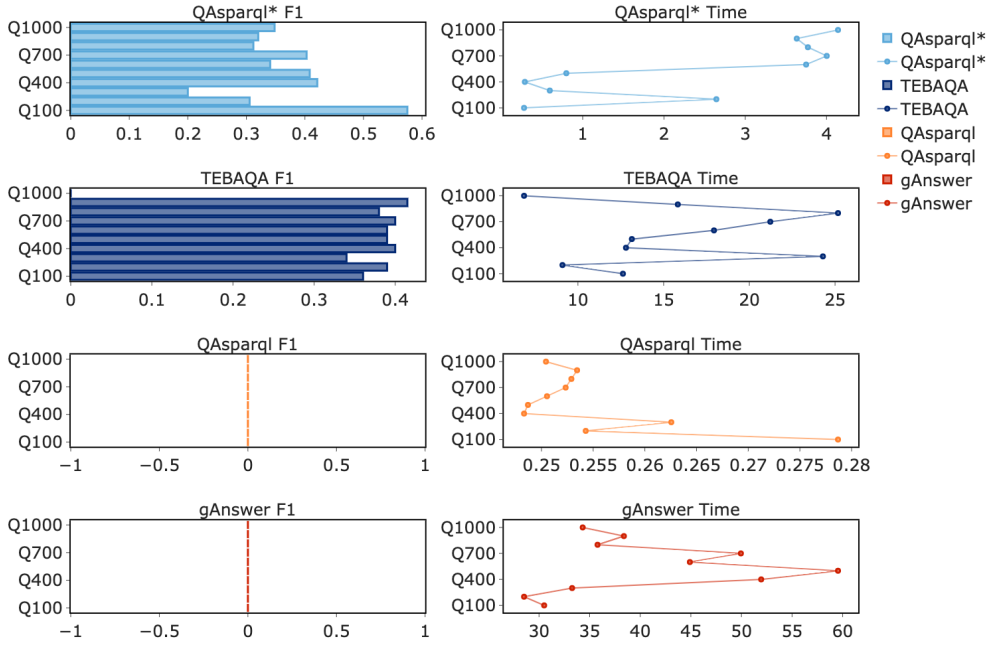


Figure 11: Comparison on COVID-KGQA with influence of question type from test set

6.4. Effects of Quantity of Questions

We then look at the impacts of another characteristic, the quantity of questions. Figure 13 depicts the results of an experiment in which the quantity factor of the question is changed from each 100 question segments order to determine the effectiveness of the Acc_{EX} and Acc_{LF} through 1000 questions of each dataset. Overall, the accuracy of the test set increases substantially when the quantity of questions varies from the 200th question to the 700th question, decreases from the 800th to 900th, and the performance of Acc_{EX} is better than Acc_{LF} in the most cases. The performance of both metrics usually reaches a peak from the 500th question to the 700th. Regarding Time in MIMICSQL, MIMICSQL*, MIMICSPARQL, as represented in Figure 13, in terms of overall performance on the Test sets, MIMICSQL and MIMICSQL* are quite close to one another. To be more specific, both datasets are high at the first 100 questions and decrease from the next 200 and 300 questions; after that, it also increases at the 400th and 500th questions, the 800th and 900th questions, and the 900th and 1000th questions, and it decreases from the 900th to the 1000th questions. The main difference is that in MIMICSPARQL, the time for both questions is initially extremely small, but it steadily increases until it reaches a maximum at the 1000th question. Regarding F1 in COVIG-KGQA and LCQUAD, as shown in Figure 9 and Figure 10, the performance of all assessed techniques as measured by both F1 and Time is illustrated

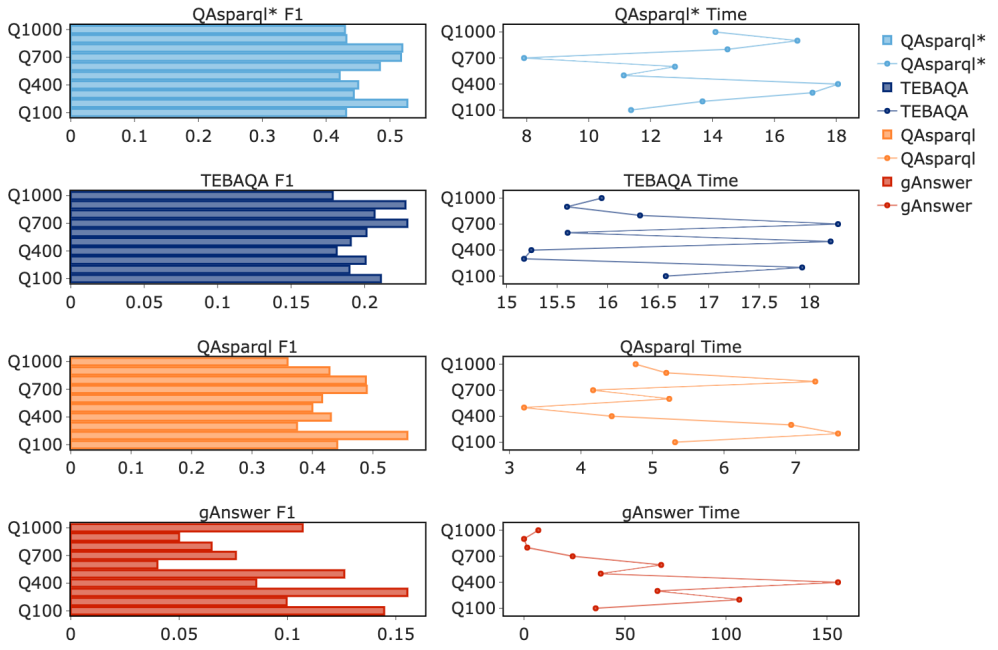


Figure 12: F1 and Time comparison on LCQUAD with influence of question quantity from test set

in the above figures. In terms of information retrieval-based techniques, it is evident that QAsparql* outperformed all other methods with all questions over 0.4 and 300 questions over 0.5, and that it was followed by QAsparql* in terms of overall performance. Conversely, the gAnswer technique, which uses Subgraph Matching, had the lowest F1 scores of all. QAsparql* beat all other techniques when compared to all other methods. Regarding time, QAsparql achieved the greatest performance, followed by QAsparql* and finally TeBaQA; on the other hand, gAnswer achieved the lowest performance one more time beside of F1 score.

7. Discussion

This article is the first to describe a large-scale, repeatable dual benchmarking investigation of question answering over knowledge base and knowledge graph. We examine a variety of datasets and techniques, then analyze the features of representative models. A new dataset covering a wide variety of forgery contents is then created using an extendable architecture that we have designed and built. In the next step, we perform a dual bench-marking comparison and analyze the findings in order to offer complete recommendations for researchers and end users.

Performance guidelines. To assist end users in selecting a suitable solution for a certain application need, we offer the following set of recommendations, which are based on our experimental findings:

- Overall, the most effective methods in both F1 and Time are information retrieval based methods such as QAsparql and QAsparql*. Even state-of-the-art KGQA algorithm, such as gAnswer, may be defeated in perfect circumstances. We have also shown efficiency of QAsparql and QAsparql* in other benchmarking experiments. However, in the instance of question type, the efficacy of QAsparql was diminished. Consequently, TeBaQA continues to face difficulties due to question type and performs best in terms of multiple criteria, but it has low computation time although it can find relevant items belonging to different subtopics. On the other hand, TREQS surpasses MIMICSQL* and MIMICSPARQL, with the exception of MIMICSQL, in terms of AccEX and AccLF. The performance of TREQS+Recover is superior to that of TREQS in MIMICSQL, but it is worse in other datasets. In terms of performance over time, MIMICSQL surpasses the competition and shows the superiority of the knowledge graph over the table.

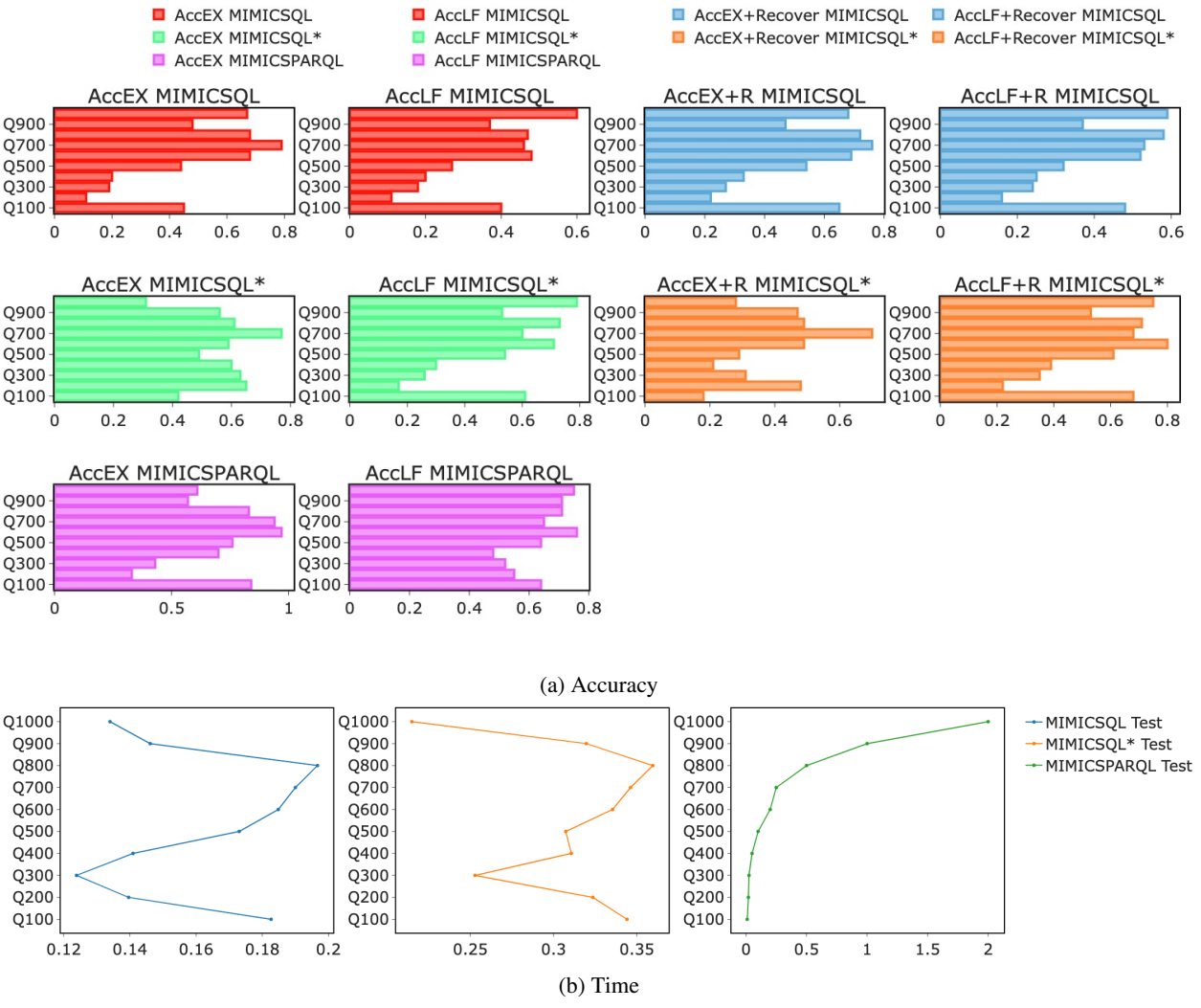


Figure 13: The effect of the number of question to Accuracy and Time on MIMICSQL, MIMICSQLSTAR and MIMICSPARQL

- For the influence of question type, overall, in terms of AccEX and AccLF, TREQS+Recover outperforms in almost question type in MIMICSQL except How, Request and What, Which, How, Yes/No in MIMICSQL*. In MIMICSPARQL, TREQS+Recover doesn't work and TREQS is effective at AccEX in How and Request questions. With COVID-KGQA, in term of F1, QAsparql* is the best choice in terms of What, When, Where question types while TeBaQA is the best in terms of the How question type. With LCQUAD, in terms of F1, QAsparql and QAsparql* are the winners. Specifically, QAsparql outperforms in What and Who questions and QAsparql* is better at Where, Which, Whose, How, Yes/No and Request questions. In contrast, TeBaQA is the best if the relevance difference between How and Yes/No is the main concern. Regarding Time, QAsparql is the first runner up in nearly all kinds of questions, while QAsparql* is the second runner up in a negligible number of questions. But the two other runners-up, TeBaQA and gAnswer, are not significantly slower. We suggest using TeBaQA.

- Regarding how the quantity of questions affects the performance of COVID-KGQA, we suggest not using gAnswer or QAsparql as these algorithms are not good at any field. Among the better algorithms, QAsparql* and TeBaQA are both good choices in terms of diversity. Considering how the quantity of question affects the performance of LCQUAD, QAsparql* should be used as it can return the highest number of subtopics. Once

Table 10
Performance guideline

category	winner	1st runner-up	worst
Overall	QAsparql*	QAsparql*	gAnswer
Question type (COVID-KGQA)	QAsparql*	QAsparql	gAnswer, QAsparql
Question type (LCQUAD)	QAsparql*	QAsparql	gAnswer
Question quantity (COVID-KGQA)	QAsparql*	QAsparql	gAnswer, QAsparql
Question quantity (LCQUAD)	QAsparql*	QAsparql	gAnswer
AccEX	TREQS+Recover	TREQS	TREQS
AccLF	TREQS+Recover	TREQS	TREQS
Multi-domain	QAsparql*	QAsparql*	gAnswer
Multi-language	QAsparql*	QAsparql*	gAnswer
Cross-Query Language	TREQS	TREQS	TREQS+Recover

again, TREQS+Recover exceeds both MIMICSQL* and MIMICSQL in terms of AccEX and AccLF except MIMICSPARQL.

- With regards to Multi-domain and Multi-language, we recommend that you avoid using gAnswer or QAsparql since these algorithms are ineffective in any area. When it comes to algorithm variety, QAsparql* and TeBaQA are both excellent options among the better algorithms. About Cross-Query Language, TREQS excels in all datasets, but TREQS+Recover, which is unable to meet this criterion for performance.

These findings are summarized in Table 10, where the best, the second-best and the worst techniques are shown for each performance category.

Improvement recommendations. The above experimental findings enable us to recommend several strategies to improve the performance. First, subsequent studies should check and consider the number of questions and the diversity of question types when developing their techniques. Second, the findings of the study also indicate that we should concentrate on lengthy questions and multiple response types to enhance the overall performance.

8. Conclusion

In this article, we discuss the significant degree of variance in the benchmarks used to assess question answering over knowledge graphs, as well as the implications of this variation. Experimental evidence indicates that these changes have an impact on the reported results for multiple-choice responding systems. To mitigate the impacts of such changes, we offer a new comparative study, a fine-grained benchmark suite that comes with a prepackaged set of popular benchmarks that target various popular knowledge graphs and is designed to be used in conjunction with a new comparative study. This research is simple to apply and may be expanded upon. Additionally, it offers a fine-grained analysis of the questions and queries that have been processed, based on a variety of characteristics, in addition to the conventionally recognized quality ratings for each question and query. It is possible for users to rapidly discover the strengths and weaknesses of their assessed question answering system by using the results of this study.

As a result of this research, benchmarks may be examined for taxonomies of the questions and inquiries included within the benchmark. Using existing benchmarks, we demonstrate that they differ substantially in terms of these characteristics, making it unreliable to evaluate question answering systems using just a small sample of them. Question answering over knowledge bases (KBQA) still faces significant difficulties in converting natural languages (NL) to query languages (e.g. SPARQL). Moreover, most KBQA systems answer simple questions just for one triple, but complicated inquiries require complex questions with sub-quests or multiple functions. In recent years, using natural language to query these knowledge graphs has been the most common method to overcoming the aforementioned difficulties. Consequently, the evaluation outcomes of QA systems may also rely on the system dataset examined. We analyzed currently available KBQA datasets on various challenges in order to provide an overview and identify unique features. These tools will be used to analyze benchmarks and evaluate KBQA systems in the future. The paper also introduces a new benchmarking corpus about COVID-19 (COVID-KGQA) and an extensive comparison of the multi approach methods for this dataset. Three approach methods (Subgraph Matching, Template, and Information Retrieval) and four algorithm adaptation-based methods (gAnswer, TeBaQA, QAsparql, and QAsparql*) are evaluated. Four well-known corpora (QALD-7, QALD-8, QALD-9, and LCQUAD) are evaluated as base learners for these methods.

References

Abujabal, A., Saha Roy, R., Yahya, M., Weikum, G., 2018. Never-ending learning for open-domain question answering over knowledge bases, in: Proceedings of the 2018 World Wide Web Conference, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE. p. 1053–1062. URL: https://doi.org/10.1145/3178876.3186004, doi:10.1145/3178876.3186004.

Abujabal, A., Yahya, M., Riedewald, M., Weikum, G., 2017. Automated template generation for question answering over knowledge graphs, in: Proceedings of the 26th International Conference on World Wide Web, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE. p. 1191–1200. URL: https://doi.org/10.1145/3038912.3052583, doi:10.1145/3038912.3052583.

Antoniou, G., Van Harmelen, F., 2004. Web ontology language: Owl, in: Handbook on ontologies. Springer, pp. 67–92.

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z., 2007. Dbpedia: A nucleus for a web of open data, in: The semantic web. Springer, pp. 722–735.

Azmy, M., Shi, P., Lin, J., Ilyas, I., 2018. Farewell Freebase: Migrating the SimpleQuestions dataset to DBpedia, in: Proceedings of the 27th International Conference on Computational Linguistics, pp. 2093–2103.

Baeza-Yates, R., Ribeiro-Neto, B., et al., 1999. Modern information retrieval. volume 463. ACM press New York.

Bahdanau, D., Cho, K., Bengio, Y., 2014. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473 .

Berant, J., Liang, P., 2014. Semantic parsing via paraphrasing, in: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Baltimore, Maryland. pp. 1415–1425. URL: https://aclanthology.org/P14-1133, doi:10.3115/v1/P14-1133.

Berners-Lee, T., Hendler, J., Lassila, O., 2001. Scientific american: Feature article: The semantic web: May 2001. Scientific American 4.

Bizer, C., Heath, T., Berners-Lee, T., 2011. Linked data: The story so far, in: Semantic services, interoperability and web applications: emerging concepts. IGI global, pp. 205–227.

Bizer, C., Heath, T., Idehen, K., Berners-Lee, T., 2008. Linked data on the web (ldow2008), in: Proceedings of the 17th international conference on World Wide Web, pp. 1265–1266.

Bollacker, K., Evans, C., Paritosh, P., Sturge, T., Taylor, J., 2008. Freebase: a collaboratively created graph database for structuring human knowledge, in: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pp. 1247–1250.

Breiman, L., 2001. Random forests. Machine learning 45, 5–32.

Brickley, D., Guha, R.V., Layman, A., 1999. Resource description framework (rdf) schema specification .

Chakraborty, N., Lukovnikov, D., Maheshwari, G., Trivedi, P., Lehmann, J., Fischer, A., 2021. Introduction to neural network-based question answering over knowledge graphs. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 11, e1389.

Consortium, W.W.W., et al., 2012. Owl 2 web ontology language document overview .

Costa, J.O., Kulkarni, A., 2018. Leveraging knowledge graph for open-domain question answering, in: 2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI), IEEE. pp. 389–394.

Cyganiak, R., Wood, D., Lanthaler, M., Klyne, G., Carroll, J.J., McBride, B., 2014. Rdf 1.1 concepts and abstract syntax. W3C recommendation 25, 1–22.

Dong, L., Wei, F., Zhou, M., Xu, K., 2015. Question answering over Freebase with multi-column convolutional neural networks, in: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), Association for Computational Linguistics, Beijing, China. pp. 260–269. URL: https://aclanthology.org/P15-1026, doi:10.3115/v1/P15-1026.

Dubey, M., Banerjee, D., Abdelkawi, A., Lehmann, J., 2019. Lc-quad 2.0: A large dataset for complex question answering over wikidata and dbpedia, in: International semantic web conference, Springer. pp. 69–78.

Dubey, M., Banerjee, D., Chaudhuri, D., Lehmann, J., 2018. Earl: joint entity and relation linking for question answering over knowledge graphs, in: International Semantic Web Conference, Springer. pp. 108–126.

Dubey, M., Dasgupta, S., Sharma, A., Höffner, K., Lehmann, J., 2016. Asknow: A framework for natural language query formalization in sparql, in: Sack, H., Blomqvist, E., d'Aquin, M., Ghidini, C., Ponzetto, S.P., Lange, C. (Eds.), The Semantic Web. Latest Advances and New Domains, Springer International Publishing, Cham. pp. 300–316.

Färber, M., Rettinger, A., 2018. Which knowledge graph is best for me? arXiv preprint arXiv:1809.11099 .

Finegan-Dollak, C., Kummerfeld, J.K., Zhang, L., Ramanathan, K., Sadasivam, S., Zhang, R., Radev, D., 2018. Improving text-to-SQL evaluation methodology, in: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Melbourne, Australia. pp. 351–360. URL: https://aclanthology.org/P18-1033, doi:10.18653/v1/P18-1033.

Goldberger, A.L., Amaral, L.A., Glass, L., Hausdorff, J.M., Ivanov, P.C., Mark, R.G., Mietus, J.E., Moody, G.B., Peng, C.K., Stanley, H.E., 2000. Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals. circulation 101, e215–e220.

Hakimov, S., Unger, C., Walter, S., Cimiano, P., 2015. Applying semantic parsing to question answering over linked data: Addressing the lexical gap, in: Biemann, C., Handschuh, S., Freitas, A., Meziane, F., Métais, E. (Eds.), Natural Language Processing and Information Systems, Springer International Publishing, Cham. pp. 103–109.

Hamon, T., Grabar, N., Mougin, F., 2017. Querying biomedical linked data with natural language questions. Semantic Web 8, 581–599.

Hao, Y., Zhang, Y., Liu, K., He, S., Liu, Z., Wu, H., Zhao, J., 2017. An end-to-end model for question answering over knowledge base with cross-attention combining global knowledge, in: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Vancouver, Canada. pp. 221–231. URL: https://aclanthology.org/P17-1021, doi:10.18653/v1/P17-1021.

Harris, S., Seaborne, A., Prud'hommeaux, E., 2013. Sparql 1.1 query language. W3C recommendation 21, 778.

He, W., Liu, K., Liu, J., Lyu, Y., Zhao, S., Xiao, X., Liu, Y., Wang, Y., Wu, H., She, Q., Liu, X., Wu, T., Wang, H., 2018. DuReader: a Chinese machine reading comprehension dataset from real-world applications, in: Proceedings of the Workshop on Machine Reading for Question Answering, Association for Computational Linguistics, Melbourne, Australia. pp. 37–46. URL: https://aclanthology.org/W18-2605,

doi:10.18653/v1/W18-2605.
Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P.F., Rudolph, S., et al., 2009. Owl 2 web ontology language primer. W3C recommendation 27, 123.
Hochreiter, S., Schmidhuber, J., 1997. Long Short-Term Memory. Neural Computation 9, 1735–1780. doi:10.1162/neco.1997.9.8.1735.
Hripcsak, G., Rothschild, A.S., 2005. Agreement, the f-measure, and reliability in information retrieval. Journal of the American medical informatics association 12, 296–298.
Hu, S., Zou, L., Yu, J.X., Wang, H., Zhao, D., 2017. Answering natural language questions by subgraph matching over knowledge graphs. IEEE Transactions on Knowledge and Data Engineering 30, 824–837.
Jin, H., Luo, Y., Gao, C., Tang, X., Yuan, P., 2019. Comqa: Question answering over knowledge base via semantic matching. IEEE Access 7, 75235–75246. doi:10.1109/ACCESS.2019.2918675.
Johnson, A.E., Pollard, T.J., Shen, L., Li-Wei, H.L., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Celi, L.A., Mark, R.G., 2016. Mimic-iii, a freely accessible critical care database. Scientific data 3, 1–9.
Jurafsky, D., Martin, J.H., 2019. Speech and language processing (3rd draft ed.).
Kacupaj, E., Zafar, H., Lehmann, J., Maleshkova, M., 2020. Vquanda: Verbalization question answering dataset, in: Harth, A., Kirrane, S., Ngonga Ngomo, A.C., Paulheim, H., Rula, A., Gentile, A.L., Haase, P., Cochez, M. (Eds.), The Semantic Web, Springer International Publishing, Cham. pp. 531–547.
Lassila, O., Swick, R., 1997. Wd-rdf-syntax-971002 resource description framework (rdf) model and syntax. worldwide web consortium.
Li, D., Madden, A., 2019. Cascade embedding model for knowledge graph inference and retrieval. Information Processing & Management 56, 102093.
Liang, S., Stockinger, K., de Farias, T.M., Anisimova, M., Gil, M., 2021. Querying knowledge graphs in natural language. Journal of Big Data 8, 1–23.
Luong, T., Pham, H., Manning, C.D., 2015. Effective approaches to attention-based neural machine translation, in: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Lisbon, Portugal. pp. 1412–1421. URL: https://www.aclweb.org/anthology/D15-1166, doi:10.18653/v1/D15-1166.
Maheshwari, G., Trivedi, P., Lukovnikov, D., Chakraborty, N., Fischer, A., Lehmann, J., 2019. Learning to rank query graphs for complex question answering over knowledge graphs, in: International semantic web conference, Springer. pp. 487–504.
McGuinness, D.L., Van Harmelen, F., et al., 2004. Owl web ontology language overview. W3C recommendation 10, 2004.
Nallapati, R., Zhou, B., Gulcehre, C., Xiang, B., et al., 2016. Abstractive text summarization using sequence-to-sequence rnns and beyond. arXiv preprint arXiv:1602.06023 .
Ngomo, N., 2018. 9th challenge on question answering over linked data (qald-9). language 7.
Nguyen, K.V., Tran, K.V., Luu, S.T., Nguyen, A.G.T., Nguyen, N.L.T., 2020. Enhancing lexical-based approach with external knowledge for vietnamese multiple-choice machine reading comprehension. IEEE Access 8, 201404–201417. doi:10.1109/ACCESS.2020.3035701.
Noraset, T., Lowphansirikul, L., Tuarob, S., 2021. Wabiqa: A wikipedia-based thai question-answering system. Information Processing & Management 58, 102431.
Ochieng, P., 2020. Parot: Translating natural language to sparql. Expert Systems with Applications: X 5, 100024.
Orogat, A., Liu, I., El-Roby, A., 2021. CBench: Towards Better Evaluation of Question Answering Over Knowledge Graphs. Proceedings of the VLDB Endowment (PVLDB) 14.
Park, J., Cho, Y., Lee, H., Choo, J., Choi, E., 2020. Knowledge graph-based question answering with electronic health records. arXiv preprint arXiv:2010.09394 .
Pomerantz, J., 2005. A linguistic analysis of question taxonomies. Journal of the American Society for Information Science and Technology 56, 715–728.
Prudhommeaux, E., 2008. Sparql query language for rdf. http://www. w3. org/TR/rdf-sparql-query/ .
Qiao, C., Hu, X., 2020. A neural knowledge graph evaluator: Combining structural and semantic evidence of knowledge graphs for predicting supportive knowledge in scientific qa. Information Processing & Management 57, 102309.
Rajpurkar, P., Zhang, J., Lopyrev, K., Liang, P., 2016. SQuAD: 100,000+ questions for machine comprehension of text, in: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics. pp. 2383–2392.
Rebele, T., Suchanek, F., Hoffart, J., Biega, J., Kuzey, E., Weikum, G., 2016. Yago: A multilingual knowledge base from wikipedia, wordnet, and geonames, in: International semantic web conference, Springer. pp. 177–185.
Romeo, S., Da San Martino, G., Belinkov, Y., Barrón-Cedeño, A., Eldesouki, M., Darwish, K., Mubarak, H., Glass, J., Moschitti, A., 2019. Language processing and learning models for community question answering in arabic. Information Processing & Management 56, 274–290.
Shi, T., Keneshloo, Y., Ramakrishnan, N., Reddy, C.K., 2021. Neural abstractive text summarization with sequence-to-sequence models. ACM Transactions on Data Science 2, 1–37.
Shi, T., Wang, P., Reddy, C.K., 2019. LeafNATS: An open-source toolkit and live demo system for neural abstractive text summarization, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations), Association for Computational Linguistics, Minneapolis, Minnesota. pp. 66–71. URL: https://www.aclweb.org/anthology/N19-4012, doi:10.18653/v1/N19-4012.
Shin, S., Jin, X., Jung, J., Lee, K.H., 2019. Predicate constraints based question answering over knowledge graph. Information Processing & Management 56, 445–462.
Siciliani, L., Basile, P., Lops, P., Semeraro, G., . Mqald: Evaluating the impact of modifiers in question answering over knowledge graphs .
Singh, K., Both, A., Sethupat, A., Shekarpour, S., 2018. Frankenstein: A platform enabling reuse of question answering components, in: European Semantic Web Conference, Springer. pp. 624–638.
Singh, K., Lytra, I., Vidal, M.E., Punjani, D., Thakkar, H., Lange, C., Auer, S., 2017. Qaestro – semantic-based composition of question answering pipelines, in: Benslimane, D., Damiani, E., Grosky, W.I., Hameurlain, A., Sheth, A., Wagner, R.R. (Eds.), Database and Expert Systems

Applications, Springer International Publishing, Cham. pp. 19–34.

Tai, K.S., Socher, R., Manning, C.D., 2015. Improved semantic representations from tree-structured long short-term memory networks, in: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), Association for Computational Linguistics, Beijing, China. pp. 1556–1566. URL: https://aclanthology.org/P15-1150, doi:10.3115/v1/P15-1150.

Tanon, T.P., Weikum, G., Suchanek, F., 2020. Yago 4: A reason-able knowledge base, in: European Semantic Web Conference, Springer. pp. 583–596.

Trivedi, P., Maheshwari, G., Dubey, M., Lehmann, J., 2017. Lc-quad: A corpus for complex question answering over knowledge graphs, in: International Semantic Web Conference, Springer. pp. 210–218.

Usbeck, R., Ngomo, A.C.N., Haarmann, B., Krithara, A., Röder, M., Napolitano, G., 2017. 7th open challenge on question answering over linked data (qald-7), in: Semantic web evaluation challenge, Springer. pp. 59–69.

Vollmers, D., Jalota, R., Moussallem, D., Topiwala, H., Ngomo, A.C.N., Usbeck, R., 2021. Knowledge graph question answering using graph-pattern isomorphism. arXiv preprint arXiv:2103.06752 .

Vrandečić, D., Krötzsch, M., 2014. Wikidata: A free collaborative knowledgebase. Commun. ACM 57, 78–85. URL: https://doi.org/10.1145/2629489, doi:10.1145/2629489.

Wang, P., Shi, T., Reddy, C.K., 2020. Text-to-sql generation for question answering on electronic medical records, in: Proceedings of The Web Conference 2020, pp. 350–361.

Weikum, G., 2021. Knowledge graphs 2021: A data odyssey. Proceedings of the VLDB Endowment (PVLDB) 14, 3233–3238.

Xu, K., Reddy, S., Feng, Y., Huang, S., Zhao, D., 2016. Question answering on Freebase via relation extraction and textual evidence, in: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Berlin, Germany. pp. 2326–2336. URL: https://aclanthology.org/P16-1220, doi:10.18653/v1/P16-1220.

Yao, X., Van Durme, B., 2014. Information extraction over structured data: Question answering with Freebase, in: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Baltimore, Maryland. pp. 956–966. URL: https://aclanthology.org/P14-1090, doi:10.3115/v1/P14-1090.

Yih, W.t., Richardson, M., Meek, C., Chang, M.W., Suh, J., 2016. The value of semantic parse labeling for knowledge base question answering, in: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pp. 201–206.

Yin, X., Gromann, D., Rudolph, S., 2021. Neural machine translating from natural language to sparql. Future Generation Computer Systems 117, 510–519.

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., Zhang, Z., Radev, D., 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task, in: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Brussels, Belgium. pp. 3911–3921. URL: https://aclanthology.org/D18-1425, doi:10.18653/v1/D18-1425.

Zaremba, W., Sutskever, I., 2014. Learning to execute. arXiv preprint arXiv:1410.4615 .

Zhang, R., Yu, T., Er, H., Shim, S., Xue, E., Lin, X.V., Shi, T., Xiong, C., Socher, R., Radev, D., 2019. Editing-based SQL query generation for cross-domain context-dependent questions, in: EMNLP), pp. 5338–5349.

Zheng, S., Rao, J., Song, Y., Zhang, J., Xiao, X., Fang, E.F., Yang, Y., Niu, Z., 2021. Pharmkg: a dedicated knowledge graph benchmark for bomedical data mining. Briefings in bioinformatics 22, bbaa344.