

LLM Store: Leveraging Large Language Models as Sources of Wikidata-Structured Knowledge

Marcelo Machado¹, João M. B. Rodrigues¹, Guilherme Lima¹, Sandro Rama Fiorini¹ and Viviane T. da Silva¹

¹IBM Research Brazil, Rio de Janeiro, Brazil

Abstract

Knowledge Integration Framework (KIF) is a Wikidata-based framework for integrating heterogeneous knowledge sources. These can be SPARQL endpoints, SQL endpoints, RDF files, CSV files, etc., and are represented in KIF as knowledge “stores”. A KIF store exposes a Wikidata view of the underlying knowledge source by interpreting its content as a set of Wikidata-like statements and allowing it to be queried through a simple but expressive pattern-matching interface. In this paper, we present LLM Store, a KIF store implementation that uses language models (LLMs) as knowledge sources. Instead of consulting a static knowledge base, when queried, the LLM Store uses the underlying LLM to synthesize Wikidata-like statements on-the-fly. The knowledge completion pipeline used by LLM Store can be fully customized and supports strategies that range from simple zero-shot prompts to retrieval-augment generation (RAG). This paper discusses the design and implementation of LLM Store and presents an evaluation using the test and validation datasets of LM-KBC Challenge @ ISWC 2024. We analyze the results of the evaluation in light of the results obtained by our submission to the same challenge, which was based on LLM Store and achieved a macro averaged F1-score of 91%. LLM Store is released as open-source and its code is available at <https://github.com/IBM/kif-llm-store>.

1. Introduction

Knowledge bases are at the core of many AI applications. Two key problems associated with these bases, especially when they are large and general purpose, are accuracy and completeness [1, 2]. It is notoriously hard and expensive to maintain accurate, up-to-date information covering a wide range of topics [3]. (A study from 2018 [4] estimates the cost per triple at about \$2 for manually curated triples and \$0.01 for automatically extracted ones.) Consider Wikidata [5], one of the largest publicly available knowledge bases. It has been maintained for over 10 years by a small army of volunteers and bots and currently has about 112 million entities and 1.5 billion statements. Despite all this effort and scale, it remains relatively easy to find Wikidata entity pages that are incomplete or contain misleading or false information (sometimes due to vandalism [6]).


This is why pre-trained (large) language models (LMs or, from now on, LLMs) are such a promising technology for unsupervised knowledge base construction/completion (KBC) [7]. Their appeal lies in their potential to offer a much faster and, ideally, more cost-effective

KBC-LM'24: Knowledge Base Construction from Pre-trained Language Models workshop at ISWC 2024

✉ mmachado@ibm.com (M. Machado); joao.bessa@ibm.com (J. M. B. Rodrigues); guilherme.lima@ibm.com (G. Lima); srfiorini@ibm.com (S. R. Fiorini); vivianet@br.ibm.com (V. T. da Silva)



© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

alternative to both manual curation and supervised methods [8]. Also, a particularly interesting feature of LLMs is that they can operate in real-time. That is, in the context of KBC, LLMs can be used to generate statements at the time the knowledge base is queried. This can be done either through zero- or few-shot probing or via retrieval-augmented generation (RAG) [9] by extracting the statements from some textual context (also obtained or provided in real-time).

In this paper, we propose an approach that leverages this flexibility and real-time operation capacity of LLMs to the problem of Wikidata-based KBC. Our approach is implemented in LLM Store¹, a software component that uses LLMs to generate Wikidata-like statements from user queries on-the-fly. LLM Store itself is implemented as a plugin within KIF², a Wikidata-based framework for integrating heterogeneous knowledge sources [10]. Both LLM Store and KIF are written in Python and released as open-source.

The name “LLM Store” comes from KIF’s store abstraction. In KIF, a *store* is an interface to a Wikidata view of a knowledge source. This can be a SPARQL endpoint, SQL endpoint, RDF file, CSV file, or in the case of LLM Store, a pre-trained LLM possibly associated with some textual context. The basic store operation is the *filter* call. It takes a pattern of the form `Filter(s, p, v)`, where *s* is the subject, *p* is the property, and *v* is the value, and returns all statements with the matching *s*, *p*, and *v* in the underlying knowledge source.

The important thing is that in KIF the patterns consumed and the statements produced by *all* stores must follow the syntax of Wikidata [11]. In the case of LLM Store, when a filter pattern using Wikidata-like entities and values is given, it must query the underlying LLM in real-time to generate matching Wikidata-like statements. The actual filter pattern evaluation pipeline used by LLM Store consists of three steps: (context-assisted) knowledge extraction, entity resolution, and statement construction. In the knowledge extraction step, the input pattern is converted into a prompt, potentially augmented with context information, and sent to the LLM. The model’s response is parsed and then, in the entity resolution step, named entities occurring in it are resolved against the target knowledge base—usually, but not necessarily, Wikidata. Finally, in the statement construction step, the resolved entities are used to instantiate the input pattern and construct corresponding statements following Wikidata’s syntax.

To evaluate LLM Store’s approach we compare different methods for the implementation of each of the steps of the pipeline above. We then analyze the results of these experiments in light of the results obtained by the system we submitted to the LM-KBC Challenge @ ISWC 2024 [12], which is based on LLM Store. This system [13] achieved a macro averaged F1-score of 91% in the test dataset of the challenge using the Llama3-8B-Instruct model³.

The rest of the paper is organized as follows. Section 2 gives a concise introduction to KIF and discusses relevant related work. Section 3 presents the filter evaluation pipeline used by LLM Store in detail and discusses the various customization options supported by the plugin. Section 4 presents and discusses the results of an experimental evaluation of LLM Store, including the results obtained in the LM-KBC Challenge @ ISWC 2024. Finally, Section 5 presents our conclusions and future work.

¹<https://github.com/IBM/kif-llm-store>

²KIF stands for Knowledge Integration Framework, which is completely different from Knowledge Interchange Format. It can be accessed at: <https://github.com/IBM/kif>

³<https://ai.meta.com/blog/meta-llama-3/>

2. Background

KIF [10] is a knowledge integration framework based on Wikidata. It uses Wikidata to standardize the syntax and possibly the vocabulary of the underlying knowledge sources. Users can then query the sources through a pattern language described in terms of the Wikidata data model. The integration done by KIF is *virtual*, i.e., the syntax and vocabulary translations happen at query time. Before detailing KIF itself, we have to recap the data model of Wikidata.

2.1. Wikidata Data Model

The Wikidata data model [11] consists of entities and statements about entities. Figure 1 shows the page of entity Q2270, which stands for benzene (the chemical compound). Every entity has a label, a description, and zero or more aliases. These are shown at the header of the page.

The figure shows a screenshot of the Wikidata entity page for benzene (Q2270). The page is annotated with various labels and boxes to highlight specific parts:

- label:** "benzene (Q2270)"
- description:** "hydrocarbon compound consisting of a 6-sided ring" and "C6H6 | [6]annulene | BNZ | cyclohexa-1,3,5-triene | 1,3,5-cyclohexatriene | ..."
- aliases:** "C6H6 | [6]annulene | BNZ | cyclohexa-1,3,5-triene | 1,3,5-cyclohexatriene | ..."
- Statements section:**
 - property:** "median lethal dose (LD50)"
 - value:** "4,700 ±1 milligram per kilogram"
 - qualifiers:** "route of administration" (oral administration, laboratory mouse) and "applies to taxon" (laboratory mouse)
 - rank:** "rank" (indicated by a triangle icon)
 - statement group:** "statement group" (indicated by a vertical line)
 - opened references:** A box containing a reference table:

stated in	PubChem
PubChem CID	241
language of work or name	English
title	benzene (English)
retrieved	12 April 2024
reference URL	https://www.cdc.gov/niosh-rtecs/CY155CC0.html
 - collapsed reference:** A box containing a reference table:

88 ±1 milligram per kilogram	
route of administration	intravenous infusion and defusion
applies to taxon	rabbit

Figure 1: Part of Wikidata’s entity page of benzene. (Adapted from [14].)

Below the header, comes the “Statements” section which groups the statements about the entity being described. A *statement* consists of two parts: subject and snak. The *subject* is the entity about which the statement is made. The *snak* is the statement’s claim. It associates a property with either a specific value (value snak), some unspecified value (some-value snak), or no value (no-value snak). In this paper, we are mainly concerned with value snaks. So, whenever we speak of a statement we mean one which can be decomposed into subject, property, and value. (Note that KIF supports all three kinds of snaks.)

Figure 1 depicts two statements which can be read as follows:

1. “benzene has an LD50 of 4,699–4,701 milligrams per kilogram”

2. “benzene has an LD50 of 87–89 milligrams per kilogram”

LD50 (or median lethal dose) is a toxicity unit that measures the dose of a substance that is required to kill half of the members of the tested population. Here the subject of both statements (1) and (2) is the same, “benzene” (Q2270). Their *snak* is a value *snak* (of the form property-value). The property of both is “median lethal dose (LD50)” (P2240). The value of (1) is the structured quantity “4,700 ± 1 mg/kg”, and the value of (2) is “88 ± 1 mg/kg”.

The data model of Wikidata also supports the notions of qualifiers and references associated with statements. *Qualifiers* are extra *snaks* that qualify the statement. In Figure 1, the qualifiers just below statement (1), highlighted in blue, indicate that the statement assertion holds when the route of administration is “oral” and taxon is “laboratory mouse”. *References* are sets of *snaks* that keep provenance information. Figure 1 shows two references for statement (1), highlighted in red. The first one indicates that the statement was obtained from PubChem [15] (a popular base of chemical information) on April 12, 2024, and carries additional *snaks* that identify the subject (benzene) in PubChem. The second reference contains a single *snak* pointing to an external page of the CDC (a public health agency in the US).

In KIF, qualifiers and references are treated as annotations and are manipulated through specific methods of the store API. One important use of references is to distinguish between statements produced by different stores. This is done by instructing KIF to attach a specific (unseen) reference to all statements produced by a given store. Using this technique, applications can distinguish statements produced by LLM Store from statements produced by more authoritative sources, like PubChem or Wikidata.

2.2. KIF Stores and Filters

The core abstraction of KIF is the store. A *store* is an interface to a Wikidata view of a knowledge source. The prototypical store is the SPARQL store, which exposes a Wikidata view of a SPARQL endpoint. Here is how we create a SPARQL store pointing to WDQS, the public SPARQL query service of Wikidata:

```
1 from kif_lib import Store
2 kb = Store('sparql', 'https://query.wikidata.org/sparql')
```

At line 1, we import from the namespace of KIF the `Store` constructor. At line 2, we use it to create a new SPARQL store `kb` pointing to WDQS. Not incidentally, because WDQS adopts the Wikidata encoding of RDF, it can be queried directly by the SPARQL store. If that was not the case, for instance, if we wanted to point the SPARQL store to PubChem’s query interface (which is not Wikidata-compatible), at line 2, we would have to provide a mapping object to be used to translate the queries and results to the syntax of Wikidata. (See [10] for details.)

With store `kb` created, we can now read statements from it as follows:

```
4 from kif_lib.vocabulary import wd
5 it = kb.filter(subject=wd.benzene, property=wd.median_lethal_dose)
6 print(next(it))
7 # Statement(Item(URI('...Q2270')), ValueSnak(Property(URI('...P2240')),
  ↪ QuantityDatatype(), Quantity(4700, Item(URI('...Q21091747')), 4699, 4701)))
```

The `kb.filter()` call on line 5 searches in `kb` for statements with subject “benzene” and property “median lethal dose”. More specifically, when `kb.filter()` is called its arguments are used to create a `Filter(s, p, v)` pattern, which is compiled by the store into a SPARQL query. This query is then evaluated over the target endpoint and the results are used to construct a (lazy) iterator `it` with the matching statements. At line 6, we print the first statement in `it`. The result is shown on line 7 and corresponds to the statement (1) of Figure 1.

Note that we used the vocabulary module of KIF (line 4) to avoid writing the full IRIs of Wikidata entities. That is, instead of `wd.benzene`, we could have written `wd.Q(2270)` or even `Item('http://www.wikidata.org/entity/Q2270')`. The same applies to the property `wd.median_lethal_dose`. That said, whenever possible we will use symbolic names in `wd` instead of numeric ids or IRIs.

Fingerprints (indirect ids) In the previous example, we used direct ids (`wd.benzene` and `wd.median_lethal_dose`) to identify the subject and property of the desired statements. Sometimes, however, we might need to specify an entity indirectly by giving not its id but a property it satisfies. In KIF, this can be done through a *fingerprint* (indirect id):

```
9 it = kb.filter(  
10     subject=wd.official_language(wd.Portuguese) & wd.continent(wd.South_America),  
11     value=wd.Argentina)  
12 print(next(it))  
13 # Statement(Item(URI('...Q155')), ValueSnak(Property(URI('...P47')), ItemDatatype()),  
    ↪ Item(URI('...Q414')))
```

This filter searches for statements such that: the subject’s official language is Portuguese *and* continent is South America (line 10); the property is anything (unrestricted); and the value is Argentina (line 11). The result, shown in line 13, is the statement “Brazil (Q155) shares border with (P47) Argentina (Q414)”. Notice that Brazil matches desired subject fingerprint.

As we will see later, in LLM Store fingerprints are used to constrain the universe of predicted answers to certain kinds of entities or values. For instance, we can set the value parameter of `filter()` to the fingerprint `wd.instance_of(wd.human)` to restrict LLM Store’s answers to statements whose value is a person.

2.3. Related Work

In 2019, Petroni et al. [16] proposed the LAMA dataset, a seminal benchmark designed to evaluate the ability of LLMs to retrieve relational facts using cloze-style prompts. This work spurred numerous follow-ups, including [17, 18, 19, 20, 21]. Among these, the work of Zhang et al. [18], which introduced LLMKE, bears the closest resemblance to our proposal.

LLMKE was the winner system of track 2 of the LM-KBC Challenge @ ISWC 2023 [22]. The task was to predict the value part (v) of incomplete triples of the form (s, p, v) , where the subject s is a Wikidata entity and the property p is an abstract property, such as *bandHasMember*, which may or may not correspond to a single property in Wikidata. Depending on the relation, the predicted value v might be a Wikidata entity or a data value (quantity, string, etc.).

The LLMKE system operates in two main stages: probing and disambiguation. During the probing stage, the system generates prompts for each input $s-p$ pair, guiding the LLM to produce the label of the Wikidata entity or data value that completes the relation. The disambiguation stage then maps entity labels to the ids of the correct Wikidata entity. For the challenge submission, LLMKE used rules tailored to each of the 21 abstract relations. These rules customized the prompts and the textual context used during the probing stage. In some cases, rules were also used during the disambiguation process to map specific keywords and labels to the corresponding data values and entities in Wikidata.

Although the design of LLM Store was inspired by LLMKE, in particular, by the approach described in [18], it differs in many aspects:

1. LLM Store is not a standalone system designed for one specific purpose. Instead, it is a component of a larger framework (KIF) and as such can be combined with other components (stores) and reused by different applications.
2. LLM Store is not tied to a particular LLM platform. It provides a common interface to access these platforms and comes with builtin support for IBM's Big AI Model (BAM), Hugging Face, and OpenAI. Support for new platforms can be added as needed.
3. LLM Store is not limited to a specific set of relations. It works out-of-the-box with any Wikidata relation and can handle filter patterns other than $(s, p, *)$, including patterns with fingerprints whose evaluation go beyond simple value prediction.
4. LLM Store allows users to customize every step of the evaluation pipeline. This includes the prompt templates, textual contexts, and methods of response parsing and entity resolution used.
5. LLM Store statements can have annotations and can be integrated with statements of other KIF stores. Annotations can be used to carry extra information, such as the model used to generate the statements, textual context, etc. Also, because LLM Store is a KIF store, the statements it produces can be seamlessly integrated with those produced by other KIF stores using, for example, a mixer store (see [10] for details).

3. LLM Store

LLM Store is a KIF store that uses an LLM as a knowledge source. Instead of searching for the given filter pattern in an existing knowledge base, LLM Store converts the pattern into a prompt, sends it to the underlying LLM, and converts the model's response back into one or more statements matching the pattern.

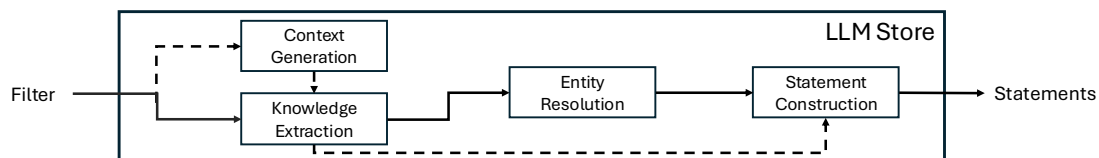


Figure 2: The LLM Store pipeline.

The LLM Store pipeline, illustrated in Figure 2, consists of three main steps: (Context-Assisted) Knowledge Extraction, Entity Resolution, and Statement Construction. The dashed arrows represent optional paths. We detail each step next.

3.1. Knowledge Extraction

The knowledge extraction step involves creating and evaluating a prompt over the underlying LLM. The prompt is created by instantiating a cloze-style prompt template from the input filter pattern. The default prompt template has the form:

```
Fill in the gap to complete the relation:  
{{subject}} {{property}} {{value}}
```

When instantiated with the filter pattern:

```
Filter(wd.Brazil, wd.official_language)
```

It produces the cloze-style prompt:

```
Fill in the gap to complete the relation:  
Brazil official language ___
```

Similarly, the same template instantiated with a filter containing a value fingerprint:

```
Filter(wd.Brazil, wd.official_language, wd.instance_of(wd.sign_language))
```

Produces the prompt:

```
Fill in the gap to complete the relation:  
Brazil official language ___ where ___ is an instance of sign language
```

The “where” above was generated from the value fingerprint in the filter and specifies a type constraint on the answers to be predicted.

In addition to describing the task, the prompt must instruct the model to format the response. The complete default prompt template is actually the following:

```
[SYSTEM]  
You are a helpful and honest assistant that resolves a TASK. Please, respond  
→ concisely, with no further explanation, and truthfully.  
  
[USER]  
TASK: Fill in the gap to complete the relation:  
{{subject}} {{property}} {{value}}  
  
The output should be only a list containing the answers, such as ["answer_1",  
→ "answer_2", ..., "answer_n"]. Do not provide any further explanation and avoid  
→ false answers. Return an empty list, such as [], if no information is available.
```

The default prompt template is accompanied by a default parsing function that parses the LLM output into a list of strings. Both the prompt template and the result-parsing function can be customized using LLM Store’s `prompt_template` and `prompt_parser` attributes.

The result of the knowledge extraction step is a list of strings matching the blanks in the query. Once such a list of strings is obtained, there are two possibilities: either (1) the list is sent to the entity resolution step to be resolved into a list of Wikidata entity ids; or (2) the list is sent directly to the statement construction step. By default, the decision is made based on the datatype of the property used in the input pattern. In Wikidata, every property has a datatype which determines the range of its possible values. If the datatype is “item” or “property”, LLM Store sends the knowledge extraction results to the entity resolution step. Otherwise, if the datatype is a data value type (string, quantity, etc.), LLM Store sends the results to the statement generation step. The default behavior can be overridden by setting the `disambiguation` attribute of LLM Store.

Textual context We just described the context-free knowledge extraction process. LLM Store also supports the use of textual contexts in the knowledge extraction step. Textual contexts tend to reduce hallucinations and are at the basis of more advanced prompting techniques, such as Retrieval-Augmented Generation (RAG) [9].

The LLM Store attribute `context`, which by default is undefined, instructs the store to use the assigned text as textual context. In practice, setting `context` makes LLM Store include in the default prompt the system instruction “Use the CONTEXT to support the answer” and the following text under the “[USER]” section:

```
CONTEXT: {{context}}
```

For other attributes related to textual contexts, see LLM Store’s documentation.

Context generation Coming up with textual contexts or obtaining them manually quickly becomes impractical. Because of that, LLM Store comes with a context generator to obtain a textual context for the input filter pattern automatically. The context generator is based on Wikidata—more specifically, on the site-links and external ids sections of Wikidata pages, which contain links to Wikipedia and other external sources describing the pages’ subjects.

Given a Wikidata entity Q described in the pattern (usually, but not necessarily, the subject), the context generator (1) retrieves all site-links and external ids from Q ’s Wikidata page; (2) passes these links through a series of scraping plugins to extract text chunks from the target HTML pages; and (3) ranks the extracted chunks by embedding similarity with a textual version of the input pattern. The best-ranked chunk is then used as context in the knowledge extraction step. Note that the context generator is disabled in the default configuration.

3.2. Entity Resolution

The goal of the entity resolution step is to resolve the input entity labels into Wikidata entity ids. For instance, given the label “Brazilian Portuguese” it should produce the item id `Q750553`, which identifies Brazilian Portuguese in Wikidata.

LLM Store currently supports three entity resolution methods. Given an input label, the three methods use Wikidata REST API’s search function to obtain a list C of candidate entities plus their descriptions. The three methods differ only in how C is processed. The first method,

baseline, is the default one; it simply picks the first entity in C . The second method, *similarity-based*, ranks the descriptions by embedding similarity with the task (and context) text and picks the entity with the best ranked description. The third method, *LLM-based*, is similar to the second but instead of computing embedding similarity, it uses an LLM to rank the descriptions, by asking which description best matches the task (and context) text. The resolution method to be used can be customized via LLM Store’s `disambiguation_method` attribute.

3.3. Statement Construction

The last step of LLM Store’s pipeline is statement construction. This step takes a list of Wikidata entities or data values as input and instantiates each of these in the input filter pattern, one at a time. The result is a sequence of output KIF statements. For instance, for a filter pattern `Filter(wd.Brazil, wd.shares_border_with)`, in which the value part is missing, and a list of entities `[wd.Argentina, wd.Paraguay, wd.Uruguay]`, the statement construction step outputs three KIF statements: (1) “Brazil shares border with Argentina”; (2) “Brazil shares border with Paraguay”; (3) “Brazil shares border with Uruguay”.

4. Evaluation

To evaluate LLM Store, we used the validation and test datasets of the LM-KBC Challenge @ ISWC 2024 [12]. As in the 2023 edition, the datasets of the 2024 challenge consist of incomplete (s, p, v) triples with the value part (v) missing. The task is to predict, for each incomplete triple, zero or more values v which, depending on the relation, can be Wikidata entities or data values (numeric quantities, strings, etc.). The subjects s are Wikidata entities and the properties p are abstract relations that may or may not correspond to a single Wikidata relation. In the 2024 edition of the challenge, only five abstract relations are considered (see Table 1).

For the evaluation, we converted each subject-property pair (s, p) in the datasets into a corresponding KIF filter pattern. For instance, the pair $(Q155, \textit{countryLandBordersCountry})$ was converted into the pattern

```
Filter(wd.Q(155), wd.shares_border_with, wd.instance_of(wd.country_))
```

(Notice the use of a value fingerprint here to restrict the results to things which are countries; the rationale is that, by itself, Wikidata’s `wd.shares_border_with` does not capture the precise meaning of the abstraction relation *countryLandBordersCountry*.) Table 1 shows the filter pattern associated with each abstract relation of the 2024 challenge.

Table 1

Mapping between abstract relations, Wikidata properties, and KIF filter patterns.

Abstract Relation	Wikidata Property	Filter Pattern
<i>awardWonBy</i>	winner (P1346)	<code>Filter(s, wd.P(1346))</code>
<i>companyTradesAtStockExchange</i>	stock exchange (P414)	<code>Filter(s, wd.P(414))</code>
<i>countryLandBordersCountry</i>	shares border with (P47)	<code>Filter(s, wd.P(47), wd.instance_of(wd.country_))</code>
<i>personHasCityOfDeath</i>	place of death (P20)	<code>Filter(s, wd.P(20), wd.instance_of(wd.city))</code>
<i>seriesHasNumberOfEpisodes</i>	number of episodes (P1113)	<code>Filter(s, wd.P(1113))</code>

With the entries of the datasets converted to KIF filters, we then proceeded to evaluate each filter using LLM Store. We tested four different configurations:

Triple LLM Store with the default prompt template and no textual context (see Section 3).

Triple-Context LLM Store with the default prompt template and a custom textual context obtained using the context generator. In this case, the context generator was configured as in our submission to the LM-KBC Challenge @ ISWC 2024, i.e., with specific scraping plugins for each abstract relation. For instance, for *awardWonBy*, we picked the *ner-extract* plugin which searches for entity names in the scraped text. (See [13] for the precise configuration used for each relation.)

Question LLM Store with a custom question-based prompt template and no textual context. By a custom question-based template, we mean one in which the task is framed as a question about the subject. For instance, for the abstract relation *seriesHasNumberOfEpisodes*, instead of the default template, obtained from the filter pattern, we used the question template “How many episodes does the series {{subject}} have?”.

Question-Context LLM Store with the question-based prompt template of Question and the custom textual context of Triple-Context. We used this configuration in our submission [13] to the LM-KBC Challenge @ ISWC 2024.

The LM-KBC’24 Challenge limits model size to 10 billion parameters, ensuring that no participating team can outperform others by monetary investment. Therefore, first, we used the Llama3-8B-Instruct model to evaluate both the validation and test datasets (the latter being the final result submitted to the challenge in [13]). We then assessed the validation dataset with a larger model (Llama3-70B-Instruct) to showcase our solution’s capability to handle different models and to compare the results with the smaller one. Both models are accessed through the IBM Big AI Model (BAM) platform.⁴ However, the results should remain consistent if using the same models from Hugging Face or platforms like Ollama⁵ (LLM Store supports both, with the *llm_name* parameter set to *hf* or *rest*, respectively). Table 2 summarizes the results obtained by each of the four configurations using: (1) Llama3-8B-Instruct over the validation and test datasets; and (2) Llama3-70B-Instruct over the validation dataset.

Runtime Figure 3 depicts the average execution time for each of the relations. We highlight the context generation time within the total execution time, as this is a key feature of the pipeline. The execution time for the *awardWonBy* relation is significantly higher than for the other relations, as the process was divided into several prompting stages. In general, context generation is the most time-consuming part. However, for the *countryLandBordersCountry* relation, the average context generation time was close to zero, due to the use of a cache. We also mitigated the execution time of the context generation process by caching the content of the accessed site-links. This solution is especially beneficial when the system is used with a repetitive set of entities.

⁴BAM is built by IBM Research as a test bed and incubator for helping accelerate generative AI research and its transition into IBM products: <https://bam.res.ibm.com/>

⁵Ollama is an open-source tool for running, creating, and sharing LLMs on your computer: <https://ollama.com/>

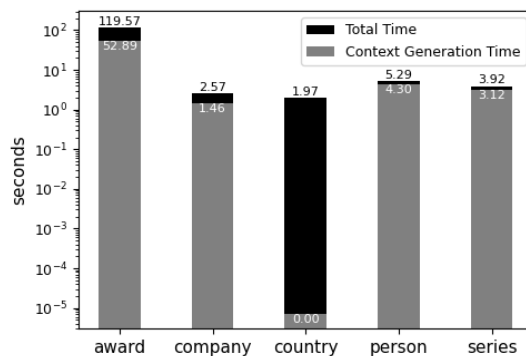
Table 2

Results for the validation and test datasets using the Llama3-8B-Instruct model.

Relation	Validation Dataset												Test Dataset		
	Triple			Triple-Context			Question			Question-Context			Question-Context		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
<i>awardWonBy</i>	0.27	0.03	0.06	0.50	0.26	0.30	0.29	0.08	0.12	0.50	0.30	0.32	0.60	0.67	0.62
<i>companyTradesAtStockExchange</i>	0.26	0.59	0.25	0.85	0.70	0.66	0.48	0.72	0.42	0.93	0.79	0.77	0.95	0.86	0.85
<i>countryLandBordersCountry</i>	0.83	0.96	0.85	0.98	0.98	0.98	0.85	0.99	0.88	0.99	0.98	0.98	0.98	0.93	0.94
<i>personHasCityOfDeath</i>	0.22	0.67	0.22	0.88	0.87	0.83	0.29	0.63	0.26	0.90	0.88	0.84	0.96	0.97	0.93
<i>seriesHasNumberOfEpisodes</i>	0.12	0.12	0.12	0.82	0.82	0.82	0.13	0.12	0.12	0.85	0.85	0.85	0.95	0.95	0.95
All Relations	0.31	0.54	0.31	0.86	0.82	0.80	0.40	0.57	0.37	0.90	0.85	0.84	0.95	0.92	0.91

Results for the validation dataset using the Llama3-70B-Instruct model.

Relation	Validation Dataset											
	Triple			Triple-Context			Question			Question-Context		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
<i>awardWonBy</i>	0.36	0.12	0.17	0.50	0.32	0.32	0.33	0.12	0.17	0.56	0.29	0.32
<i>companyTradesAtStockExchange</i>	0.45	0.60	0.37	0.95	0.80	0.77	0.73	0.77	0.61	0.93	0.78	0.74
<i>countryLandBordersCountry</i>	0.84	0.99	0.85	0.99	0.99	0.99	0.92	0.99	0.93	0.99	0.99	0.99
<i>personHasCityOfDeath</i>	0.41	0.72	0.39	0.93	0.88	0.87	0.51	0.71	0.42	0.94	0.87	0.86
<i>seriesHasNumberOfEpisodes</i>	0.30	0.30	0.30	0.84	0.84	0.84	0.46	0.42	0.42	0.85	0.85	0.85
All Relations	0.47	0.61	0.44	0.91	0.85	0.84	0.62	0.68	0.56	0.91	0.85	0.84

**Figure 3:** Comparison of Context Generation Time and Total Execution Time.

Discussion Using Llama3-8B-Instruct, the Triple configuration achieved an average macro F1-score of 31%. The best-performing relation was *seriesHasNumberOfEpisodes*, with an F1 score of 85%, while the worst was *awardWonBy*, with only 6%. The average score improved slightly, to 37%, with the question-based configuration (Question), as it mitigated some task formulation errors. However, the overall F1-score remained low, with the primary bottleneck being the knowledge extraction process. In particular, the queries involving specific information are probably not present in the LLM’s training datasets.

The best results for Llama3-8B-Instruct were obtained when contextual information was incorporated into the prompt, whether using triple or question prompt templates. In the validation dataset, the average macro F1-score of Triple-Context was 80% and the average score of Question-Context was 84%. The latter configuration achieves the best average macro F1-score, 91%, in the test dataset.

In all configurations, relation *awardWonBy* had the worst score. This is primarily due to the difficulty in generating relevant context for this relation. When reviewing the dataset entries, we realized that some entries lacked external links on their Wikidata pages (e.g., Q38215093).

Consequently, the context generator simply did not work in these cases. (In the test dataset, most of these entities have external links, which explains the improved score of 62%). Also, even when external links were available, extracting direct answers was often problematic. The structure of the HTML pages of links varied a lot, making it impossible to design a custom scraping plugin that worked well for the various cases.

Using Llama3-70B-Instruct we expected to mitigate the problem in the knowledge extraction process. While the results improved without the use of additional context, the model still struggles to provide accurate answers due to the specificity of the domain. However, an interesting point when using larger models is that considering the settings that use context, the way we instruct the LLM to perform the task is no longer a differentiating factor. And, aside from context acquisition issues in certain relations, the remaining errors are likely due to the entity resolution process and inaccuracies within the dataset.

5. Concluding Remarks

In this paper, we presented the LLM Store, a plugin to KIF that accesses an LLM and responds Wikidata-structured statements. We detailed how LLM Store works, passing for each of its pipeline components. Then, we created a system that uses the LLM Store to tackle an LM-KBC task, showing its efficiency. Results showed that our approach achieved a macro average F1-score of 91% in the test dataset of the LM-KBC'24 Challenge. The key feature to achieve this result is the use of our proposed context generation module. However, when not using contexts, the results may vary, since the knowledge extraction process has a large impact on accuracy. Very specific relations will probably present poor results. Besides, even when using the context generation module, our approach may have limitations. The reliance on specialized plugins that scrape textual context from specific parts of HTML pages introduces a dependency on the stability of pages, which can easily change and break the plugin. Thus, although LLM Store may be used as a source of knowledge, it should be used with caution.

Although our system was evaluated over the LM-KBC'24 Challenge, the approach used by LLM Store is not limited to the challenge's specific relations. LLM Store can address any question defined as a pattern involving Wikidata entities. Moreover, we claim that, as a highly configurable tool, our solution has the potential to accommodate other proposals such as those already presented in previous years of LM-KBC.

In future work, we aim to enhance the capabilities of LLM Store by introducing some key improvements and extensions. We will enable more expressive KIF filters, supporting logical *AND*, *OR*, and negation. Since the question-based prompt template yielded the best results, we aim to make it the default by developing a module that automatically generates questions from input filter patterns. We aim to improve the LLM-based entity resolution method to make it the default approach. While the baseline performed better in the Challenge, this method is flawed because it does not use any context during the disambiguation. Additionally, leveraging KIF's ability to create mixed stores (see [10]), we will explore combining multiple LLM Stores for complementary usage. This approach will involve assigning specific roles to each LLM Store within a mixer, along with rules governing their functions. For example, one LLM Store might serve as a judge, by reflecting about the responses from others and determining the final answer.

References

- [1] S. Razniewski, H. Arnaout, S. Ghosh, F. Suchanek, Completeness, recall, and negation in open-world knowledge bases: A survey, *ACM Computing Surveys* 56 (2024) 1–42.
- [2] M. Machado, G. Lima, E. Soares, V. Nascimento, R. Brandao, M. Moreno, An extensible approach for query-driven multimodal knowledge graph completion., in: *International Semantic Web Conference.*, CEUR-WS, 2022.
- [3] J. Chicaiza, P. Valdiviezo-Diaz, A comprehensive survey of knowledge graph-based recommender systems: Technologies, development, and contributions, *Information* 12 (2021) 232.
- [4] H. Paulheim, How much is a triple? estimating the cost of knowledge graph creation, in: M. van Erp, M. Atre, V. Lopez, K. Srinivas, C. Fortuna (Eds.), *Proc. ISWC 2018 Posters & Demonstrations, Industry and Blue Sky Ideas Tracks*, co-located with ISWC 2018, Monterey, USA, October 8–12, 2018, 2028. URL: https://ceur-ws.org/Vol-2180/ISWC_2018_Outrageous_Ideas_paper_10.pdf.
- [5] D. Vrandečić, M. Krötzsch, Wikidata: A free collaborative knowledgebase, *Commun. ACM* 57 (2014) 78–85. doi:10.1145/2629489.
- [6] S. Heindorf, M. Potthast, B. Stein, G. Engels, Vandalism detection in wikidata, in: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, 2016, pp. 327–336.
- [7] B. Veseli, S. Singhanian, S. Razniewski, G. Weikum, Evaluating language models for knowledge base completion, in: C. Pesquita, E. Jimenez-Ruiz, J. McCusker, D. Faria, M. Dragoni, A. Dimou, R. Troncy, S. Hertling (Eds.), *The Semantic Web*, Springer Nature Switzerland, Cham, 2023, pp. 227–243.
- [8] A. Ratner, C. Ré, Knowledge base construction in the machine-learning era: Three critical design points: Joint-learning, weak supervision, and new representations, *Queue* 16 (2018) 79–90. URL: <https://doi.org/10.1145/3236386.3243045>. doi:10.1145/3236386.3243045.
- [9] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al., Retrieval-augmented generation for knowledge-intensive NLP tasks, *Advances in Neural Information Processing Systems* 33 (2020) 9459–9474.
- [10] 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 knowledge sources, 2024. URL: <https://arxiv.org/abs/2403.10304>. arXiv:2403.10304.
- [11] MediaWiki Team, Wikibase/DataModel, 2024. URL: <https://www.mediawiki.org/wiki/Wikibase/DataModel>, last accessed September 30, 2024.
- [12] J.-C. Kalo, T.-P. Nguyenand, S. Razniewski, B. Zhang, LM-KBC: Knowledge base construction from pre-trained language models, in: *Semantic Web Challenge @ ISWC*, CEUR-WS, 2024. URL: <https://lm-kbc.github.io/challenge2024>.
- [13] M. Machado, J. Rodrigues, G. Lima, V. Silva, LLM Store: A KIF plugin for Wikidata-based knowledge base completion via LLMs, in: *Semantic Web Challenge @ ISWC*, CEUR-WS, 2024. URL: <https://lm-kbc.github.io/challenge2024>.
- [14] J. Odell, M. Lemus-Rojas, L. Brys, Wikidata for Scholarly Communication Librarianship, IUPUI University Library, Indianapolis, 2022. doi:10.7912/9Z4E-9M13.
- [15] 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.
- [16] F. Petroni, T. Rocktäschel, P. Lewis, A. Bakhtin, Y. Wu, A. H. Miller, S. Riedel, Language models as knowledge bases?, *arXiv preprint arXiv:1909.01066* (2019).
- [17] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, et al., Opt: Open pre-trained transformer language models, 2022, URL <https://arxiv.org/abs/2205.01068> 3 (2023) 19–0.
- [18] B. Zhang, I. Reklós, N. Jain, A. M. Peñuela, E. Simperl, Using large language models for knowledge engineering (LLMKE): A case study on Wikidata, in: *CEUR Workshop Proceedings*, volume 3577, CEUR-WS, 2023.
- [19] T. Li, W. Huang, N. Papasarrantopoulos, P. Vougiouklis, J. Z. Pan, Task-specific pre-training and prompt decomposition for knowledge graph population with language models, *arXiv preprint arXiv:2208.12539* (2022).
- [20] G. Qin, J. Eisner, Learning how to ask: Querying LMs with mixtures of soft prompts, *arXiv preprint arXiv:2104.06599* (2021).
- [21] Z. Zhong, D. Friedman, D. Chen, Factual probing is [mask]: Learning vs. learning to recall, *arXiv preprint arXiv:2104.05240* (2021).
- [22] S. Singhanian, J.-C. Kalo, S. Razniewski, J. Z. Pan, LM-KBC: Knowledge base construction from pre-trained language models, in: *Semantic Web Challenge @ ISWC*, CEUR-WS, 2023. URL: <https://lm-kbc.github.io/challenge2023>.