LLM Store: A KIF Plugin for Wikidata-Based Knowledge Base Completion via LLMs

Marcelo Machado[1], João M. B. Rodrigues[1], Guilherme Lima[1] and Viviane T. da Silva[1]

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

Abstract

We present LLM Store, a plugin of a Wikidata-based knowledge integration framework (called KIF) that can be used for knowledge base completion via pre-trained language models in real-time. The LLM Store is one of the participants of the LM-KBC Challenge @ ISWC 2024. This paper describes the implementation of LLM Store, its adaptation to the LM-KBC Challenge, and an analysis of its overall performance—it achieved a macro averaged F1-score of 90.83% in the challenge. One of the key features of LLM Store's approach is a context generation module which uses the external ids and sitelinks in Wikidata to generate textual context for a particular query on-the-fly. The code of our LLM Store-based system is available at https://github.com/IBM/kif-llm-store/tree/lm-kbc-challenge.

1. Introduction

Knowledge base construction (KBC) is a promising application of pre-trained (large) language models (LMs or LLMs). It can be seen as a controlled way to extract structured data from the LM while ensuring it operates within the boundaries of the knowledge base schema. In the LM-based version of KBC, the model is given a template for an edge (s, p, o) in the knowledge base graph—i.e., an edge with some of its components masked—and is asked to predict the masked parts. In the LM-KBC Challenge @ ISWC 2024 [1], the masked component is the object o and the task is to predict for a given template $(s, p, *)$ *all* possible o's matching the "$*$". This year, there is a 10 billion parameter limit for participating systems, and only a small set of five abstract relations is considered. As in last year's edition [2], the subjects (s) are Wikidata items, the objects (o) are Wikidata items or values, and the properties (predicates) (p) are abstract relations which may or may not correspond to a single property in Wikidata.

This paper presents LLM Store (https://github.com/IBM/kif-llm-store), our proposed system for the LM-KBC Challenge @ ISWC 2024. LLM Store is implemented as a store plugin for KIF [3], an open-source knowledge integration framework based on Wikidata [4]. A KIF store is an interface to a Wikidata view of a knowledge source. The basic store operation is the *filter*, which takes a pattern of the form

$$\texttt{Filter}(subject, property, value)$$

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); vivianet@br.ibm.com (V. T. da Silva)

and returns all statements matching it in the underlying knowledge source. Since KIF adopts Wikidata's data model, the components of KIF patterns and statements are Wikidata entities and values. The task of this year's LM-KBC Challenge can be rephrased naturally in KIF as the problem of generating statements matching a particular filter pattern—i.e., one in which the value is left unspecified. This similarity of purpose motivated us to develop and submit LLM Store to the challenge.

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, the 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. This pipeline has three main steps: probing, entity resolution, and parsing.

The probing process aims to find facts represented within an LLM. To achieve this, we adapted the LLM Store for the LM-KBC Challenge by converting the $(s, p, *)$ entries into equivalent KIF filters and evaluated them over the LLM Store.

While probing smaller models, the probability of hallucination is higher. This can be mitigated by providing relevant context information at the time of the prompt. To this end, we developed a retrieval-augmented generation (RAG) [5] module to obtain relevant context related to the input pattern. We used Wikidata as the source of external references. The generated contexts are incorporated into the final prompt. The LLM Store uses this prompt to query the underlying model and returns the response as text. This text is subsequently converted into a data structure for processing during the entity resolution step.

The goal of entity resolution is to match Wikidata entities with the labels returned from the probing (context-assisted) step. The LLM Store offers several methods for this task, including direct entity search via the Wikidata API[1], similarity-based, and LLM-based disambiguation techniques. Users can choose from these methods or define their own.

Finally, the parsing step converts the results containing Wikidata entities or values into statements that adhere to the KIF Store format. These statements constitute the final output of the completion process.

Summary of results Our RAG-based adaptation of the LLM Store plugin of KIF to the LM-KBC @ ISWC 2024 challenge, using *Llama3-8B-Instruct*[2], achieved a macro averaged F1-score of 90.83%. The detailed performance metrics for each of the five relations are shown in Table 3.

The rest of the paper is organized as follows. Section 2 describes our approach in detail, including LLM Store filters definition, context generation, prompt template, and entity resolution. Section 3 presents and discusses the results we obtained. Finally, section 4 concludes the paper and presents some perspective of future work.

[1]Wikidata REST API: https://www.wikidata.org/wiki/Wikidata:REST_API
[2]Meta Llama 3: https://ai.meta.com/blog/meta-llama-3/

2. Method

Our system processes each entry in the LM-KBC Challenge dataset according to the abstract relation it references. For each abstract relation, we instantiate an LLM Store with a specific configuration and execute the filter function to obtain the final answer. Consider the following example that queries Brazil's official language:

```python
from kif_lib import Store
from kif_lib.vocabulary import wd

# instantiate an LLM Store
llm_store = Store(
    'llm',                   # type of store
    llm_name='bam',          # LLM API (here IBM BAM - Big AI Models)
    llm_endpoint=os.environ['LLM_API_ENDPOINT'],   # api address
    llm_api_key=os.environ['LLM_API_KEY'],         # api key
    llm_model_id='meta-llama/llama-3-8b-instruct', # model id
)
# generate statements such that:
res = llm_store.filter(
    subject=wd.Q(155),          # subject is "Brazil" (Q155)
    property=wd.P(37))          # property is "official language" (P37)
```

Figure 1: Using the LLM Store to determine Brazil's official language.

When calling the `filter()` function, the pattern variables are incorporated into a prompt template. If no prompt template is provided, the LLM Store uses a default triple-based template that asks the model to complete the missing element in the relation, as shown below:

```
[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}} {{predicate}} ___

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.
```

In this template, *subject* and *predicate* are internally replaced with *Brazil* and *official language*, respectively. When setting up an LLM Store, users can customize the prompt template using the *prompt_template* parameter.

The common settings for the LLM Store across all relations are shown in Figure 1. Each relation has a defined filter and a set of specific settings, such as prompt template, context, and entity resolution method. In the following subsections, we present for each of the relations the filters and settings used.

2.1. LLM Store filters

Each abstract relation is translated into a filter. Since the LLM Store is a specialization of a KIF Store, we use Wikidata entities for filtering. The subject of the dataset entry can be used directly, as it is already a Wikidata item. However, the abstract relations need to be mapped to Wikidata properties. Table 1 shows the association between abstract relations and Wikidata properties.

Table 1
Mapping between abstract relations and Wikidata properties.

Abstract Relation	Wikidata Property
awardWonBy	ID: P1346, label: winner
companyTradesAtStockExchange	ID: P414, label: stock exchange
countryLandBordersCountry	ID: P47, label: shares border with
personHasCityOfDeath	ID: P20, label: place of death
seriesHasNumberOfEpisodes	ID: P1113, label: number of episodes

Some Wikidata properties do not precisely match the semantics of abstract relations. For instance, *countryLandBordersCountry* represents a relation where a country shares a land border with another country. The closest Wikidata property is *P47*, but its label (shares border with) does not fully capture the specific context of land borders between countries. Similarly, for *personHasCityOfDeath*, the closest Wikidata property is *P20* (place of death), but its range is not limited to cities. In these cases, we create composite filters. Below, we present the patterns defined to filter each entry according to these relations:

awardWonBy:

```
>>> predicate = wd.P(1346, 'winner')
>>> pattern = FilterPattern(subject, predicate)
```

companyTradesAtStockExchange:

```
>>> predicate = wd.P(414, 'stock exchange')
>>> pattern = FilterPattern(subject, predicate)
```

countryLandBordersCountry:

```
>>> predicate = wd.P(47, 'shares border with')
>>> country = wd.Q(6256, 'country')
>>> instance_of = wd.P(31, 'is a')
>>> pattern = FilterPattern('place of death')
```

where the composite filter indicates that the object must be an instance of *Country*.

personHasCityOfDeath:

```
>>> city = wd.Q(515, 'city')
>>> instance_of = wd.P(31, 'is a')
>>> pattern = FilterPattern(subject, predicate, [instance_of(city)])
```

where the composite filter indicates that the object must be an instance of *City*.

seriesHasNumberOfEpisodes:

```
>>> predicate = wd.P(1113, 'number of episodes')
>>> pattern = FilterPattern(subject, predicate)
```

Consider the following example entry from the test dataset:

```
{
    "SubjectEntityID": "Q317038",
    "SubjectEntity": "Max Planck Medal",
    "Relation": "awardWonBy"
}
```

To filter this entry using the LLM Store, the corresponding Python code would be:

```
subject = wd.Q(317038, 'Max Planck Medal')
predicate = wd.P(1346, 'winner')
pattern = FilterPattern(subject, predicate)

stmts = llm_store.filter(pattern=pattern)
```

The LLM Store is responsible for incorporate the pattern components into a prompt template that will form the final prompt to be sent to the underlying LLM.

2.2. Context Generation

Context generation is a key aspect of our system. We developed a module dedicated to generating textual context to support the LLM Store in generating statements for a given filter pattern.

Our retrieval-augmented generation (RAG) process is based on Wikidata. Given a Wikidata entity Q described in the pattern, the *context generator* module: (1) fetches all external ids and sitelinks from Q's Wikidata page in parallel, (2) passes these links through a series of scraping plugins to extract text from the target HTML pages, and (3) ranks all extracted texts by embedding similarity against a textual version of the input pattern.

For the LM-KBC Challenge, we adjusted this approach by skipping step (3) whenever a specific external link was available for a particular filter pattern. For example, if the pattern corresponding to the *seriesHasNumberOfEpisodes* relation and the subject's external IDs included a link to its IMDb page[3], we used this link and passed it through a custom IMDb scraper plugin. This plugin searches specifically for the number of episodes and generates a synthetic sentence to be used as context.

[3]IMDb is a popular and authoritative source for movies, TV, and celebrity content: https://www.imdb.com/

After obtaining the information from the context generator, we adjust the LLM Store, initially set with common settings, to incorporate the context into the prompt template. This is done by setting the context attribute with the generated information. Below, we discuss the context generator plugins used to generate context for each abstract relation of the LM-KBC dataset.

awardWonBy For this relation, we created a plugin called *ner-extract*. Its goal is to recognize named entities within an HTML page. We also developed a version of this plugin that searches for tables containing the names of award winners in Wikipedia pages. The returned entities are formatted as a list of names.

Specifically for this relation, the number of responses can be large, which makes the prompt and the model's response exceed the maximum token size. To cope with that, we break down the returned results according to each letter of the alphabet. Then, we iterated through each of these parts, calling the LLM with the following context:

```
The winners of the {{subject}} award are among the following: {names}
```

Here, *names* is a list containing only entities whose names begin with the letter being iterated from A to Z. At the end of the process, we aggregated the results returned for each part.

companyTradesAtStockExchange For this relation, we created two plugins, one to intercept Wikipedia pages and another to access Yahoo Finance pages. For the final submission, we used only the Wikipedia plugin, which produced the best results.

The Wikipedia plugin searches for stock exchange information in the infobox of the subject's Wikipedia page. Depending on the page's language, it may be necessary to specify the terms referring to "stock exchange" in that language. If no such information is found, the context is set with a sentence indicating that no information was found. This approach prevents the LLM Store from fabricating an answer when the information is unavailable. However, it also limits scenarios where the LLM Store could correctly provide an answer not found during the context search. When information is returned, the context is set as follows:

```
{{subject}} shares can be traded at {exchange_names}.
```

Here, *exchange_names* refers to a list of the returned stock exchange names. If no stock exchange information is found, the context is set to:

```
The company {{subject}} does not trade shares on any exchange.
```

countryLandBordersCountry For this relation, we do not use the context generation module. In this case, we use an open API to access a list of countries that border other countries[4] and use the following as context:

[4] https://restcountries.com/v3.1/all

```
The countries that share a land border with {{subject}} are: {country_borders}
```

Here *country_borders* is a list of countries' names returned by the API. Because the number of answers is relatively small and rarely change, for this relation, we keep a cache file with the contents of the open API call. When the information is not found, we also indicate as context the sentence that no information was found on countries that share a border with the country in question.

personHasCityOfDeath For this relation, we created a plugin called *wikipedia-place-of-death*. This plugin extracts the death place information in the infobox of the subject Wikipedia page. Depending on the page's language, it may be necessary to specify the terms referring to "place of death" for that language. If no information is found, we indicate it in the context. The default sentence we use as context is:

```
The city where {{subject}} died is in the following sentence: {death_sentence}.
```

Here, *death_sentence* is replaced by the result obtained from the plugin. This usually contains the city together with the country and the date.

seriesHasNumberOfEpisodes For this relation, we created three plugins, one to intercept Wikipedia pages, another to access IMDb pages, and another to access Google's infobox.

The Wikipedia plugin accesses the page's HTML and searches for the number of episodes information in the infobox. If no information is found, the system goes to the next plugin. The second visited plugin is the IMDb-based. Finally, if the first two plugins do not find an answer, we use the Google plugin to search for information about the number of episodes of the series and tries to capture the answer in the infobox or on the first page of the results. Finally, the sentence used as context is:

```
The number of episodes of {{subject}} is {total_episodes}.
```

Here, *total_episodes* is the number of episodes returned by a plugin.

2.3. Prompt Template

A prompt template is a string with placeholders for creating dynamic prompts. The LLM Store replaces these placeholders with filter components and context. If the user does not specify a prompt template, the triple-based default is used. However, when initializing an LLM Store with the *context* attribute set, the default prompt template changes to the following:

```
[SYSTEM]
You are a helpful and honest assistant that resolves a TASK based on the CONTEXT.
↪  Only perfect and explicit matches mentioned in CONTEXT are accepted. Please,
↪  respond concisely, with no further explanation, and truthfully.

[USER]
```

```
TASK:
Fill in the gap to complete the relation:
{{subject}} {{predicate}} ___

CONTEXT:
{{context}}

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.
```

Our final results were achieved using an adaptation of the default prompt. Our analysis using the validation dataset demonstrated that the best results in the probing process were related to the use of straightforward questions. Therefore, we used question-based prompt templates[5], where the TASK is framed as a question about the subject. For instance, for the relation *seriesHasNumberOfEpisodes*, the TASK part in the prompt template is:

```
...
TASK:
How many episodes does the series {{subject}} have?
...
```

The LLM Store maps each masked variable in the prompt template to a known parameter. For example, *{{subject}}* is bound to the subject component of the pattern, while *{{context}}* is bound to the *context* parameter passed in when the LLM Store was initialized.

2.4. Entity Resolution

We implement three methods to transform the returned labels into Wikidata entities: baseline, similarity-based, and LLM-based.

The first method is the *baseline* approach. It performs a search using the Wikidata REST API with the label as the search key and retrieves the top response. To use this method, we set the LLM Store attribute *disambiguation_method* to *"baseline"*.

The second method is the *similarity-based*. It also accesses the Wikidata REST API using the label as the search key. However, since multiple candidates may be returned, it transforms the description of each candidate entity into embeddings and compares them with an embedding derived from the (context-assisted) prompt using cosine similarity. To use this method, we set the LLM Store attribute *disambiguation_method* to the value *"sim"*.

Based on [6], the third method is *LLM-based*. It also searches for entities in the Wikidata REST API using the label as the search key. But, to disambiguate the multiple candidate entities returned, this method uses a LLM. In short, the prompt for this step asks the LLM to respond to a given task by answering which entity has the description that most closely matches what is required in the task. Different than [6], we use the generated context to support the disambiguation. To use this method, we set the attribute *disambiguation_method* to *"llm"*.

[5] https://github.com/IBM/kif-llm-store/blob/main/data/lm-kbc-2024/prompt_templates/question_prompts.csv

Despite being a naive implementation, the baseline method achieved the best disambiguation results on the validation dataset. Therefore, we selected this approach to generate the final results with the test dataset.

For relation *seriesHasNumberOfEpisodes* we do not need to link the response to Wikidata entities because it should be literal values. In this case, we set the *disambiguate* parameter to false, so the pipeline skips the entity resolution step.

3. Results and Discussion

The main entry point to our system is the *run.py*[6] script, located in the root directory of the project. This script executes the system with the specified parameters and exports the results in the format required by the LM-KBC Challenge. The exported results can then be used for final evaluation by comparing the obtained results with the expected ones.

Before running our system with its best configuration on the test dataset, we evaluate our proposal using the validation dataset of the LM-KBC Challenge. Four configurations were assessed to understand the impact of each component on the final results. These configurations were named: *Triple*, *Triple-Context*, *Question*, and *Question-Context*. Configurations beginning with "Triple" indicate the use of the default prompt template from the LLM Store, which is based on triples. Configurations beginning with "Question" utilize a question-based prompt template, where abstract relations are transformed into direct questions in natural language. The term "Context" means that the context generation module, along with its plugins, is used to augment the prompt with additional external information. Table 2 shows the results for the validation dataset considering each relation.

Table 2
Results for the validation dataset using the *Llama3-8B-Instruct* model.

Relation	Validation Dataset											
	Triple			Triple-Context			Question			Question-Context		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
awardWonBy	0.27	0.03	0.06	0.50	0.26	0.30	0.29	0.08	0.12	0.50	0.30	0.32
companyTradesAtStockExchange	0.26	0.59	0.25	0.85	0.70	0.66	0.48	0.72	0.42	0.93	0.79	0.77
countryLandBordersCountry	0.83	0.96	0.85	0.98	0.98	0.98	0.85	0.99	0.88	0.99	0.98	0.98
personHasCityOfDeath	0.22	0.67	0.22	0.88	0.87	0.83	0.29	0.63	0.26	0.90	0.88	0.84
seriesHasNumberOfEpisodes	0.12	0.12	0.12	0.82	0.82	0.82	0.13	0.12	0.12	0.85	0.85	0.85
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**

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 probing process. In particular, the queries involving specific information are probably not present in the LLM's training datasets. The best results were obtained when contextual information was incorporated into the prompt, whether using triple or question prompt templates. The average macro F1-score

[6] https://github.com/IBM/kif-llm-store/blob/lm-kbc-challenge/run.py

of *Triple-Context* was 80% and the average score of *Question-Context* was 84%. As the latter configuration achieves the best average macro F1-score, we use in to evaluate our system over the test dataset.

Table 3 presents the results of our system using the challenge's test dataset. The results show that our system achieved an average macro F1 score of approximately 90%. The best performance was for the relation *seriesHasNumberOfEpisodes*, with an average F1 score of 95%, while the worst performance was for the relation *awardWonBy*, with an F1 score of approximately 62%.

Table 3
Results for the test dataset using the *Llama3-8B-Instruct* model.

Relation	macro-p	macro-r	macro-f1
awardWonBy	0.6031	0.6717	0.6160
companyTradesAtStockExchange	0.9550	0.8573	0.8550
countryLandBordersCountry	0.9819	0.9297	0.9369
personHasCityOfDeath	0.9600	0.9700	0.9300
seriesHasNumberOfEpisodes	0.9500	0.9500	0.9500
All Relations	0.9504	0.9197	**0.9083**

These results demonstrate that our system is efficient in the knowledge base completion task when supported by a "good" context generator. Most relations achieved high precision and recall, above 90%, indicating that the system returns most correct answers while keeping incorrect answers to a minimum.

The relation *awardWonBy* had the lowest performance, mainly due to the difficulty in finding relevant context information for the LLM. Additionally, the large number of possible results and the broad range of answer types (sometimes referring to a person, other times to a thing) made it challenging to create a precise prompt. This highlights the limitations of our system when dealing with relations that have these characteristics.

Another limitation of our work is the reliance on plugins that scrape the content of HTML pages. These plugins can easily break if the structure of the target pages changes. Consequently, they must be continuously updated to maintain their functionality and effectiveness. That said, the system comes with a *fallback* plugin that uses a generic approach to obtain text from an arbitrary HTML. Improvements in the similarity ranking of the extracted paragraphs could improve the overall performance of the fallback plugin.

4. Final Remarks

In this paper, we described the adaptation of LLM Store to the LM-KBC Challenge @ ISWC 2024. We explored integrating LLM and RAG for knowledge base completion. The developed system, leveraging the LLM Store and context generation from Wikidata, achieved high-performance metrics, particularly in relations with well-defined contexts. Our results suggest a strong ability to handle diverse queries with an average macro-F1 score of 90.83%, showcasing the system's efficiency.

The work is not without limitations. The low performance on relations like *awardWonBy* highlights the challenges in generating good prompts and text context for abstract queries.

Furthermore, 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.

We also emphasize that, despite the promising results, this system can still produce false statements. This highlights the need for caution when utilizing this information, especially in sensitive contexts. For *personHasCityOfDeath*, for example, a macro-F1 score of less than 100% indicates that the system sometimes fails to recognize the correct place of death or erroneously predicts incorrect places.

While our system primarily focused on the LM-KBC Challenge, the LLM Store's approach is not limited to the specific relations within the challenge. LLM Store is designed to handle any query defined as a pattern involving Wikidata entities, using default probing and entity resolution methods. Furthermore, as a KIF Store, it can seamlessly integrate with other data sources, potentially enhancing the robustness of various applications. Beyond the scope of the Challenge, depending on the use case, the LLM Store could directly return responses generated by plugins or it could use the LLM as a judge to decide on the information coming from different plugins.

In future work, we intend to generalize the LLM Store to handle other types of knowledge base completion tasks. For example, we aim to address scenarios where the subject is the missing element, i.e., $(*, p, o)$, or even the predicate, i.e., $(s, *, o)$. We also envision to improve LLM Store by using existing tools, such as LangChain[7], integrating components more widely accepted in the community and facilitating external collaboration. This future endeavors would further enhance the versatility and applicability of our system across various KBC tasks.

References

[1] 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.

[2] S. Singhania, 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.

[3] 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.

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

[5] 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.

[6] B. Zhang, I. Reklos, 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.

[7] https://www.langchain.com/