Towards Large Language Models Interacting with Knowledge Graphs Via Function Calling

Sven Hertling[1,2], Harald Sack[2,3]

[1]Data and Web Science Group, University of Mannheim, Germany
[2]FIZ Karlsruhe – Leibniz Institute for Information Infrastructure, Eggenstein-Leopoldshafen, Germany
[3]Karlsruhe Institute of Technology (AIFB), Karlsruhe, Germany

Abstract

This position paper introduces a new way for large language models (LLMs) to interact with Knowledge Graphs (KGs), especially for information extraction. Most relation extraction approaches focus on finding the corresponding textual spans for subject, relation, and object in a text but ignore the actual state of the KG and the schema defined therein. In this work, the function-calling possibility of LLMs is explored to search for already defined entities in the KG and add information in the form of triples to the KG. A huge improvement over previous techniques is that it can also deal with various large ontologies / KGs and that the added information fits to a provided KG on a schema and instance level. Further validation checks can be added to increase the quality of the extracted statements. A small playground is provided to analyze the effectiveness of the approach.

1. Introduction

The rise of the transformer architecture and Large Language Models (LLMs) has led to increased use of those models in information extraction tasks to enhance extraction quality. For closed source information extraction (all possible relations are already given) either the model is trained to extract a fixed set of relations or the possible relations are provided in the prompt. The latter approach will not scale for large knowledge graphs with like Wikidata [1], which contains more than 6,236 relations [2]. Moreover, the former approach requires additional steps, such as entity disambiguation, which results in error propagation.

In this work, we propose an alternative technique to combine LLMs and KGs via function calling. Some LLMs like GPT-3.5 Turbo (closed source) or Mistral Small [3] and Large (open-source) can already expect a set of functions in the prompt which are called when the model returns a structured request to run one of the functions. The parameters of the function are similarly generated by the LLM.

Given carefully designed functions that interact with the KG, as e.g., searching/creating instances or properties, the information extraction process can be simplified and scaled to larger ontologies and KGs.

KBC-LM'24: Knowledge Base Construction from Pre-trained Language Models workshop at ISWC 2024
sven.hertling@uni-mannheim.de (S. Hertling); harald.sack@fiz-karlsruhe.de (H. Sack)

2. Related Work

In the NLP community, the closest task to our approach is referred to as relation extraction, and the survey by Xu et al. [4] provides a good overview of all LLM-related information extraction approaches. Zaratiana et al. [5] use an encoder-decoder architecture to jointly extract the entities and relations. Similarly, Cabot et al. [6] use autoregressive seq2seq models to overcome the error propagation of multiple-step pipelines. One approach that also comprises entity linking is ReLiK [7], which consists of two modules – retrieve and read. The retriever module identifies candidate relations and entities, whereas the reader module distinguishes the retrieved entities and relations and links them to the corresponding textual spans. To the best of our knowledge, no related work is available that focuses on information extraction with LLMs via function calls.

3. Function Calling Setup

Given the scenario that some instances, properties, and classes are already defined in the KG, the task is to extract information from natural language text that fits to the given schema. This means that each resource is defined in the KG and no two resources are created for the same real-world entity. The idea is to allow LLMs to interact with the KG via function calls. All possible functions, together with a description of the functions and parameters, are provided in the prompt as well as the text to extract information from. The model can then return a structured query which is executed, and the textual result is provided to the LLM, which can then decide to call another function or stop the chat by providing a final statement without calling a function.

According to the example sentence "Paris is the capital of France", we will show which functions should exist and which parameters they should expect. All functions are listed in Tables 1, 2, and 3. In the first step, the subject (in this case "Paris") needs to be searched in the KG because it might already exist. Thus, a function `search_instance` which expects some text is needed. The result of such a function needs to be a textual representation of possible candidates in the KG which contains enough information to disambiguate the entities. Thus, the identifier (URL), the *rdfs:label* and *rdfs:comment* are included in the result. A simple SPARQL query is shown in Table 1. The result of the SPARQL query is a mapping between the selected variables and the possible substitutions, which need to be serialized as text for the LLM. We implemented all serializers available in RDFLib, which are CSV, JSON, TXT, and XML[1].

Based on the result of the `search_instance` function, the LLM needs to decide if the entity described in the text is already represented in the KG or not. In the latter case, the LLM can either call the `search_instance` function with another text to search for (which increases the possibility of finding entities that are represented with synonyms) or create the entity in the KG. To this end, an additional function `create_instance` is required (shown in Table 2). It expects a label, a comment, and an instance type. All function parameters can freely be filled with text generated by the LLM.

Due to the fact that the LLM calls an actual function that is implemented in Python, various

[1] https://rdflib.readthedocs.io/en/stable/plugin_query_results.html

Table 1
Functions that search for entities in the knowledge graph.

Function	SPARQL query

search_instance(search_text)	`SELECT ?instance_id ?label ?comment ?typeLabel`
	`WHERE {`
	`  ?instance_id a ?type;`
	`  rdfs:label ?label; rdfs:comment ?comment.`
	`  ?type rdfs:label ??typeLabel.`
	`  FILTER regex(?label, "{search_text}", "i")`
	`}`

search_property(search_text)	`SELECT ?property_id ?label ?comment ?domain ?range`
	`WHERE {`
	`  {?property_id a rdf:Property. }`
	`  UNION {?property_id a owl:ObjectProperty. }`
	`  UNION {?property_id a owl:DatatypeProperty. }`
	`  ?property_id rdfs:label ?label; rdfs:comment ?comment;`
	`  rdfs:domain ?domain; rdfs:range ?range.`
	`  FILTER regex(?label, "{search_text}", "i")`
	`}`

search_class(search_text)	`SELECT ?class_id ?label ?comment`
	`WHERE {`
	`  { ?class_id a rdfs:Class. }`
	`  UNION { ?class_id a owl:Class. }`
	`  ?class_id rdfs:label ?label; rdfs:comment ?comment.`
	`  FILTER regex(?label, "{search_text}", "i")`
	`}`

validation checks can be performed. As an example, the instance type parameter needs to be a class defined in the KG. Most often, LLMs create URIs that are based on public KGs such as DBpedia [8], YAGO [9], or Wikidata [1]. Thereby, it often occurs that the instance type URI of Paris might look like https://dbpedia.org/ontology/City because these URIs appear often in the training data. But in the function implementation, a check is inserted to test if the URI is actually a class in the KG. In case the validation check fails, a textual representation of the error is returned, as e.g., "The instance type is not a class. Search for it with function search_class.", which triggers the LLM to make further function calls. The `search_class` function works in the same way as `search_instance` but restricts the search to RDFS or OWL classes. The user can also allow the model to call functions that make changes on the schema level, as e.g., adding classes or properties (see Table 3). Thus, the previous error message about an undefined class can be extended by "If the class cannot be found, create it with create_class function.". This function expects a label, comment, and optionally a superclass. If a superclass is given, then it also needs to be a defined class in the KG.

After the instance for Paris is either found or created in the KG, the same approach is applied to the property (capital of) and object (France). The LLM decides which information to extract from the given text, which entities to search/create, and the order in which they are processed. Finally, the statement containing the subject, predicate, and object can be added to the KG via

Table 2
Functions that create instances or whole statements.

Function	SPARQL query / validation code

create_instance
(label, comment,
instance_type)
	check if instance_type is a class in the KG:
	`ASK{`
	`  { <{instance_type}> a rdfs:Class. }`
	`  UNION { <{instance_type}> a owl:Class. }`
	`}`
	if not, return "The instance type of the instance is not a class".
	else, create a new URI named instance_uri and execute:
	`INSERT DATA {`
	`  <{instance_uri}> a <{instance_type}>;`
	`  rdfs:label "{label}"; rdfs:comment "{comment}".`
	`}`
	return instance_uri

create_statement
(subject, predicate, object)
	check that subject and object are resources defined in the KG
	and predicate is a property defined in the KG
	check that subject and object are coherent with domain and range
	if all validation checks are successful, execute:
	`INSERT DATA {`
	`  <{subject}> <{predicate}> <{object}>.`
	`}`

Table 3
Functions that change the schema of the given knowledge graph.

Function	SPARQL query / validation code

create_class
(label, comment,
super_class)
	if super_class is given, check that it is a RDFS or OWL class in the KG
	create a new URI named class_uri and execute:
	`INSERT DATA {`
	`  <{class_uri}> a owl:Class; rdfs:subClassOf <{super_class}>;`
	`  rdfs:label "{label}"; rdfs:comment "{comment}";`
	`}`
	return class_uri

create_property
(label, comment,
domain, *range*,
super_property_id)
	if domain and/or range is given, check that it is a class in the KG
	if super_property_id is given, check that it is a property in the KG
	if all validation checks are successful, create property_uri and execute:
	`INSERT DATA {`
	`  <{property_uri}> a rdf:Property;`
	`  rdfs:subPropertyOf <{super_property_id}>;`
	`  rdfs:label "{label}"; rdfs:comment "{comment}";`
	`  rdfs:domain <{domain}>; rdfs:range <{range}>.`
	`}`
	return property_uri

the `create_statement` function.

The validation checks are a huge advancement because it can be ensured that the added instance will have the correct class defined in the KG, which would not be possible otherwise. Thus, it is also a disadvantage to just provide a function `execute_sparql` (which would also be possible) because of those additional validations.

The whole extraction works in a greedy way because only one text is processed at a time. Thus, a decision cannot be revoked once taken, e.g., a domain of a property is fixed when the property is created. Still, there are possibilities to add more functions that change/remove classes, properties, or instances. One advantage of the greedy approach is that no further alignment is necessary, which would be the case when each Wikipedia page is processed in parallel with an empty KG because each resulting KGs needs to be aligned (the amount of alignments is usually $\binom{n}{2}$, where n is the number of KGs [10]).

We provide an implementation[2] of all functions using the *LangChain* framework[3] to easily parse the LLM output and call the corresponding functions implemented in Python using RDFLib. In our first experiments with GPT-3.5 Turbo, it turned out that the best SPARQL result serialization is CSV (probably because it is very structured). But the final goal is to use open-source LLMs to have a reproducible setup and the possibility of running the information extraction model on our hardware without paying for API access.

4. A Dataset for Fine-Tuning

Due to the fact that most LLMs are not specifically trained for function calls, there is a need for a dataset showing the use of exactly those functions. Some of the most promising open-source models are the Mistral models Small [3], Large, Nemo, and Mixtral 8x22B[4] because they are already trained for function calls. Nevertheless, also general pretrained models can be fine-tuned to be able to make function calls.

But all of the models would rely on the definition of the function without any example of how to apply the functions and what to do with the result of a function call. Thus a dataset needs to be generated to show how these functions work in actual information extraction scenarios. A natural choice is Wikipedia because the text is usually checked by multiple persons, and many knowledge graphs based on Wikipedia exist. For their REBEL approach (Relation Extraction By End-to-end Language generation) [6], the authors created a large dataset for end-to-end relation extraction[5]. The dataset is based on Wikipedia abstracts and uses links for entity disambiguation. The alignment from the Wikipedia links to Wikidata is performed via *wikimapper*[6]. To check that the given text containing two mapped entities actually entails the relation, they use RoBERTa [11] by providing the sentence and a verbalized version of the complete triple: `[CLS]{sentence}[SEP]{verbalized triple}[EOS]` All triples with a higher entailment prediction than 0.75 are kept. With such a dataset all function calls can be

[2] https://github.com/sven-h/LLMs4IE
[3] https://github.com/langchain-ai/langchain
[4] https://docs.mistral.ai/capabilities/function_calling/
[5] https://github.com/Babelscape/crocodile
[6] https://github.com/jcklie/wikimapper

exemplified and shown to the LLM. The dataset can be even further extended by using not only the abstracts but the whole Wikipedia article. Furthermore, it should be taken care of that usually only the first occurrence of an entity is linked, whereas all following representations are not linked explicitly [7]. Thereby the number of candidates can be further increased.

In addition, all surface forms[8] of entities can be used to showcase that the parameter for search functions can be changed to actually find the correct entity in the KG. Given the running example, the search for the entity France could be called multiple times with different textual representations such as `search_instance("France")` and `search_instance("French Republic")`. It shows that the search functions can and should be called multiple times in case the correct entity is not yet retrieved. In comparison to relation extraction, the dataset also needs to reflect the different states of the KG, e.g., the entity is not contained in the KG or the search functions return multiple possible candidates. This could be regulated by a hyperparameter $\gamma \in [0, 1]$, which represents a probability of how often the KG should be assumed to not contain the required entity. In cases where the entity is missing, the create functions should be used. All information required to create an instance, property, or class is already defined in the KG and can be used to fill the parameters.

5. Conclusion and Future Work

In this paper we proposed a new technique to combine Large Language Models and Knowledge Graphs. The basic idea is to let the LLM interact with the KG by function calls that directly reflect the state of the KG. Thus, all extracted triples will be coherent to the schema, which is already defined but can also be changed by the LLM if corresponding functions are provided.

The next step is to create the dataset and fine-tune open-source LLMs such as Mistral or Llama 3 models [13]. Due to the size of the models, the training will be conducted by using a quantized model with low-rank adaptations (QLoRA) [14].

Up to now, the dataset only provides examples of relations between individuals but not between individuals and literals. The main difficulty is to map the literal value to the text in a Wikipedia article, which can be arbitrarily complex, e.g., dates that are written in completely different ways, such as '2024-01-01', 'Jan. 1, 2024', or even not the full information like 'Jan. 1'.

Currently, the search functions only search for entities where the exact search string is contained in the labels, and the LLM needs to call those functions possibly multiple times. An improvement would be to implement a semantic search via Sentence-BERT models [15] like `all-mpnet-base-v2`[9].

Another line of future work is to extend the number of involved LLMs and to include a human in the loop. Especially when humans create ontologies, the best results are achieved if more people interact together. In the same way, the number of LLMs could be increased. Technically, another function like `ask_expert` (similarly a function `ask_human_expert` for human involvement) could be introduced.

[7] https://en.wikipedia.org/wiki/Wikipedia:Manual_of_Style/Linking#Duplicate_and_repeat_links

[8] The term "surface form" was mainly introduced by DBpedia Spotlight [12] and covers all link texts that refer to one article/entity in the KG.

[9] https://huggingface.co/sentence-transformers/all-mpnet-base-v2

References

[1] D. Vrandečić, M. Krötzsch, Wikidata: a free collaborative knowledgebase, Communications of the ACM 57 (2014) 78–85.

[2] N. Heist, S. Hertling, D. Ringler, H. Paulheim, Knowledge graphs on the web–an overview, Knowledge Graphs for eXplainable Artificial Intelligence: Foundations, Applications and Challenges (2020) 3–22.

[3] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, et al., Mistral 7b, arXiv preprint arXiv:2310.06825 (2023).

[4] D. Xu, W. Chen, W. Peng, C. Zhang, T. Xu, X. Zhao, X. Wu, Y. Zheng, E. Chen, Large language models for generative information extraction: A survey, arXiv preprint arXiv:2312.17617 (2023).

[5] U. Zaratiana, N. Tomeh, P. Holat, T. Charnois, An autoregressive text-to-graph framework for joint entity and relation extraction, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 38, 2024, pp. 19477–19487.

[6] P.-L. H. Cabot, R. Navigli, Rebel: Relation extraction by end-to-end language generation, in: Findings of the Association for Computational Linguistics: EMNLP 2021, 2021, pp. 2370–2381.

[7] R. Orlando, P.-L. Huguet Cabot, E. Barba, R. Navigli, ReLiK: Retrieve and LinK, fast and accurate entity linking and relation extraction on an academic budget, in: Findings of the Association for Computational Linguistics ACL 2024, Association for Computational Linguistics, 2024, pp. 14114–14132. URL: https://aclanthology.org/2024.findings-acl.839.

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

[9] F. M. Suchanek, G. Kasneci, G. Weikum, Yago: a core of semantic knowledge, in: Proceedings of the 16th international conference on World Wide Web, 2007, pp. 697–706.

[10] S. Hertling, H. Paulheim, Order matters: matching multiple knowledge graphs, in: Proceedings of the 11th Knowledge Capture Conference, 2021, pp. 113–120.

[11] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, Roberta: A robustly optimized bert pretraining approach, arXiv preprint arXiv:1907.11692 (2019).

[12] P. N. Mendes, M. Jakob, A. García-Silva, C. Bizer, Dbpedia spotlight: shedding light on the web of documents, in: Proceedings of the 7th international conference on semantic systems, 2011, pp. 1–8.

[13] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, et al., The llama 3 herd of models, arXiv preprint arXiv:2407.21783 (2024).

[14] T. Dettmers, A. Pagnoni, A. Holtzman, L. Zettlemoyer, Qlora: Efficient finetuning of quantized llms, Advances in Neural Information Processing Systems 36 (2024).

[15] N. Reimers, I. Gurevych, Sentence-bert: Sentence embeddings using siamese bert-networks, in: Empirical Methods in Natural Language Processing (EMNLP), ACL, 2019, pp. 3980–3990. URL: https://arxiv.org/abs/1908.10084.