KIF-QA: Using Off-the-shelf LLMs to Answer Simple Questions over Heterogeneous Knowledge Bases

Marcelo Machado[1,*], João Pedro Porto Campos[1], Guilherme Lima[1] and Viviane Torres da Silva[1]

[1]*IBM Research, Rio de Janeiro, Brazil*

Abstract

We present KIF-QA, a semantic parsing-based approach for answering simple questions over heterogeneous knowledge bases. KIF-QA uses off-the-shelf pre-trained large language models (LLMs) and in-context (few-shot) learning to transform questions into interpretable logical forms (queries) without requiring any fine-tuning. Because it uses KIF (the knowledge integration framework) to mediate all access to the underlying knowledge base, KIF-QA can be easily adapted to target any base accessible through KIF (which out-of-the-box includes Wikidata, DBpedia, PubChem, and others). We evaluate KIF-QA over the Wikidata and DBpedia versions of the SimpleQuestions benchmark using Llama 3.3, Llama 4 Maverick, and Mistral Medium 3. The results show competitive performance to comparable state-of-the-art methods. KIF-QA implementation is made available under an open-source license.

Keywords

knowledge base question answering, large language model, in-context learning, Wikidata, SPARQL, KIF

1. Introduction

Knowledge base question answering (KBQA) is a classical task in artificial intelligence. Given a natural language question, the goal of KBQA is to produce an answer supported by facts extracted from a knowledge base [1, 2]. In the knowledge graph version of KBQA, the base is a knowledge graph, such as Wikidata [3] or DBpedia [4], and the answers are nodes (entities) or edges (statements or triples) in the graph. For example, given the question "Where was James Brown born?", a KBQA system may retrieve and present as an answer the entity "Barnwell, SC, USA", which in Wikidata would be wd:Q586262 and in DBpedia dbr:Barnwell,_South_Carolina.

The appeal of KBQA lies in making knowledge bases accessible to non-expert users. The user can ask natural language questions directly to the system, without any prior understanding of the knowledge base schema or its query language (e.g., in the case of knowledge graphs, without having to write SPARQL [5] or Cypher [6] queries). A similar argument applies to non-human users. For instance, a retrieval augmented generation (RAG) [7] agent can use KBQA to pose natural language questions directly to a knowledge base. This way, the obtained answers are grounded and can be used as additional context to improve the output of pre-trained (large) language models ((L)LMs) [8].

KBQA systems can be broadly classified according to the type of question they are designed to handle (simple versus complex questions) and the method they use to produce the final answer (semantic parsing versus information retrieval) [2]. *Simple questions* such as "What is the capital of Brazil?" involve a single fact (or hop) in the graph, while *complex questions* involve multiple hops or complex operations (e.g., aggregation or arithmetic). In *semantic parsing*-based KBQA systems, answers are produced by first transforming the natural language question into an logical form (query) and then executing it over the graph. In contrast, in *information retrieval*-based systems, answers are obtained by searching the graph directly, without employing an explicit query.

Wikidata'25: Wikidata Workshop at ISWC 2025

*Corresponding author.

mmachado@ibm.com (M. Machado); joao.porto@ibm.com (J. P. P. Campos); guilherme.lima@ibm.com (G. Lima); vivianet@br.ibm.com (V. T. da Silva)

0000-0002-0894-9750 (M. Machado); 0009-0004-4476-443X (J. P. P. Campos); 0000-0002-7660-9256 (G. Lima); 0000-0003-4669-7299 (V. T. da Silva)

We present KIF-QA, a semantic parsing-based KBQA approach that uses off-the-shelf LLMs to answer simple questions over heterogeneous knowledge graphs.

We focus on simple questions because they are easier to work with and, once solved, can be used as a basis for solving complex questions iteratively [9, 10, 11]. Also, simple questions capture the essence of the KBQA task, serving as a baseline for testing the applicability of LLMs to semantic parsing in a training-free setting. Most methods for simple question answering, including the most successful ones [12, 13, 14, 15], involve some form of training, which makes them dependent on the availability of training datasets or couple them to the structure of the underlying graph. Our approach, on the other hand, requires no prior knowledge of the graph schema and relies entirely on in-context (few-shot) learning [16] which means it involves neither training nor fine-tuning. All that is needed is access to (i) a pre-trained language model; (ii) a standard query interface to the target knowledge graph (e.g., a SPARQL endpoint); and (iii) a search interface capable of looking up graph entities by label (e.g., a keyword-based search service).

The reason we chose to work with semantic parsing, i.e., producing an explicit query (logical form), is twofold. First, compared to information retrieval, semantic parsing is more interpretable as the produced logical form serves as an explanation for the answers [2]. Second, semantic parsing can be designed in such a way that the final logical form is obtained through successive refinements [12], starting with an abstract "draft query" and ending with a concrete query to be evaluated over the target graph. This design avoids an early commitment with the particular query language and schema of the target graph. In KIF-QA, we go one step further and delegate the problem of producing a final query and efficiently evaluating it to a completely separate layer, called KIF.

KIF [17][1] is an open-source knowledge integration framework based on Wikidata. It uses a plugin architecture to expose a unified interface to the graphs it integrates allowing us, for example, to query DBpedia [4] or PubChem (RDF) [18] as if they followed the data model (syntax) of Wikidata. KIF-QA uses KIF to mediate all access to underlying knowledge graph, which makes our approach somewhat independent of it. As long as there are KIF plugins to query and search the target graph, KIF-QA can be instantiated over it to answer simple questions.

Our contributions are as follows:

- An approach for semantic parsing that uses in-context (few-shot) learning over an off-the-shelf LLM to transform a natural language question into a set of KIF filters (queries to the KIF layer).
- An evaluation of our approach over the Wikidata and DBpedia versions of the SimpleQuestions dataset [19], namely, SimpleQuestions-Wikidata [20] and SimpleDBpediaQA [21], using Llama 3.3 (70B Instruct) [22], Llama 4 Maverick (17Bx128E), and Mistral Medium 3 (20.25). The results indicate that our approach achieves competitive performance against the state-of-the-art for simple questions while requiring no training or fine-tuning and being easily adaptable to work with any knowledge base accessible via KIF.
- An open-source implementation of our approach together with the datasets used to evaluate it, which should ease the replication of our experiments.[2]

The rest of the paper is organized as follows. Section 2 presents some background on semantic parsing-based KBQA and KIF. Sections 3 and 4 present KIF-QA and its experimental evaluation. Section 5 presents our conclusions and future work.

2. Background and Related Works

2.1. Semantic Parsing-based KBQA and LLMs

In semantic parsing-based KBQA, the system answers a natural language question by first translating it into a logical form (query) and then executing the logical form over a knowledge base [2]. A standard

[1] https://github.com/IBM/kif
[2] https://github.com/IBM/kif-llm

semantic parsing system typically performs three tasks: entity linking, relation linking, and query generation. *Entity linking* and *relation linking* are the processes of identifying and linking mentions of entities and relations in the question to corresponding entities and relations in knowledge base. These tasks are sometimes referred to as alignment or grounding tasks and may involve sub-tasks such as named entity recognition, candidate generation, and disambiguation. *Query generation* is the process of generating an executable query capturing the intent of the question.

Some semantic parsing systems follow an *end-to-end* design in the sense that they perform the three tasks at once, using a single neural network model to generate a query directly from the question. A recent example of such a system is SPARKLE [15] which, given a question, employs constrained decoding within a single sequence-to-sequence model (a fine-tuned BART [23]) to generate a corresponding SPARQL [5] query. It is more common for semantic parsing-based KBQA systems to adopt a *multi-stage* design though. In multi-stage systems, entity linking, relation linking, and query generation are performed in stages, one after the other, and are sometimes delegated to specialized tools, such as the popular entity linker BLINK [24].

As we discuss in detail in Section 3, our system, KIF-QA, follows a multi-stage design. For entity and relation linking, KIF-QA first uses KIF [17] to obtain candidate entities and relations in the namespace of the target graph. Then it employs an off-the-shelf pre-trained (large) language model ((L)LM) with in-context learning for disambiguation (selecting the best candidates). KIF-QA also uses in-context (few-shot) learning to instruct the same LLM to generate a draft logical form from the natural language question. Finally, this draft logical form together with the top-ranked candidate entities and relations are used to generate the final queries (in our case, KIF queries).

Many multi-stage systems also adopt pre-trained language models for query generation combined with independent entity and relation linking modules. For instance, STaG-QA [12] uses a pre-trained language model to generate a skeleton SPARQL query which is refined by resolving entities using BLINK [24]. GETT-QA [13] uses a fine-tuned text-to-text model (T5 [25]) to generate a skeleton SPARQL query which is refined by linking entities via a BM-25-based label search over the graph. Other multi-stage systems that employ pre-trained language models for generating SPARQL are [10, 26, 27]. One of the few multi-stage systems that uses an LLM in a training-free setting (in-context learning) for semantic parsing is KB-Binder [28]. KB-Binder uses examples (few-shot) from a training dataset when available and disambiguate entities using a lexicon-based similarity search over the whole graph.

Three things distinguish KIF-QA from the aforementioned multi-stage systems. First, while KIF-QA uses off-the-shelf models and in-context (few-shot) learning, with the exception of KB-Binder, all of the other systems require some form of training or fine-tuning. Second, while KIF-QA can be adapted to work with any target graph (as long as it can be queried and searched through KIF), most of the aforementioned systems target a particular graph or depend on direct access to the raw graph data (e.g., to construct custom indexes). Third, while KIF-QA delegates to KIF the problem of efficiently executing the generated query (e.g., the handling of pagination, parallelization of disjoint subqueries, SPARQL dialect-specific optimizations), the aforementioned systems focus on the query generation task and mostly ignore the pragmatics of SPARQL query execution in the real world.

2.2. KIF

KIF [17] is an open-source knowledge integration framework based on Wikidata [3]. It adopts the data model[3] and vocabulary of Wikidata as a *lingua franca* for integrating heterogeneous knowledge sources. The integration done by KIF is *virtual* in the sense that the data model and vocabulary translations happen at query time, guided by used-provided mappings.

KIF is in essence a Python library (kif-lib[4]) for evaluating queries over *engines* and obtaining objects of the Wikidata data model as a result. KIF engines are implemented following a plugin architecture. There are various predefined (built-in) engine plugins which can be extended by applications and loaded dynamically into the library. The two basic types of KIF engines are stores and searchers.

[3] https://www.wikidata.org/wiki/Wikidata:Data_model
[4] https://pypi.org/project/kif-lib/

Store. A KIF *store* is an interface to a knowledge source, typically but not necessarily a knowledge graph. The most common type of KIF store is the SPARQL store, which reads Wikidata-like statements from a given SPARQL endpoint. (SPARQL [5] is the query language of RDF [29], a standard format for representing graph data.) To create a SPARQL store, one needs specify the address of the target SPARQL endpoint and the SPARQL mappings to be used to generate a concrete SPARQL query (i.e., one adhering to the RDF encoding of the target graph). Besides mappings for Wikidata itself, KIF comes with SPARQL mappings for DBpedia [4], PubChem (RDF) [18], UniProt [30], among others. New mappings can be added programmatically.

We can use the KIF command-line interface (CLI), which is distributed together with the library, to run a simple query over a KIF store. For instance:

```
1 $ kif filter --subject='wd.Q(7251)' --property=wd.doctoral_advisor
2 (Statement (Item Alan Turing) (ValueSnak (Property doctoral advisor) (Item Alonzo Church)))
```

The command on line 1 runs a KIF *filter* (i.e., a statement match query) using the default store plugin, `wdqs`, which targets the public Wikidata Query Service (WDQS) and uses KIF's Wikidata SPARQL mappings. This particular filter instructs KIF to search for statements whose subject is wd:Q7251 (Alan Turing) and property is wd:P184 (doctoral advisor). To evaluate a filter, the SPARQL store uses the provided mappings to compile it into a SPARQL query and then runs the query over the target endpoint.[5] The resulting bindings are used to construct Wikidata-like statements. The returned statement in this case is shown on line 2 and stands for the assertion "Alan Turing's doctoral advisor is Alonzo Church".

In the Wikidata data model, which is adopted by KIF, every statement consists of a subject (which can be an item or property) and a *snak*. The subject of the above statement is the item Alan Turing and its snak is a *value snak* (i.e., property-value pair) with property equal to doctoral advisor and value equal to the item Alonzo Church.

KIF distinguishes between unannotated statements, such as the one above, and annotated statements. The latter, besides a subject and a snak, also carry a set of qualifiers, a set of reference records, and a rank. For instance, the filter below matches *annotated statements* (due to the `--annotated` flag) with property wd:P2054 (density) and value equal to 0.88 grams per cubic centimeter:

```
3  $ kif filter --property=wd.density --value='Quantity("0.88", wd.Q(13147228))' --annotated
4  (AnnotatedStatement (Item benzene) (ValueSnak (Property density) 0.88 ±0.01 g/cm³)
5    (QualifierRecord
6      (ValueSnak (Property temperature) 20 ±1 °C)
7      (ValueSnak (Property phase of matter) (Item liquid)))
8    (ReferenceRecordSet
9      (ReferenceRecord
10       (ValueSnak (Property HSDB ID) "35#section=TSCA-Test-Submissions")
11       (ValueSnak (Property stated in) (Item Hazardous Substances Data Bank))))
12   NormalRank)
13 ⋮
```

The annotated statement shown on lines 4–12 asserts that "benzene's density is 0.88±0.01 g/cm³". Its qualifier record (lines 5–7) qualifies this assertion, i.e., says in addition that this is the case when "the temperature is 20±1 °C" and "the phase of matter is liquid". Its reference record set (lines 8–11) contains a single reference record (lines 9–11) which indicates the provenance of the statement, namely, the entry with the given HSDB ID in the Hazardous Substances Data Bank. Finally, the value NormalRank for its rank (line 12) is the default one and means "neutral" (neither preferred nor deprecated).

More complex filters. So far, we have used simple values to match the components of statements in filters. In KIF, we can also specify the desired subject, property, or value indirectly, by giving a *constraint* that it must satisfy. For instance:

[5] KIF handles query pagination and parallelization automatically. It also comes with an `asyncio`-compliant *async* API which can be used to run queries asynchronously (without blocking waiting on their results).

```
14 $ kif filter
    ↪ --subject='(wd.shares_border_with(wd.Argentina)|wd.shares_border_with(wd.Paraguay)) &
    ↪ wd.located_in_or_next_to_body_of_water(wd.Atlantic_Ocean)'
    ↪ --property='wd.official_language'
15 (Statement (Item Brazil) (ValueSnak (Property official language) (Item Portuguese)))
16 (Statement (Item Uruguay) (ValueSnak (Property official language) (Item Spanish)))
```

This filter matches statements such that (i) the subject "shares border with Argentina *or* Paraguay" *and* "is located in or next to the Atlantic Ocean"; and (ii) the property is wd:P37 (official language). The only matching subjects in this case are wd:Q155 (Brazil) and wd:Q77 (Uruguay) and their official languages, as indicated by the statements above, are wd:Q5146 (Portuguese) and wd:Q1321 (Spanish).

Besides single-hop checks, KIF supports constraints involving n-hop property paths. These can be combined via conjunction (`&`) and disjunction (`|`) operators, as shown above. Simple datatype checks and projection on the statement components (subject, property, value) are also supported.

Targeting other graphs. To query a graph other than Wikidata, all we need to do is change the store plugin. For instance:

```
17 $ kif filter --store=dbpedia --subject='db.r("Alan_Turing")'
    ↪ --value='(db.almaMater/db.city)(db.r("Princeton,_New_Jersey"))'
18 (Statement (Item dbr:Alan_Turing) (ValueSnak (Property dbo:doctoralAdvisor) (Item
    ↪ dbr:Alonzo_Church)))
19 (Statement (Item dbr:Alan_Turing) (ValueSnak (Property doctoral advisor) (Item
    ↪ dbr:Alonzo_Church)))
```

The `--store=dbpedia` argument instructs KIF to run the filter using the `dbpedia` store plugin, which targets the public DBpedia SPARQL endpoint and uses KIF's DBpedia SPARQL mappings. The filter in this case matches statements whose subject is dbr:Alan_Turing and value is "someone whose dbo:almaMater is located in the dbo:city of dbr:Princeton,_New_Jersey".

Despite querying DBpedia, as expected, we get as a result two Wikidata-like statements (lines 18–19) asserting that "Alan Turing's doctoral advisor is Alonzo Church". We say these statements are "Wikidata-like" because although following Wikidata syntax, their components mostly have DBpedia URLs. This happens because KIF's DBpedia SPARQL mappings do not attempt to convert entities in the DBpedia namespace into that of Wikidata. The exception are properties. If there exists a Wikidata property which is said to be the same as (`owl:sameAs`) a given DBpedia property in the DBpedia graph, then the generated SPARQL query will match it. This is why on line 19 we get a statement whose property is wd:P184 (doctoral advisor) even though we are querying DBpedia.

Searchers. Besides stores, the other kind of engine in KIF are searchers. A KIF *searcher* is an interface to a similarity search method within a namespace. For example, given a string, the `wikidata-wapi` search plugin looks in the Wikidata namespace for entities with a similar label or alias:

```
20 $ kif search --search=wikidata-wapi 'pizza' --item
21 (Item pizza)
22 (Item Mariagrazia Pizza)
23 (Item Pizza Hut)
24 ⋮
```

The `--search=wikidata-wapi` argument specifies the search plugin to be used, "pizza" is the search string, and `--item` indicates that we want to search for items. (KIF, as Wikidata, distinguishes between three types of entities: items, properties, and lexemes.) The top three items found by the search are shown on lines 21–23.

In addition to Wikidata, KIF comes with built-in plugins to search for entities in the namespaces of DBpedia and PubChem.

3. Proposal

The KIF-QA approach consists of five main steps (see Figure 1).

Step 1: Draft triple pattern generation. Given a simple question Q in natural language, we ask the LLM to generate a draft triple pattern (s, p, o) using text labels for the known components (s-p or p-o) and the variable "?x" for the unknown component (either s or o) (see Figure 2 (left)). We provide examples of "(question, draft triple pattern)" pairs to the LLM when these are available.

Step 2: Entity linking. Given a draft triple pattern (s, p, o), we perform entity-linking on its *topic entity label*, i.e., the known s or o. This is done through KIF by searching for entities with a similar label or alias in the namespace of the target graph. The result is a set of candidate entities accompanied by their descriptions. We then ask the LLM to *disambiguate* the topic entity label against the candidate entities, i.e., select among the candidates those entities whose label and description best match the topic entity label and the text of the question (see Figure 2 (right)). The result of this step is a refined set $\mathscr{E}$ of *candidate topic entities*.

Step 3: Relation linking. For resolving the predicate component p of the draft triple pattern (s, p, o), we proceed as follows. For each candidate topic entity E in $\mathscr{E}$, we use KIF to obtain the candidate relations R_i and R_j such that, for some x, $R_i(E, x)$ or $R_j(x, E)$ in the graph. We then ask the LLM to disambiguate the predicate label p against the label and description of each candidate relation R_k, the label and description of E, and the text of question Q. The result of this step is, for each topic entity E, a refined set $\mathscr{R}_E$ of *candidate relations for* E. This set is constructed in a way that, for each R_k in $\mathscr{R}_E$, we can always determine whether E occurs as source or target of R_k in the graph.

Steps 4 and 5: Filter generation and evaluation. For each candidate topic entity E in $\mathscr{E}$ and candidate relation R_k in $\mathscr{R}_E$, we generate a KIF filter $f(E, R_k, \_)$ if E occurs as a source in R_k and a KIF filter $f(\_, R_k, E)$ if E occurs as a target in R_k. We then execute all filters generated for all candidate entities in parallel. The resulting statement set $\mathscr{A}$ is the *answer set* associated to the original natural language question Q.

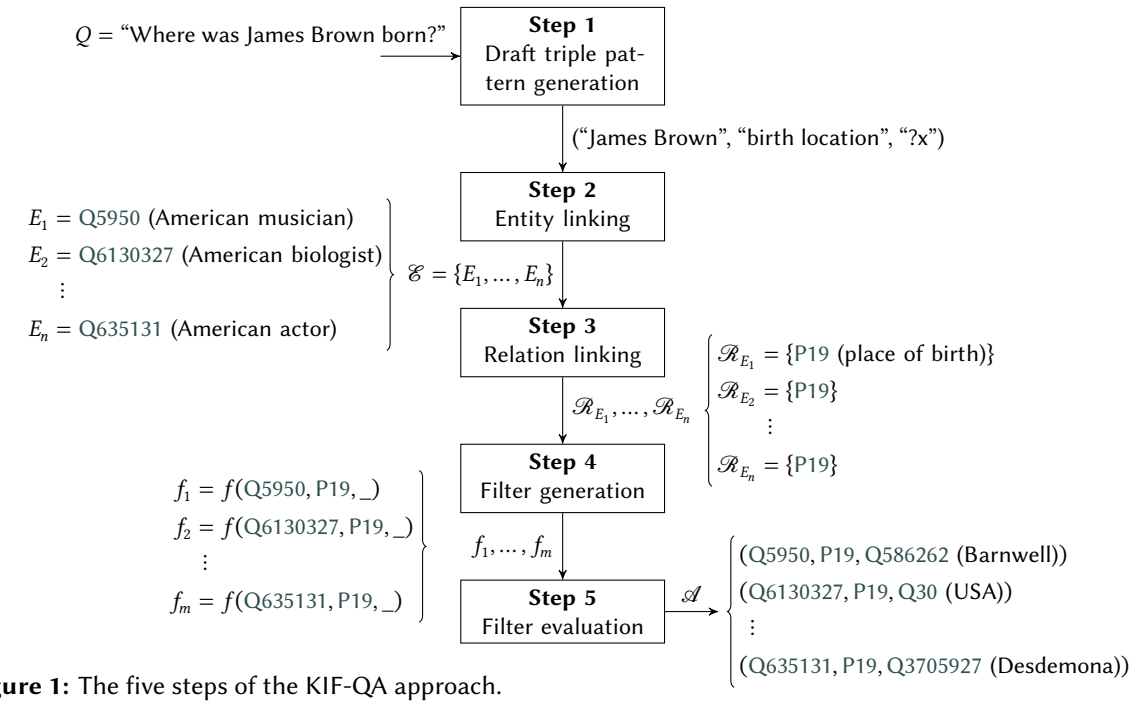

Figure 1: The five steps of the KIF-QA approach.

A few remarks about the steps are in order.

Step 1. KIF-QA assumes that the input question is a simple question whose corresponding draft triple pattern has a single unknown component, either the subject or the object. This is a standard assumption in simple-question KBQA [2]. (Some systems and benchmarks adopt an even more restrictive assumption, namely, that the unknown component is always the object [19, 31, 32].)

When asking the LLM to transform the question into a draft triple pattern (see Figure 2 (left)), KIF-QA can be provided with examples. These are question-draft triple pattern pairs, such as:

When did World War II begin?	("World War II", "begin", "?x")
Where was Freddie Mercury born?	("Freddie Mercury", "born at", "?x")
Who painted Mona Lisa?	("?x", "painted", "Mona Lisa")

The set of examples can be fixed (the same for all questions) or can be chosen from a pool of examples relevant to the question at hand. In the evaluation of KIF-QA, discussed in Section 4, we compare the performance of using a fixed set of examples versus a custom set chosen from a training dataset.

Steps 2 and 3. The disambiguation of the topic entity (s or o) and the predicate (p) of the draft triple pattern (s, p, o) involves searching for entities and relations with similar labels or aliases in the namespace of the target graph. KIF-QA does this through KIF by using an appropriate *search plugin*. For instance, for searching in the Wikidata entity namespace (http://www.wikidata.org/entity/) KIF-QA uses the built-in KIF search plugin `wikidata-wapi`, which calls Wikidata's public Wikibase API[6]. Similarly, for searching in the DBpedia resource (http://dbpedia.org/resource/) and ontology (http://dbpedia.org/ontology/) namespaces, KIF-QA uses the search plugins `dbpedia-lookup` and `dbpedia-ddgs`. The former calls DBpedia's public Lookup API[7] to search for DBpedia resources, while the latter uses the meta-search Python library DDGS[8] to search in the DBpedia website for URLs matching those of DBpedia ontology classes and properties. To make the disambiguation step work with a different graph, all one needs to do is use a different KIF search plugin (one that supports searches in the namespace of the target graph).

After obtaining the candidate entities and relations, KIF-QA asks the LLM to pick the candidates that best match the question (see Figure 2 (right)). A similar method is used in [33] to disambiguate candidate entities within a certain textual context. Note that we could use a different method here. A popular alternative is to use some embedding distance metric, such as [34], to compare the labels, aliases, or descriptions of the candidates entities and relations to the text of the question.

In Step 3, the candidate relations for a given topic entity E are obtained by querying the graph for statements (edges) whose source or target entity is E. This is done through KIF by using an appropriate *store plugin*. For querying Wikidata and DBpedia, KIF-QA uses the store plugins `wdqs` and `dbpedia` which evaluate queries over the public Wikidata Query Service (WDQS)[9] and the public DBpedia SPARQL endpoint[10], respectively. The query in this case is the union of the KIF filters

`kb.filter_p(subject=`E`)` and `kb.filter_p(value=`E`))`

where `kb` is the target KIF store. The call `kb.filter_p()` looks for statements in `kb` matching the given pattern (i.e., statements with subject E or value E) and selects (projects) their property component, hence the suffix "_p". By using a KIF here, instead of SPARQL, we delegate to KIF both the problem of generating a concrete SPARQL query (i.e., one matching the RDF encoding of the target graph) and the problem of executing it efficiently (see Section 2.2). As before, to target a different graph, all one needs to do is instruct KIF-QA to use a different KIF store plugin—the filters remain the same.

[6] https://www.wikidata.org/w/api.php
[7] https://lookup.dbpedia.org/
[8] https://github.com/deedy5/ddgs
[9] https://query.wikidata.org/sparql
[10] https://dbpedia.org/sparql

[SYSTEM]
You are responsible for recognizing incomplete subject-predicate-object triple patterns from simple natural language questions.

- Subjects are entities (e.g., people, organizations, locations), and objects can be either entities or literals (e.g., dates, numbers).
- The predicate describes the relationship between the subject and the object.
- Each triple must have exactly one unknown element, represented as "?x".
- Return a Python list of dictionaries, where each dictionary contains exactly one triple, represented as three string values under the keys "subject", "predicate", and "object".
- Your output must adhere to valid Python syntax. Do not include any extra explanations or text.

(Examples go here)

[USER]
Where was James Brown born?

[SYSTEM]
You are a precise entity-linking assistant. Your task is to select the ID of the candidate that unambiguously matches the target term in the sentence, ensuring factual accuracy and semantic coherence. Rules:

1. Strict Matching: Only return a candidate ID if the context in the sentence explicitly aligns with the candidate's description.
2. No Guessing: Do not infer or assume missing information. If the sentence lacks sufficient context, return nothing.
3. Output Format:
   - If there is a match, your response should be a list of comma separated IDs.
   - If there is no clear match respond an empty string. Do not include any extra explanations.

(Examples go here)

[USER]
Now follow the format strictly:
Sentence: Where was James Brown born?
Term: "James Brown"
Candidates:

- ID: C1, Label: James Brown, Description: American musician
- ID: C2, Label: James H. Brown, Description: American Biologist

⋮

- ID: C*n*, Label: James Brown, Description: American actor

Output:

Figure 2: Prompts used by KIF-QA to answer the question "Where was James Brown born?". *Left*: Prompt used in Step 1 (draft triple pattern generation). *Right*: Prompt used in Step 2 (entity linking). The prompt used in Step 3 (relation linking) is similar to the latter.

Steps 4 and 5. The same KIF store `kb` that was used in Step 3 to obtain the candidate relations is used in Step 5 to execute (in parallel) the filters generated in Step 4. That is, the answer set $\mathscr{A}$ is obtained by computing, for each E in $\mathscr{E}$ and R in $\mathscr{R}_E$, the union of the following filters:

$$\begin{cases} \texttt{kb.filter(subject=}E\texttt{, property=}R\texttt{)} & \text{if } E \text{ is a source of } R \text{ in the graph} \\ \texttt{kb.filter(property=}R\texttt{, value=}E\texttt{)} & \text{if } E \text{ is a target of } R \text{ in the graph} \end{cases}$$

Because we use the call `kb.filter` above, $\mathscr{A}$ will contain KIF statements. These are objects consisting of a subject (item or property) and *snak*, i.e., a pair property-value where the value is an entity, shallow data value (IRI, text, external id, or IRI), or deep data value (quantity or time).

We could have used `kb.filter_annotated` in place of `kb.filter` above to obtain statements together with their qualifiers, references, and rank (see Section 2.2). (In the case DBpedia, the default built-in SPARQL mappings always set the rank to "normal" and do not generate qualifiers or references.) Another possibility is using `kb.filter_s` (projection on subject) or `kb.filter_v` (projection on value) in place of `kb.filter` to obtain just the unknown entity or value instead of whole statements.

Note that because we use KIF we avoid the trouble of having to know how "statements" (which are not necessarily RDF triples) and "entities" or "values" (which are not necessarily single RDF terms) are represented in the RDF encoding of the target graph. We also do not need to worry about writing a SPARQL query that will run efficiently on a particular triplestore. Wikidata's public query service uses Blazegraph[11] while DBpedia's uses Virtuoso RDF[12]. Although these two triplestores claim to be standards compliant, in practice, they accept different dialects of SPARQL—subtle changes in the query can affect the performance in unexpected ways. For instance, when targeting Blazegraph, the KIF's SPARQL compiler uses Blazegraph's named subquery extension to enforce an evaluation order in subqueries, which greatly speeds up some types of queries.

[11] https://blazegraph.com/
[12] https://virtuoso.openlinksw.com/

4. Evaluation

4.1. Dataset

To evaluate KIF-QA, we use the Wikidata and DBpedia ports of the SimpleQuestions dataset [19]. The original SimpleQuestions dataset targeted Freebase [35] and consisted of simple questions paired with the single triple pattern that answered them over Freebase. The Wikidata port of SimpleQuestions is called SimpleQuestions-Wikidata [20] and the DBpedia port is called SimpleDBpediaQA [21]. For our evaluation, we selected 1,238 questions from the test subset of the answerable version of SimpleQuestions-Wikidata and 1,222 from the test subset of SimpleDBpediaQA.[13]

A known issue of the SimpleQuestions dataset is ambiguity, which stems from the method used to create it [31]. Annotators were shown a single Freebase triple such as "(James Brown, birth place, Beaumont)" and asked to write a simple question, e.g., "which city was james brown born". The problem here is that the link between the specific entity James Brown mentioned in the triple and the substring "james brown" in the question is lost. Just from the question, without any other context, it is impossible to determine the exact James Brown to which it refers. For instance, the subject of the gold triple for this question in SimpleQuestions-Wikidata is wd:Q6130343 (James Brown), not the famous American musician, but the American football quarterback born in 1975.

Since our focus is measuring KIF-QA's ability in providing accurate final answers, rather than solely creating logical forms, to workaround this ambiguity problem, we extended the set of gold triples in both SimpleQuestions-Wikidata and SimpleDBpediaQA to include all entities in the corresponding knowledge graph whose label or alias matched exactly the label of the gold entity. For example, in the "James Brown" case above, we included as gold entities every item in Wikidata whose label or alias matched the label of wd:Q6130343. This means that in our curated versions of SimpleQuestions-Wikidata and SimpleDBpediaQA every question is paired not with a single gold triple but with a set of one or more gold triples.

4.2. Models and Metrics

We tested three different LLMs with varying scales and architectural approaches: (i) *Llama 3.3 (70B Instruct)*[14], a large instruction-tuned 70B-parameter model with strong general language understanding, serving as a high-performance baseline for knowledge-intensive tasks; (ii) *Llama 4 Maverick (17Bx128E)*[15], a 17B-parameter multimodal model featuring a Mixture of Experts (MoE) architecture that activates 128 experts simultaneously; and (iii) *Mistral Medium 3 (20.25)*[16], a smaller, 25B-parameter model optimized for efficient inference, representing resource-constrained scenarios.

The KIF-QA architecture allows the use of different models in different steps (draft triple pattern generation, entity linking, and relation linking). But in the evaluation discussed here we used the same model in all steps of a same run.

We report both micro- and macro-averaged precision (P), recall (R), and F1 scores. Micro-averaging aggregates results over all instances, while macro-averaging computes the metrics per question and then averages them, giving equal weight to each question regardless of answer set size.

4.3. Results

Fixed-prompt baseline. The results on SimpleQuestions-Wikidata using a prompt with a fixed set of examples (manually curated and always the same) are shown in Table 1. As can be seen, all models achieved a high recall, indicating that the system retrieves many correct answers. The significant drop in precision observed in the micro-averaged results is explained by a few anomalous questions which produced a large number of false positives.

[13] https://huggingface.co/ibm-research/kif-qa
[14] https://github.com/meta-llama/llama-models/blob/main/models/llama3_3/MODEL_CARD.md
[15] https://github.com/meta-llama/llama-models/blob/main/models/llama4/MODEL_CARD.md
[16] https://mistral.ai/news/mistral-medium-3

Table 1
Results on SimpleQuestions-Wikidata using a fixed prompt.

	Micro			Macro		
Model	**P**	**R**	**F1**	**P**	**R**	**F1**
Llama 3.3	0.2460	0.8471	**0.3813**	0.8303	0.8900	**0.8396**
Llama 4 Maverick	0.2400	0.7873	0.3679	0.8128	0.8463	0.8125
Mistral Medium 3	0.0939	0.7975	0.1681	0.7899	0.8427	0.7976

Table 2
Results on SimpleQuestions-Wikidata using 5-shot retrieval.

	Micro			Macro		
Model	**P**	**R**	**F1**	**P**	**R**	**F1**
Llama 3.3	0.5979	0.8519	0.7026	0.8549	0.8988	**0.8600**
Llama 4 Maverick	0.7757	0.7598	**0.7677**	0.7840	0.7922	0.7783
Mistral Medium 3	0.4765	0.8347	0.6067	0.8330	0.8758	0.8349

Table 3
Results on SimpleQuestions-Wikidata using 5-shot retrieval (single-answer questions only).

	Micro			Macro		
Model	**P**	**R**	**F1**	**P**	**R**	**F1**
Llama 3.3	0.6385	0.9188	0.7534	0.8640	0.9188	**0.8773**
Llama 4 Maverick	0.8139	0.8054	**0.8096**	0.7877	0.8054	0.7922
Mistral Medium 3	0.5722	0.8991	0.6993	0.8570	0.8990	0.8659

Table 4
Results on SimpleDBpediaQA using 5-shot retrieval.

	Micro			Macro		
Model	**P**	**R**	**F1**	**P**	**R**	**F1**
Llama 3.3	0.5542	0.8912	0.6834	0.7781	0.8846	**0.8021**
Llama 4 Maverick	0.7518	0.8317	**0.7897**	0.7695	0.8093	0.7797
Mistral Medium 3	0.5988	0.8191	0.6918	0.7547	0.7872	0.7628

Retrieval-augmented prompting. We also tested the performance of the models using a 5-shot retrieval approach. For each question Q, before running the KIF-QA pipeline we used a Sentence Transformers model [34] to find questions similar to Q in the training dataset of SimpleQuestions-Wikidata. We then selected the top-5 most similar questions together with their verbalized gold triple patterns and used these as examples to the draft triple pattern generation step. To goal here was to improve relation disambiguation. For instance, consider the question "which city was james brown born". One of the top-ranked examples retrieved from the training dataset for this question was:

Where was charles grayson born? ("charles grayson", "place of birth", "?x")

This example should influence the LLM to generate a draft triple pattern in which "james brown" occurs in the subject and the predicate is "place of birth", which matches exactly the label of the gold relation.

Table 2 shows the results on SimpleQuestions-Wikidata using 5-shot retrieval. The dramatic improvement in the micro-averaged precision values is explained by a significant reduction in the number of false positives for all models. The improvement in the recall of Llama 3.3 and Mistral Medium 3 suggests consistent gains across diverse relation types.

Single-answer questions. In an attempt to reduce the influence of ambiguity in the results, we tested a variation of the 5-shot approach considering only single-answer questions, i.e., questions whose gold answer set consists of a single gold triple. For example, questions about specific subjects like "who published neo contra" are usually associated to exactly one triple in the graph. (Contrast this

with the question “Who was born in san diego”.) By focusing on single-answer questions, we can evaluate the model performance in a context where the influence of disambiguation errors is minimized.

The results on SimpleQuestions-Wikidata using 5-shot retrieval considering only single-answer questions are shown in Table 3. As can be seen, all models maintained a high recall, with Llama 4 Maverick achieving the highest micro precision and Llama 3.3 leading in macro F1. These results suggest that less ambiguous questions tend to generate fewer false positives. In other words, when the question is associated with a small number of triples in the knowledge base, the models seem to be better at both identifying the correct answer and avoiding extraneous or incorrect ones. This suggests that question ambiguity plays a significant role in KBQA performance and that precision gains can be achieved by narrowing the scope of candidate answers.

DBpedia. Finally, to demonstrate the generality KIF-QA across knowledge graphs, we tested it on SimpleDBpediaQA using a 5-shot retrieval configuration. The results are shown in Table 4. As can be seen, Llama 4 Maverick achieved the highest micro F1 and Llama 3.3 the highest macro F1, while Mistral Medium 3 showed competitive results. Overall the numbers obtained on SimpleDBpediaQA are slightly lower than those obtained on SimpleQuestions-Wikidata using a similar 5-shot configuration.

4.4. Classes of Failures

In light of the above results, we sought to identify *classes of failures* by analyzing cases in which KIF-QA failed by either producing wrong answers or no answer at all.

The first class of failures consists of those caused not by a problem in KIF-QA but by issues in the SimpleQuestions-Wikidata and SimpleDBpediaQA datasets. For example, the question “where was what play did born” is agrammatical and its associated gold triple is completely unrelated:

(“Gabrielle Roy”, “place of birth”, “?x”)

The second class of failures consists of those caused by unconstrained logical forms. These are usually generated when KIF-QA drops a constraint from the question at the first step of the pipeline (draft triple pattern generation). For instance, consider the question “who is someone famous with attention deficit hyperactivity disorder”. Using Llama 3.3 and given this question, KIF-QA generates the following draft triple pattern:

(“?x”, “diagnosed with”, “Attention Deficit Hyperactivity Disorder”)

This pattern does not capture the constraint that “?x” must be someone famous. Because of that, when evaluated over the knowledge base, the corresponding query will produce a large number of false positives.

Related to this issue, we observed that in many cases Llama 4 Maverick and Mistral Medium 3 failed to generate the draft triple pattern precisely because they tried to constrain it using more than one triple pattern. For instance, for the question “What swedish city was gerard de geer born in?”, Mistral Medium 3 generated two draft triple patterns:

(“Gerard De Geer”, “place of birth”, “?x”) and (“?x”, “country”, “Sweden”)

Note that this a clear indication that in-context (few-shot) learning could be used to instruct these models to tackle questions requiring more than one hop.

Failures of the third class are those caused by model hallucinations. For example, Llama 3.3 misrecognized “fre**e**kstyle” as “fre**a**kstyle” in the question “who published the game freekstyle”, making it impossible to retrieve the correct entity from the knowledge base.

Finally, failures of the fourth class are those caused by problems in entity or relation linking. In some cases, the search returned no candidate for the topic entity label of the question, while in other cases the returned candidates had labels which were unrelated to that of the topic entity. For instance, for the question “where is sheila larken from”, the search returned the candidate entity wd:Q3481761, whose

label is “Marteen Huell” and description is “American television actress”. Because of the mismatch of surface forms (“sheila larken” versus “Marteen Huell”) the LLM failed to disambiguate the topic entity. One way to mitigate this problem is by using not only the labels and descriptions of the candidate entities but also their aliases in the disambiguation step.

4.5. Remarks

The results presented in this section show that even without fine-tuning KIF-QA achieves a competitive performance on the SimpleQuestions-Wikidata and SimpleDBpediaQA datasets, which demonstrates the flexibility and robustness of its modular, prompt-based design. This robustness is notable given the inconsistencies, missing or mislabeled entities, and unconstrained questions that make 100% accuracy in these datasets unattainable. Many of the failures we discussed above stem from factors external to the LLMs, which suggests that simple extensions like handling constraints and employing aliases in disambiguation could yield immediate gains.

The results also demonstrate that performance is improved when training data is incorporated in the draft triple pattern generation step. High recall values were obtained consistently in different settings. This suggests that the approach is particularly suitable for applications where coverage is more important than perfect precision, i.e., applications such as exploratory search, knowledge discovery, or open-domain question answering. Finally, the competitive performance with smaller models like Mistral Medium 3 shows that KIF-QA can be employed effectively even in resource-constrained environments, broadening its applicability to real-world scenarios where efficiency and scalability are essential.

5. Conclusion

In this paper, we presented a semantic parsing-based KBQA approach, called KIF-QA, that uses off-the-shelf LLMs to answer simple question over heterogeneous knowledge graphs. Our evaluations showed that it is possible to obtain competitive performance (comparable to the state-of-the-art) by employing only in-context (few-shot) learning, without any fine-tuning. Because in the design of KIF-QA we separate the knowledge base-independent part of semantic-parsing, namely, generating the draft logical form (step 1 of our pipeline), from the entity linking, relation linking, and query generation parts (steps 2–4), and because we use KIF (the knowledge integration framework) to mediate all access to the underlying knowledge base, our approach can be easily adapted to work with any knowledge base accessible through KIF. This flexibility is demonstrated by our evaluations of the Wikidata and DBpedia versions of SimpleQuestions in which the change from Wikidata to DBpedia involved only instantiating KIF-QA with a different set of KIF plugins.

In future work, we will explore extensions of the approach to simple questions involving constraints, such as “What Italian dish includes cheese as an ingredient?”. To answer questions such as this one we need to be able to constraint the subject to “Italian dishes”. This type of constraint is already supported by KIF filters and our informal experiments have shown that it is possible to use in-context learning to instruct LLMs to generate them.

Declaration on Generative AI

The authors have not used any generative AI tools to write this paper.

References

[1] J. Berant, A. Chou, R. Frostig, P. Liang, Semantic parsing on Freebase from question-answer pairs, in: Proc. 2013 Conf. Empirical Methods in Natural Language Processing, Seattle, Washington, USA, 18–21 October, 2013, ACL, 2013, pp. 1533–1544.

[2] Y. Lan, G. He, J. Jiang, J. Jiang, W. X. Zhao, J.-R. Wen, Complex knowledge base question answering: A survey, IEEE Trans. Know. Data Eng. 35 (2023) 11196–11215. doi:10.1109/TKDE.2022.3223858.

[3] D. Vrandečić, M. Krötzsch, Wikidata: A free collaborative knowledgebase, Commun. ACM 57 (2014) 78–85. doi:10.1145/2629489.

[4] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, C. Bizer, DBpedia - a large-scale, multilingual knowledge base extracted from Wikipedia, Semant. Web 6 (2015) 167–195. doi:10.3233/SW-140134.

[5] W3C SPARQL Working Group, SPARQL 1.1 Overview, W3C Recommendation, W3C, 2013. URL: http://www.w3.org/TR/2013/REC-sparql11-overview-20130321/.

[6] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, A. Taylor, Cypher: An evolving query language for property graphs, in: Proc. 2018 Int. Conf. Management of Data, SIGMOD '18, ACM, 2018, pp. 1433–1445. doi:10.1145/3183713.3190657.

[7] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. tau Yih, T. Rocktäschel, S. Riedel, D. Kiela, Retrieval-augmented generation for knowledge-intensive NLP tasks, in: Proc. 34th Int. Conf. Neural Information Processing Systems, NIPS '20, Curran Associates Inc., 2020.

[8] W. Fan, Y. Ding, L. Ning, S. Wang, H. Li, D. Yin, T.-S. Chua, Q. Li, A survey on RAG meeting LLMs: Towards retrieval-augmented large language models, in: Proc. 30th ACM SIGKDD Conf. Knowledge Discovery and Data Mining, KDD '24, ACM, 2024, pp. 6491–6501. doi:10.1145/3637528.3671470.

[9] W. Xiong, X. L. Li, S. Iyer, J. Du, P. Lewis, W. Y. Wang, Y. Mehdad, W. tau Yih, S. Riedel, D. Kiela, B. Oğuz, Answering complex open-domain questions with multi-hop dense retrieval, 2021. arXiv:2009.12756.

[10] X. Huang, S. Cheng, Y. Shu, Y. Bao, Y. Qu, Question decomposition tree for answering complex questions over knowledge bases, in: Proc. AAAI Conf. Artificial Intelligence, volume 37, 2023, pp. 12924–12932. doi:10.1609/aaai.v37i11.26519.

[11] H. Rajabzadeh, S. Wang, H. J. Kwon, B. Liu, Multimodal multi-hop question answering through a conversation between tools and efficiently finetuned large language models, 2023. arXiv:2309.08922.

[12] S. Ravishankar, D. Thai, I. Abdelaziz, N. Mihindukulasooriya, T. Naseem, P. Kapanipathi, G. Rossiello, A. Fokoue, A two-stage approach towards generalization in knowledge base question answering, in: Findings of the ACL: EMNLP 2022, ACL, 2022, pp. 5571–5580. doi:10.18653/v1/2022.findings-emnlp.408.

[13] D. Banerjee, P. A. Nair, R. Usbeck, C. Biemann, GETT-QA: Graph embedding based T2T transformer for knowledge graph question answering, in: The Semantic Web: 20th Int. Conf., ESWC 2023, Hersonissos, Crete, Grece, May 28–June 1, 2023, Proceedings, Springer, 2023, pp. 279–297. doi:10.1007/978-3-031-33455-9.

[14] Y.-H. Chen, E. J.-L. Lu, K.-H. Cheng, Enhancing SPARQL query generation for question answering with a hybrid encoder–decoder and cross-attention model, J. Web Semant. 87 (2025) 100869. doi:10.1016/j.websem.2025.100869.

[15] J. Lee, H. Shin, SPARKLE: Enhancing SPARQL generation with direct KG integration in decoding, Expert Syst. Appl. 289 (2025) 128263. doi:10.1016/j.eswa.2025.128263.

[16] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners, in: Proc. 34th Int. Conf. Neural Information Processing Systems, NIPS '20, Curran Associates Inc., 2020.

[17] G. Lima, J. M. B. Rodrigues, M. Machado, E. Soares, S. R. Fiorini, R. Thiago, L. G. Azevedo, V. T. da Silva, R. Cerqueira, KIF: A Wikidata-based framework for integrating heterogeneous knowl-

edge sources, 2024. arXiv:2403.10304.
[18] S. Kim, J. Chen, T. Cheng, A. Gindulyte, J. He, S. He, Q. Li, B. A. Shoemaker, P. A. Thiessen, B. Yu, L. Zaslavsky, J. Zhang, E. E. Bolton, PubChem 2023 update, Nucleic Acids Res. 51 (2023) D1373–D1380. doi:10.1093/nar/gkac956.
[19] A. Bordes, N. Usunier, S. Chopra, J. Weston, Large-scale simple question answering with memory networks, 2015. arXiv:1506.02075.
[20] D. Diefenbach, T. P. Tanon, K. D. Singh, P. Maret, Question answering benchmarks for Wikidata, in: Proc. ISWC 2017 Posters, Demonstrations, and Industry Tracks co-located with 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 23–25, 2017, Springer, 2017. URL: https://ceur-ws.org/Vol-1963/paper555.pdf.
[21] M. Azmy, P. Shi, J. Lin, I. Ilyas, Farewell Freebase: Migrating the SimpleQuestions dataset to DBpedia, in: Proc. 27th Int. Conf. Computational Linguistics, ACL, 2018, pp. 2093–2103. URL: https://aclanthology.org/C18-1178/.
[22] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, G. Lample, LLaMA: Open and efficient foundation language models, 2023. arXiv:2302.13971.
[23] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, L. Zettlemoyer, BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension, in: Proc. 58th Annual Meeting of the ACL, ACL, 2020, pp. 7871–7880. doi:10.18653/v1/2020.acl-main.703.
[24] L. Wu, F. Petroni, M. Josifoski, S. Riedel, L. Zettlemoyer, Scalable zero-shot entity linking with dense entity retrieval, in: Proc. 2020 Conf. Empirical Methods in Natural Language Processing, ACL, 2020, pp. 6397–6407. doi:10.18653/v1/2020.emnlp-main.519.
[25] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu, Exploring the limits of transfer learning with a unified text-to-text transformer, J. Mach. Learn. Res. 21 (2020).
[26] R. Omar, I. Dhall, P. Kalnis, E. Mansour, A universal question-answering platform for knowledge graphs, Proc. ACM Manag. Data 1 (2023). doi:10.1145/3588911.
[27] Y. Chen, H. Li, Y. Hua, G. Qi, Formal query building with query structure prediction for complex question answering over knowledge base, in: Proc. Twenty-Ninth Int. Joint Conf. on Artificial Intelligence, IJCAI '20, 2021.
[28] T. Li, X. Ma, A. Zhuang, Y. Gu, Y. Su, W. Chen, Few-shot in-context learning on knowledge base question answering, in: Proc. 61st Annual Meeting of the ACL (Volume 1: Long Papers), ACL, 2023, pp. 6966–6980. doi:10.18653/v1/2023.acl-long.385.
[29] W3C RDF Working Group, Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Recommendation, W3C, 2014. URL: https://www.w3.org/TR/rdf10-concepts/.
[30] The UniProt Consortium, UniProt: The Universal Protein Knowledgebase in 2025, Nucleic Acids Res. 53 (2024) D609–D617. doi:10.1093/nar/gkae1010.
[31] M. Petrochuk, L. Zettlemoyer, SimpleQuestions nearly solved: A new upperbound and baseline approach, in: Proc. 2018 Conf. Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31–November 4, 2018, ACL, 2018, pp. 554–558.
[32] C. Badenes-Olmedo, O. Corcho, MuHeQA: Zero-shot question answering over multiple and heterogeneous knowledge bases, Semant. Web 15 (2024) 1547–1561. doi:10.3233/SW-233379.
[33] M. Machado, J. M. B. Rodrigues, G. Lima, S. R. Fiorini, V. T. da Silva, LLM Store: Leveraging large language models as sources of Wikidata-structured knowledge, in: Joint Proc. 2nd Workshop on Knowledge Base Construction from Pre-Trained Language Models (KBC-LM 2024) and the 3rd Challenge on Language Models for Knowledge Base Construction (LM-KBC 2024) co-located with the 23nd International Semantic Web Conference (ISWC 2024), Baltimore, USA, November 12, 2024, CEUR-WS.org, 2023. URL: https://ceur-ws.org/Vol-3853/paper6.pdf.
[34] N. Reimers, I. Gurevych, Sentence-BERT: Sentence embeddings using Siamese BERT-networks, in: Proc. 2019 Conf. Empirical Methods in Natural Language Processing, ACL, 2019.
[35] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, J. Taylor, Freebase: A collaboratively created graph

database for structuring human knowledge, in: Proc. 2008 ACM SIGMOD Int. Conf. on Management of Data, SIGMOD '08, ACM, 2008, pp. 1247–1250. doi:10.1145/1376616.1376746.