On the Pertinence of LLMs for Ontology Learning

Marion Schaeffer[1,2,*], Matthias Sesboüé[1], Léa Charbonnier[1], Nicolas Delestre[1], Jean-Philippe Kotowicz[1] and Cecilia Zanni-Merk[1]

[1] *INSA Rouen Normandie, LITIS (UR 4108), Normandie Université, 76000 Rouen, France*
[2] *Wikit, 41 quai Fulchiron, 69005 Lyon, France, https://www.wikit.ai/*

Abstract
Ontology learning from text is traditionally approached as sub-tasks tackled with linguistic, statistical or logic-based methods. Large language models and their generation capabilities have recently caught much interest. We investigate the pertinence of such generative models for ontology learning. We evaluate the created ontologies on two different use cases by aligning with a reference ontology and compare components for each sub-task using the OLAF ontology learning framework. In addition to demonstrating the relevance of large language models for ontology learning, we discuss component combinations, LLM size, and environmental impact in creating efficient pipelines while limiting resource consumption.

Keywords
LLM, Ontology Learning, Knowledge Graph Construction

1. Introduction

Ontologies and Large Language Models (LLMs) have recently caught much attention. With LLMs demonstrating outstanding capabilities, the research communities converged to explore how LLMs could be leveraged for Knowledge Graph (KG) construction and ontology engineering tasks [1]. The particular task of Ontology Learning (OL) involves moving from unstructured information in text to structured data as axioms in a chosen formalism. The process can be considered end-to-end, like a translation, or a series of interlinked specialised sub-tasks. With LLMs' ability to generate and summarise text, reason and identify relevant information, they are good candidates as tools for OL.

In this work, we focus on OL from text and investigate the pertinence of LLM for such a task. We study unsupervised techniques from term to axiom extraction. We explore zero-shot prompting, i.e., prompting with only instructions, and few-shot prompting, i.e., prompting with a few examples added to the instructions. We investigate the influence of LLM size on its performance in the OL task as well as the energy consumption and greenhouse gas emissions of such models for the OL task. Our study is based on two different use cases. On the one

NLP4KGC: 3rd International Workshop on Natural Language Processing for Knowledge Graph Creation, September 17, 2024, Amsterdam, Netherlands
*Corresponding author.
marion@wikit.ai (M. Schaeffer); matthias.sesboue@insa-rouen.fr (M. Sesboüé); lea.charbonnier@insa-rouen.fr (L. Charbonnier); nicolas.delestre@insa-rouen.fr (N. Delestre); jean-philippe.kotowicz@insa-rouen.fr (J. Kotowicz); cecilia.zanni-merk@insa-rouen.fr (C. Zanni-Merk)
0000-0001-9854-7482 (M. Schaeffer); 0000-0003-2665-2229 (M. Sesboüé); 0000-0002-0925-5582 (N. Delestre); 0000-0003-3985-2190 (J. Kotowicz); 0000-0002-5189-9154 (C. Zanni-Merk)

hand, we select a reference ontology and generate its descriptive text using an LLM. On the other hand, we manually build an ontology from a text using the Ontology Development 101 guide [2]. We create three different pipelines, execute them with the texts as input and evaluate their performances by aligning the ontologies produced with the reference ontologies. We also compare components' efficiency in unitary ways to build pipelines maximising performance while limiting the computational resources consumed. The code is publicly available in our repository: https://github.com/wikit-ai/olaf-llm-nlp4kgc2024.

2. Related work

The terms ontology and KG are often used similarly. In computing, an ontology is a concrete, formal representation of what terms mean within the scope in which they are used [3]. A KG is a graph of data intended to accumulate and convey knowledge, whose nodes represent entities of interest and whose edges represent relations between these entities [3]. This paper considers a KG as an ontology with the data. Nevertheless, we recognise that an ontology is only a possible part of a KG. We consider ontologies expressed with the Semantic Web standards RDF, RDFS and OWL [4, 5]. Therefore, we implement the RDF representation of OWL axioms. Our framework is based on Description Logics for formal knowledge representation.

2.1. Ontology Learning

Collecting and modelling knowledge to build an ontology is known as knowledge acquisition. Such a process is complex, time-consuming, and requires access to domain experts rarely available in practice. The challenges inherent to the knowledge acquisition task led to the knowledge acquisition bottleneck [6]. In the literature, we find different terms denoting the tools and approaches to tackle knowledge acquisition, such as KG construction [7], ontology engineering [8], and OL [9].

The OL task aims at automatically constructing ontologies from existing resources [9]. Concentrating on OL from text, current approaches focus on tools to support the ontology engineering process or use case-specific methods in which the human still has a predominant role [10]. This paper uses the OL from text framework OLAF [10] and its implementation[1] to construct fully automatic OL pipelines. We implement pipelines and investigate the use of LLM for OL.

2.2. LLMs for Ontology Learning

The joint use of LLMs and KG is a widely discussed topic in the literature [1]. We focus on the LLM-based approaches for OL. Grapher [11] is one of the first works to extract RDF triples from text using LLMs. A trainable system is used for the tasks of node and edge generation. More recently, LLMs4OL [12] aims to evaluate different LLMs for specific OL tasks. The latter work provides information on which LLM to use for the OL and shows the positive impact of fine-tuning. The authors of [13] also demonstrate the latter fine-tuning benefit for KG completion tasks. Another approach is AutoKG [14], where multiple LLM-based agents build

[1]https://github.com/wikit-ai/olaf (Accessed on October 11, 2024)

KGs. The authors evaluate different OpenAI models on specific tasks with zero-shot and few-shot prompting. Their results stay within the performance of the current state-of-the-art models. Conversely, the authors of [15] obtained good results with an LLM-based iterative process for KG construction manually evaluated task by task. Similarly, PiVe [16] offers a small-size LLM-based iterative verification of a generated KG. Therefore, it could be used in addition to the previously described methods to improve them.

Nevertheless, some issues persist with the works described. Datasets barely represent real-world use cases and are task-focused. Evaluations are also task-focused. Hence, the complete OL process or the ontologies created are not discussed. Furthermore, the methods only use the LLMs' internal knowledge instead of contextualisation. We aim to address these points with our work. We are interested in unsupervised OL from text, so we use LLMs with zero-shot and few-shot prompting depending on the task, rather than fine-tuning.

3. Contributions

LLMs are widely used to generate text and structure data [1]. Hence, they are appropriate candidates for moving from plain text to OWL axioms. However, we argue that this task should not be seen as a simple translation. In this section, we demonstrate our hypothesis with an in-depth analysis of applying LLMs in different ways and highlighting the emerging advantages, disadvantages and behaviours for OL. This section describes our approach and how we handle the OL discriminative sub-task of axiom extraction.

3.1. Evaluation approach

OL from text derives an ontology from a textual corpus. Hence, the perfect data to evaluate this task would be a corpus with one or more expected associated ontologies. However, we could not find such a text/ontology combination in the literature. Thus, we build our two own to assess the pertinence of LLMs for the OL task.

For the first use case, we choose an ontology as a reference and apply a reverse engineering process. We generate a text describing the ontology from its textual labels and use different OL techniques to produce ontologies from the same generated text. We compare them with each other and with the reference ontology. For the second use case, we select another text and manually build an ontology based on the ontology development 101 methodology [2]. We use different OL techniques to produce ontologies from the same selected text and compare them with each other and with the manually built ontology. The three ontology pipelines compared in both cases are the following:

- **LLM Text2OWL** lets the LLM generate the ontology as a single task based on a prompt
- **OLAF LLM** builds an ontology with a list of all LLM-based components
- **OLAF no-LLM** creates an ontology using only components that are not LLM-based.

Evaluating a learned ontology is a complex task addressed in various ways in the literature [10]. We use gold standard evaluation techniques as we have data combining text and reference ontology. The gold standard evaluation compares the learned ontology with a reference ontology.

This reference ontology is an idealised outcome of the learning algorithm, usually previously created and known as the gold standard [17]. These approaches are based on ontology mapping or ontology alignment. Therefore, we manually align the learned ontology with the reference ontology to assess three standard ontology evaluation criteria: conciseness, completeness and correctness [17]. Conciseness ensures that the ontology does not contain irrelevant elements to the domain. We compute it as the proportion of common elements between the learned ontology and the reference ontology in relation to the number of elements in the learned ontology. Completeness measures if the domain of interest is appropriately covered in this ontology. We compute it as the proportion of common elements between the learned ontology and the reference ontology in relation to the number of elements in the reference ontology. Finally, correctness ensures the ontology compliance to defined gold standards. We compute it as the harmonic mean of conciseness and completeness.

3.2. Axiom extraction

The axiom extraction task remains challenging and little addressed in the literature [10]. However, this step truly distinguishes a graph of data from a KG. In OLAF, the axiom extraction component is dedicated to extract axioms from the Knowledge Representation (KR), which represents knowledge as a set of concepts and relations. We extended OLAF with two approaches for axiom extraction, one rule-based and another prompting an LLM. The latter prompt only restricts the output format for the axioms and provides the KR constructed in the previous tasks as context. The former rule-based axiom extraction approach is based on the idea of Ontology Design Patterns (OPDs) [18]. We assume an ontology is constructed with a purpose, directly impacting the kind of axioms to define. For our implementation, we decided to rely on the OWL language. The OLAF OWL axiom extraction component constructs an OWL RDF graph from the previously extracted concepts and relations. The user provides functions generating OWL RDF triples. Some straightforward examples of such OWL axiom generator functions are creating each KR concept as OWL class and each KR relation as an object property. With the same spirit as the ODPs, we can create particular OWL constructs based on the KR.

The OLAF OWL axiom extractor component also checks for the generated OWL ontology logical consistency. The process generates the full OWL RDF graph and runs a reasoner. It stops if the reasoner finds no logical inconsistency or unsatisfiable classes. Based on whether a logical inconsistency or an unsatisfiable class is found, the OLAF OWL axiom extraction process will iteratively prune axioms until a consistent ontology is reached.

4. Experiments

This section introduces our corpus construction process before describing each OL pipeline we compare in the next section. The experiment implementations and prompts are available on our companion repository for extensive details.

4.1. Pizza Ontology textual description

We consider the Pizza Ontology[2] as one of our reference ontology. It has been created for an OWL tutorial with the Protégé software[3] and introduces basic pizza concepts such as ingredients, pizza categories, and the most popular pizzas. We use an LLM to generate descriptive text about the Pizza Ontology. We extract the Pizza Ontology's RDFS labels as a list of strings, removing the OWL constructs. These text labels feed the prompt context to generate the textual description. The prompt instructs to generate a text describing pizzas with all the given labels. Prompt tuning techniques are discussed in section 4.6. We select the *GPT-4* OpenAI model[4] for this task and set the *temperature* to 0. LLMs are not deterministic, so we performed several runs with the same prompt. The text obtained was identical for 5 successive generations. Each paragraph is a document from the corpus. The corpus comprises 10 documents with an average of 48 words each.

4.2. Manual construction of the metallic surface defect detection ontology

We consider the description of the C10-DET dataset [19] as our reference text. We extract the text (available in the companion repository) as 11 documents with an average of 44 words each. We manually build an ontology following the ontology development 101 methodology [2]. All the steps are detailed below.
Step 1: To determine the domain and scope of the ontology, we rely on the corpus describing metallic surface defect detection. The ontology must cover the entire scope of the text without adding information not present in the text. Domain knowledge is not added so as not to disadvantage ontologies learned only from text. This ontology aims to trace the cause of an identified defect, so we're focusing on this application.
Step 2: We do not reuse existing ontologies to be consistent with the ontology learning process. However, linking our ontology to a higher-level ontology could be a guarantee of quality for manually created or learned ontologies. We will explore this in future work.
Step 3: We enumerate all the terms of the text that could be important in the ontology.
Step 4: We combine the top-down and bottom-up approaches for classes and hierarchy definition. We identify broad categories and the most specific defect information and reconstruct the intermediate hierarchy by grouping terms into generic classes.
Step 5: We define the properties of classes by selecting the properties in the text and determining which class they describe.
Step 6: We define the facets of the slots with the domain, i.e., the classes to which a slot is attached, and the range, i.e., allowed classes for slots.
Step 7: We choose not to create instances on our ontology and stay at the schema level with classes and properties.

[2]https://github.com/owlcs/pizza-ontology/tree/master (Accessed on October 11, 2024)
[3]https://protege.stanford.edu/ (Accessed on October 11, 2024)
[4]https://arxiv.org/abs/2303.08774 (Accessed on October 11, 2024)

4.3. LLM Text2OWL

When determining whether LLMs are suitable for OL, one of the first things to investigate is having the LLM directly generate the ontology from the text. We choose the *GPT-3.5-turbo-16k* OpenAI model and *Mistral 7B* model, ensuring that the input and output context windows were large enough to contain all the necessary information. For example, we prefer *GPT-3.5-turbo-16k* to *GPT-3.5-turbo* because *GPT-3.5-turbo* can generate a maximum of 4096 tokens, which is not enough to produce the ontology directly from the text. Otherwise, the result is truncated. We do not use *GPT-4* to limit bias as it was already used for the Pizza Ontology textual description generation. We choose the *Mistral 7B* model to compare the performance of LLMs for the OL task according to their size and to propose a solution with an open-source LLM. The *temperature* for both models is set to 0. We instruct the model to generate an OWL ontology in the turtle format provided a namespace and the textual description as context. To limit the effect of the LLMs' non-deterministic behaviour, we carried out 10 trials. The generated ontologies were always the same for the Mistral model. We kept the ontology generated identically 7 times for the pizza ontology and 9 times for the defect detection ontology with the OpenAI model. We discuss whether the LLMs already know the ontologies we are asking them to generate and the impact on our results in the section 6.1.

4.4. OLAF LLM pipeline

We consider OL a process comprising several tasks [10]. Hence, we create a pipeline entirely relying on LLM components. We use the framework OLAF [10] extended with LLM-based components for each sub-task. For consistency in our comparisons, we keep *GPT-3.5-turbo-16k* and *Mistral 7B* with *temperature* 0. First, the LLM extracts candidate terms that could be concepts for each document. It then enriches these candidate terms by adding lists of synonyms. The LLM groups enriched candidate terms to form concepts. The LLM is then asked to create hierarchy triples among concepts. These steps are repeated for relations. For all the previous steps, the Pizza Ontology and metallic surface defect detection description extracts are in the prompt to contextualise the text generation. Finally, the LLM generates the OWL axioms based on the lists of concepts and relations. The prompts are discussed in section 4.6, along with the method used to obtain them.

4.5. OLAF no-LLM pipeline

We build a pipeline without any LLM component to complete the LLM performance evaluation for OL. We select the most suitable no-LLM components available in OLAF [10]. The candidate terms extraction uses TF-IDF [20] scores with a threshold. These candidate terms are enriched using WordNet [21]. They are grouped as concepts with Agglomerative Clustering (AC)[22] and the *sentence-t5-base* Sentence Transformer embedding model [23]. We compute hierarchies based on the Subsumption algorithm [24]. The relation candidate terms are extracted from *VERB* POS-tags, enriched using WordNet and grouped as relations with AC. OWL axioms are extracted with the rules described in section 3.2.

4.6. Prompts tuning

Prompts are natural language instructions that provide context to guide model output without altering parameters[5]. Therefore, the form and content of the prompt can directly affect the result produced by the LLM. Prompt engineering has become essential to design task-specific instructions according to various techniques. We rely on techniques capable of handling new tasks without extensive training, i.e., zero-shot and few-shot prompting[5]. While zero-shot prompting relies on a precise task description in the prompt, few-shot prompting adds a few input-output examples to induce an understanding of the given task. As the examples imply additional tokens in model input, it can become prohibitive for longer text inputs. LLMs have fixed-size context windows divided between input and output tokens. For example, if a model has a context window of 10k tokens and the input is 6k tokens, the output cannot exceed 4k tokens. The prompt must also respect the specific format supported by the chosen model. Once the first version of the prompt is established, it is tested and modified iteratively and empirically to adapt the output.

In our work, we create a variety of prompts. They are all available in our companion repository. The first prompt we develop is generating the pizza ontology's textual description. As we are querying the *GPT-4* model, we refer to the OpenAI guidelines[6]. We prefer zero-shot to few-shot prompting so that the number of tokens in the input does not restrict the output of the LLM. We then propose the prompts to generate an ontology directly from a textual description. There is one prompt for the *GPT-3.5-turbo-16k* model and one for *Mistral-7B* based on the model guidelines[7]. We prefer zero-shot to few-shot prompting again to leave enough tokens to the LLM output. A large number of prompts are required for the LLM pipeline. The one-shot prompting strategy is applied to extract concept and relation candidate terms. An LLM call is made on each document, and the example added in the prompt is small, which does not penalise the model output. Term enrichment is achieved by calling the LLM for each term with one-shot prompting. We tried adding more examples, but this did not improve the generation of the examples tested. To extract concepts and relations, we develop a zero-shot prompting strategy, as we list all the candidate terms in the prompt, which already represents a large number of tokens. For the Mistral model, an example of the expected output data format must be added. We tried different strategies for the hierarchisation task. Despite the large number of input tokens with the list of all concepts, we applied a one-shot prompting strategy because the results in zero-shot were too bad. In this case, the LLM poorly understood the task without an example. Finally, axiomatisation is managed using zero-shot prompting because the input provided and the expected output already involve a large number of tokens.

5. Results

The section presents a quantitative analysis of the created ontologies and compares their alignments with the Pizza Ontology and the defect detection ontology. We conclude the section with an in-depth study of the relevance of each OL sub-task LLM.

[5]https://arxiv.org/abs/2402.07927, Accessed on October 11, 2024.
[6]https://platform.openai.com/docs/guides/prompt-engineering, Accessed on October 11, 2024.
[7]https://docs.mistral.ai/guides/prompting_capabilities/, Accessed on October 11, 2024.

5.1. Ontologies evaluation

To evaluate the created ontologies, we begin with a quantitative analysis. In Tables 1 and 3, we count the OWL named classes, object properties, named individuals and RDFS *subClassOf* tuples for each ontology. We then evaluate conciseness, completeness and correctness through manual alignment with the reference ontologies in Tables 2 and 4 for the two use cases.

Pizza ontology The OLAF LLM ontology built with the OpenAI model has the closest approximate size to the Pizza Ontology. The number of OWL-named classes and RDFS *subClassOf* tuples are almost similar. The LLM Text2OWL ontology built with the OpenAI model does well on the OWL object properties and named individuals but falls short for OWL-named classes and RDFS *subClassOf* tuples. It suggests that this ontology is more straightforward than the others. The same trend appears for the Mistral model pipelines. A few elements are extracted compared to the Pizza Ontology, except object properties for the OLAF LLM pipeline. For the OLAF no-LLM ontology, although the number of OWL named classes and object properties is consistent with that of the Pizza Ontology, many more OWL named individuals and RDFS *subClassOf* tuples were extracted, suggesting this ontology is noisy.

Table 1
Pizza ontologies' OWL axioms counts per type.

Counts	Pizza Ontology	Text2OWL OpenAI	Text2OWL Mistral	OLAF LLM OpenAI	OLAF LLM Mistral	OLAF no-LLM
OWL named classes	97	36	23	99	23	111
OWL object properties	8	2	0	77	13	22
OWL named individuals	5	27	0	97	1	343
RDFS *subClassOf* tuples	141	33	23	114	5	390

We manually align the created ontologies with the Pizza Ontology and detail the metrics in Table 2 to explore the results further. The LLM Text2OWL ontologies of both LLMs perform well for class extraction, with an advantage for OpenAI regarding completeness. Every class found is in the Pizza Ontology, but two-thirds of the Pizza Ontology classes are missing. No individuals are found, which suggests that the LLMs tend to create classes rather than instances when no precise instructions are given. Text2OWL OpenAI pipeline performs best for object properties because the Pizza Ontology contains very few. They are all found, though the LLM generates other ones. On the contrary, the Mistral one fails to generate object properties. For both LLMs, half of the extractions for the *subClassOf* pairs are relevant, but the completeness is very low as there are few extracted pairs. These mixed scores suggest that LLMs can generate relevant and precise OWL axioms, but a significant part of them is missing.

The OLAF LLM OpenAI ontology obtains the best correctness for 4 of the 5 aligned object types. Half of the Pizza Ontology classes are correctly extracted. The Mistral model achieves

Table 2
Results of aligning the Pizza Ontology with the created ones.

Metrics	Text2OWL OpenAI	Text2OWL Mistral	OLAF LLM OpenAI	OLAF LLM Mistral	OLAF no-LLM
Classes conciseness	**1.000**	**1.000**	0.567	0.826	0.387
Classes completeness	0.379	0.253	**0.579**	0.200	0.453
Classes correctness	0.550	0.410	**0.573**	0.322	0.417
Individuals conciseness	0.000	0.000	**0.011**	0.000	0.006
Individuals completeness	0.000	0.000	0.200	0.000	**0.400**
Individuals correctness	0.000	0.000	**0.020**	0.000	0.012
Classes and individuals conciseness	0.540	**0.870**	0.557	0.609	0.130
Classes and individuals completeness	0.239	0.141	0.380	0.099	**0.415**
Classes and individuals correctness	0.332	0.242	**0.452**	0.170	0.198
Object properties conciseness	**0.250**	0.000	0.065	0.000	0.136
Object properties completeness	**1.000**	0.000	0.625	0.000	0.375
Object properties correctness	**0.400**	0.000	0.118	0.000	0.200
SubClassOf pairs conciseness	0.515	0.565	0.237	**0.800**	0.012
SubClassOf pairs completeness	0.066	0.051	**0.105**	0.016	0.023
SubClassOf pairs correctness	0.117	0.093	**0.146**	0.030	0.015

greater conciseness but much less completeness. Some individuals are generated by the OLAF LLM OpenAI pipeline, even if their quality is still inferior. The Mistral model does not extract any. The same applies to properties, where only the OpenAI pipeline finds most of the few Pizza Ontology object properties but adds many others. For the *subClassOf* pairs, only some extracted are aligned with the Pizza Ontology, which gives a lower score. Thus, the OLAF LLM OpenAI pipeline produced the ontology closest to the Pizza Ontology.

Though the OLAF no-LLM ontology performs poorly at first glance, just under half of the Pizza Ontology classes and individuals are correctly extracted. It is closer to a third for object properties. The OLAF no-LLM pipeline extracts many axioms, so the lower correctness scores are often linked to conciseness. Indeed, few of these axioms are ultimately relevant. It likewise applies to completeness in the case of *subClassOf* pairs. Therefore, the OLAF no-LLM pipeline is producing mixed results. It is promising, but the extracted axioms must be filtered to reduce noise.

Defect detection ontology The OLAF LLM ontologies built with both models have the closest approximate size to the defect detection ontology. The LLM Text2OWL ontologies are the same with both LLMs and contain very few elements. This suggests that the generated ontologies are too straightforward compared with the reference ontology, validating the behaviour already observed with the Pizza Ontology. For the OLAF no-LLM ontology, many elements are extracted.

It confirms the hypothesis made with the Pizza Ontology that this pipeline produces ontologies that are too noisy.

Table 3
Defect detection ontologies' OWL axioms counts per type.

Counts	Defect Ontology	Text2OWL OpenAI	Text2OWL Mistral	OLAF LLM OpenAI	OLAF LLM Mistral	OLAF no-LLM
OWL named classes	51	1	1	20	26	100
OWL object properties	5	0	0	12	12	34
OWL named individuals	0	10	10	47	0	317
RDFS *subClassOf* tuples	45	0	0	14	7	377

Table 4
Results of aligning the defect detection ontology with the created ones.

Metrics	Text2OWL OpenAI	Text2OWL Mistral	OLAF LLM OpenAI	OLAF LLM Mistral	OLAF no-LLM
Classes conciseness	**1.000**	**1.000**	0.700	0.577	0.210
Classes completeness	0.019	0.019	0.269	0.288	**0.404**
Classes correctness	0.038	0.038	**0.389**	0.385	0.276
Classes and individuals conciseness	**1.000**	**1.000**	0.280	0.577	0.082
Classes and individuals completeness	0.212	0.212	0.269	0.288	**0.654**
Classes and individuals correctness	**0.349**	**0.349**	0.275	0.385	0.145
Object properties conciseness	0.000	0.000	0.083	0.009	**0.088**
Object properties conciseness	0.000	0.000	0.200	0.400	**0.600**
Object properties correctness	0.000	0.000	0.118	0.018	**0.154**
SubClassOf pairs conciseness	0.000	0.000	0.000	**0.143**	0.006
SubClassOf pairs conciseness	0.000	0.000	0.000	0.044	**0.067**
SubClassOf pairs correctness	0.000	0.000	0.000	**0.068**	0.011

We manually align the created ontologies with the defect detection ontology and detail the metrics in Table 4 to explore the results further. The LLM Text2OWL ontologies of both LLMs perform well for class conciseness but poorly for class completeness. The correctness is improved by considering individuals because classes defined in the reference ontology are generated as individuals by the LLMs. Unlike the Pizza Ontology, the LLMs tend to create individuals rather than classes when no precise instructions are given for this use case. Text2OWL OpenAI pipelines do not work for properties and *SubClassOf* pairs as they generate none. These weak scores confirm that the LLMs can generate relevant and precise OWL axioms, but a major part of them is missing. For this use case, the LLM size does not impact the results.

The OLAF LLM ontologies obtain the best correctness for only one aligned object type each, even though they have the closest number of elements to the reference pipeline. More than a third of the defect detection ontology classes are correctly extracted. The Mistral model achieves greater correctness for both classes/individuals and *SubClassOf* pairs extraction, while the OpenAI model performs best for classes and object properties extraction. The OpenAI model does not extract *SubClassOf* pairs. Thus, the OLAF LLM pipelines produced the ontology closest to the defect detection ontology but with room for improvement. Contrary to the Pizza Ontology, the model size does not give a significant advantage to one pipeline over another.

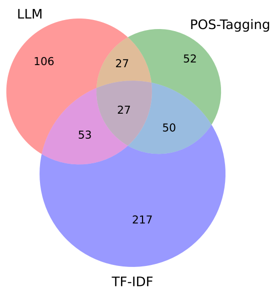
Though the OLAF no-LLM ontology performs poorly at first glance, it obtains the best completeness for all aligned object types. As the pipeline extracts many axioms, the lower correctness scores are often linked to conciseness again. It likewise applies to completeness in the case of *subClassOf* pairs. Therefore, we can conclude in the same way as for the Pizza Ontology: the OLAF no-LLM pipeline is producing promising results, but the extracted axioms must be filtered to reduce noise.

5.2. Analysis by component

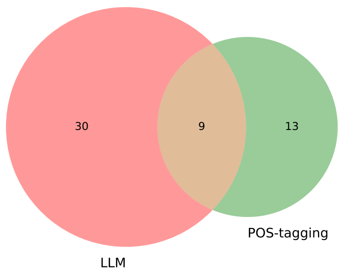
We have shown that LLMs perform well when executing the OL unitary tasks. We now further compare the components available in OLAF one by one. This analysis aims to build a pipeline mixing LLM and non-LLM components to maximise performance while limiting the computational resources consumed. Computational resources are highly related to the model size. Model hosting costs or third-party model providers should also be considered. This detailed analysis is based on the Pizza Ontology use case with the *GPT-3.5-turbo-16k* model. Each component is executed on the same input.

Candidate terms extraction Candidate terms extraction is compared with POS-tagging (extraction of terms according to the *NOUN* tag for concepts and *VERB* for relations), TF-IDF (selection of terms with a score above a chosen threshold) and with LLM (prompt to generate candidate terms in a given text). The results are presented in Figure 1. The POS-tagging approach is restrictive as it can only extract one-word terms. The TF-IDF component extracts a lot for concept candidate terms and requires appropriate post-processing. It is not implemented for relation candidate terms because we can not filter terms specific to relations rather than concepts. The LLM method is, therefore, the best option. We notice a few common terms extracted by the different methods.

Term enrichment Term enrichment is compared with semantic similarity (cosine similarity between spaCy model embeddings), WordNet (extraction of defined synonyms if the term is in the knowledge base) and an LLM (prompt to generate synonyms for each candidate term). For both concept and relation candidate terms enrichment, as depicted in Figure 2, only the LLM finds synonyms for all candidate terms. Furthermore, according to Figure 3, the LLM adds more synonyms on average than other methods. Semantic enrichment is restricted to the vocabulary of the spaCy model used, and Wordnet enrichment is limited to the textual representation of



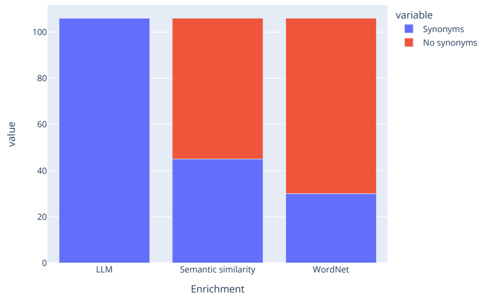
(a) Concept candidate terms.



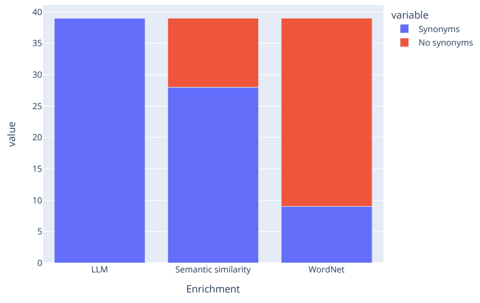
(b) Relation candidate terms.

Figure 1: Venn diagram comparing term extraction methods. The number indicate the candidate terms in common between methods or specific to a method.

the term, which explains the fewer synonyms. The LLM is the best available method for term enrichment.



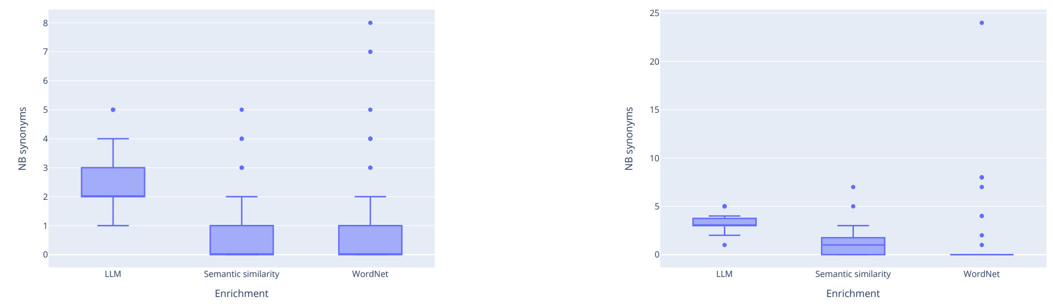
(a) Concept candidate terms.



(b) Relation candidate terms.

Figure 2: Number of candidate terms with and without synonyms for each term enrichment method.

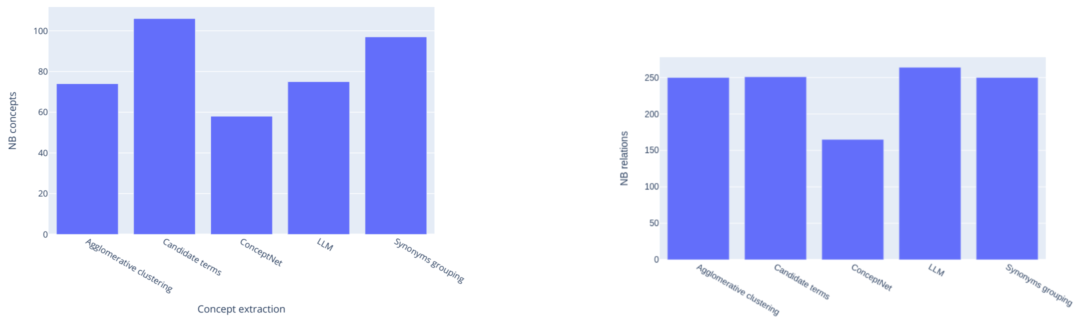
Concept and relation extraction Concept and relation extraction are based on the same methods with AC (AC runs on the embedded terms and the output groups of terms become concepts), candidate terms validation (all candidate terms are converted into concepts), ConceptNet (creates a concept if the term exists in the knowledge base), synonym grouping (groups terms with common synonyms into concepts) and an LLM (prompt to generate concepts and relations from a list of terms). The results are presented in Figure 4. The ConceptNet-based method retains only entities that exist in ConceptNet, which can result in a significant loss of knowledge. Synonym grouping highly depends on the enrichment task. AC is promising and appears to perform similarly to the LLM. Hence, AC should be preferred for these tasks.



(a) Concept candidate terms.

(b) Relation candidate terms.

Figure 3: Number of synonyms by candidate term with average and standard deviation for each term enrichment method.



(a) Concepts.

(b) Relations.

Figure 4: Number of concepts and relations extracted by each method.

Concept hierarchisation Concept hierarchisation is compared with the Subsumption algorithm [24] and an LLM (prompt for generating hierarchies between concepts). The concept hierarchies in Figure 5 have few in common. Nevertheless, the Subsumption algorithm extracts many more than the LLM. It may be due to the sparse data limiting the algorithm's capabilities. It is a disadvantage for the OLAF no-LLM pipeline. The LLM should be preferred for this task with our data.

6. Discussion

This work is a first attempt at evaluating the performance of LLMs for OL. Therefore, it comes with a few limitations. Most of them stem from using LLMs, such as prompting strategies to stabilise the generation, scalability across corpus size, or the final cost of LLM components. This section discusses two particular aspects of LLM usage: their internal knowledge and environmental impact.

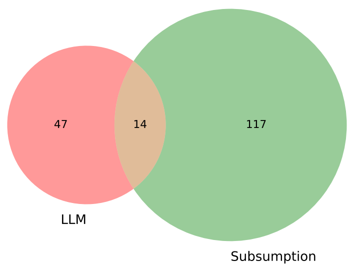


Figure 5: Venn diagram comparing concept hierarchisation methods.

6.1. LLM internal knowledge

The LLM internal knowledge is the knowledge learned during training. This knowledge is not provided during inference but is intrinsic to the model, represented in its weights. Since we prompt LLMs to generate ontologies or parts of ontologies, we ask ourselves whether they carry out this task based on the context and instructions provided or simply use the knowledge acquired during their training. This bias could affect our results, particularly the Text2OWL pipeline for the Pizza Ontology, as the ontology is available on the web.

Table 5
Similarity score between ontologies.

First ontology	Second ontology	Similarity score
Pizza Ontology	Defect detection Ontology	0.937
Pizza Ontology	OpenAI output for pizza ontology	0.942
Pizza Ontology	Mistral output for pizza ontology	0.939
OpenAI output for pizza ontology	OpenAI Text2OLW pizza ontology	0.974
Mistral output for pizza ontology	Mistral Text2OWL pizza ontology	0.962

To mitigate the potential bias, we instruct LLMs to generate the pizza ontology we seek to obtain without any context and measure the semantic similarity between the ontology obtained, the reference ontology and the results of the pipelines. Table 5 presents the cosine similarity computed between the ontologies embedded with the *sentence-transformers/sentence-t5-xl* model. We start by calculating the similarity between the reference pizza ontology and the manually constructed defect detection ontology. We notice a high similarity between them because the two ontologies have similar syntaxes. This score will serve as a landmark for future comparisons. The similarities between the reference Pizza Ontology and the pizza ontologies generated by the LLMs without context are not significantly higher than the landmark score. Regarding the pizza ontologies generated by the LLMs against the Text2OWL pipelines' results, similarity scores are higher than the landmark score but do not indicate that the outputs are similar. This suggests that even if the LLMs have prior knowledge regarding our use case ontology, the techniques applied do not rely only on the LLMs' internal knowledge and produce different and interesting outputs.

6.2. Carbon footprint

The massive development of LLMs has raised several questions, particularly about the amount of resources they consume. These models are particularly energy-hungry because of their size and the infrastructure required. Different stages need to be taken into account to measure their impact, such as the training phase, which has the greatest impact, and the inference phase, which extends over a long period of time[8]. In our work, we want to measure the energy consumed by LLMs to propose responsible solutions so that performance considers ecological criteria. As we are using trained models, we focus on measuring the energy consumed and GreenHouse Gases (GHG) emitted during inference. We measure them for each task we perform and for each model we test. We use the EcoLogits calculator[9], available for different providers, to quantify environmental impact according to different criteria and phases.

Table 6
Energy consumption and greenhouse gas emissions of prompt executions for the pizza use case.

Task	Model	Energy consumption	GHG Emissions
Pizza textual description generation	GPT-4	473 Wh	280 gCO2eq
OWL ontology generation	GPT-3.5-turbo-16k	15 Wh	9 gCO2eq
OWL ontology generation	Mistral 7B	3 Wh	2 gCO2eq
Concept candidate terms extraction for one document	GPT-3.5-turbo-16k	0.4 Wh	0.2 gCO2eq
Concept candidate terms extraction for one document	Mistral 7B	0.2 Wh	0.1 gCO2eq
Relation candidate terms extraction for one document	GPT-3.5-turbo-16k	0.2 Wh	0.1 gCO2eq
Relation candidate terms extraction for one document	Mistral 7B	< 0.1 Wh	< 0.1 gCO2eq
Enrichment for one candidate term	GPT-3.5-turbo-16k	0.4 Wh	0.2 gCO2eq
Enrichment for one candidate term	Mistral 7B	0.3 Wh	0.2 gCO2eq
Concepts or relations extraction	GPT-3.5-turbo-16k	4 Wh	2 gCO2eq
Concepts or relations extraction	Mistral 7B	1 Wh	0.8 gCO2eq
Concepts or relations hierarchisation	GPT-3.5-turbo-16k	11 Wh	7 gCO2eq
Concepts or relations hierarchisation	Mistral 7B	4 Wh	2 gCO2eq
Axiom extraction	GPT-3.5-turbo-16k	11 Wh	6 gCO2eq
Axiom extraction	Mistral 7B	3 Wh	2.4 gCO2eq

Table 6 details each prompt and model's energy consumption and GHG emissions during the usage phase. The Mistral model has a lower environmental impact for the same task than

[8]https://arxiv.org/abs/2309.14393, Accessed on October 11, 2024.
[9]https://github.com/genai-impact/ecologits, Accessed on October 11, 2024.

Table 7
Energy consumption and greenhouse gas emissions of LLM pipelines executions for the pizza use case.

Pipeline	Model	Energy consumption	GHG Emissions
LLM Text2OWL	GPT-3.5-turbo-16k	15 Wh	9 gCO2eq
LLM Text2OWL	Mistral 7B	3 Wh	2 gCO2eq
OLAF LLM	GPT-3.5-turbo-16k	96 Wh	50 gCO2eq
OLAF LLM	Mistral 7B	57 Wh	38 gCO2eq

the OpenAI model. This is directly linked to the size of the models and the geographical areas in which they are hosted (Europe for Mistral, North America for OpenAI). A clear link can be made between the cumulative size of LLM input and output and its environmental impact. Indeed, the generation of the Pizza Ontology textual description has by far the highest impact, with many tokens in input and output. The OWL ontology generation, concept and relation extraction, hierarchisation and axiomatisation processes have a close environmental impact. The number of input tokens is approximately the same, while the number of output tokens varies slightly depending on the instructions given. Other prompts have less impact because they have a much smaller input and output. Table 7 shows the total impact of pipelines using LLMs. Unsurprisingly, the OLAF LLM pipeline has a much greater environmental impact than the LLM Text2OWL pipeline. However, this pipeline produces results that are better and, above all, more explicable with elements added step by step. The model chosen and its size significantly influence the environmental impact. Therefore, a balance must be struck between the technique used and the model chosen to execute it.

7. Conclusion

The community's growing interest in LLMs has questioned their relevance to knowledge extraction. Hence, our work explores the relevance of LLMs for OL from text. We propose two use cases with a reverse engineering process for evaluating OL pipelines by alignment with a reference ontology. We leverage our framework OLAF [10], which implements OL tasks from term to axiom extraction. We demonstrate the pertinence of LLMs for OL with zero-shot or one-shot prompting, particularly when used for specific tasks rather than on an end-to-end basis. The contextual information provided during text generation is a differentiating feature compared to other works in the literature. We show the positive impact of using a larger LLM on one of the use cases, while the results are not significantly different for the second one. This is encouraging for future work, as we can consider using smaller LLMs for the OL task. We also detail the environmental impact of the pipelines developed to propose efficient and responsible solutions. We argue that the environmental impact is mainly linked to the model chosen and the size of the inputs and outputs used. Combining LLMs with other sub-task components creates even more efficient pipelines by limiting the resources consumed according to the application's needs. As future work, we will explore other algorithms for currently critical tasks without LLM and test more LLMs to balance model performance and environmental impact.

References

[1] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, X. Wu, Unifying large language models and knowledge graphs: A roadmap, IEEE Transactions on Knowledge and Data Engineering (2024) 1–20. URL: http://dx.doi.org/10.1109/TKDE.2024.3352100. doi:10.1109/tkde.2024.3352100.

[2] N. Noy, D. Mcguinness, Ontology development 101: A guide to creating your first ontology, Knowledge Systems Laboratory 32 (2001).

[3] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutiérrez, S. Kirrane, J. E. Labra Gayo, R. Navigli, S. Neumaier, A.-C. Ngonga Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. F. Sequeda, S. Staab, A. Zimmermann, Knowledge Graphs, number 22 in Synthesis Lectures on Data, Semantics, and Knowledge, Springer, 2021. URL: https://kgbook.org/. doi:10.2200/S01125ED1V01Y202109DSK022.

[4] R. Cyganiak, D. Wood, M. Lanthaler, RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation, w3c, 2014. URL: https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[5] W. O. W. Group, OWL 2 Web Ontology Language Document Overview, W3C Recommendation, 20122. URL: https://www.w3.org/TR/owl2-overview/.

[6] C. M. Keet, An Introduction to Ontology Engineering, https://people.cs.uct.ac.za/m̃keet/OEbook, 2020. URL: https://people.cs.uct.ac.za/~mkeet/OEbook.

[7] X. Ma, Knowledge graph construction and application in geosciences: A review, Computers & Geosciences 161 (2022) 105082. URL: https://www.sciencedirect.com/science/article/pii/S0098300422000450. doi:https://doi.org/10.1016/j.cageo.2022.105082.

[8] D. L. M. Elisa F. Kendall, Ontology Engineering, Synthesis Lectures on Data, Semantics, and Knowledge, Springer, 2019. URL: https://link.springer.com/book/10.1007/978-3-031-79486-5. doi:https://doi.org/10.1007/978-3-031-79486-5.

[9] M. N. Asim, M. Wasim, M. U. G. Khan, W. Mahmood, H. M. Abbasi, A survey of ontology learning techniques and applications, Database 2018 (2018). doi:10.1093/database/bay101, bay101.

[10] M. Schaeffer, M. Sesboüé, J.-P. Kotowicz, N. Delestre, C. Zanni-Merk, Olaf: An ontology learning applied framework, Procedia Computer Science 225 (2023) 2106–2115. URL: https://www.sciencedirect.com/science/article/pii/S1877050923013595. doi:https://doi.org/10.1016/j.procs.2023.10.201, 27th International Conference on Knowledge Based and Intelligent Information and Engineering Sytems (KES 2023).

[11] I. Melnyk, P. Dognin, P. Das, Grapher: Multi-stage knowledge graph construction using pretrained language models, in: NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications, 2021.

[12] H. Babaei Giglou, J. D'Souza, S. Auer, Llms4ol: Large language models for ontology learning, in: International Semantic Web Conference, Springer, 2023, pp. 408–427.

[13] L. Yao, J. Peng, C. Mao, Y. Luo, Exploring large language models for knowledge graph completion, arXiv preprint arXiv:2308.13916 (2023).

[14] Y. Zhu, X. Wang, J. Chen, S. Qiao, Y. Ou, Y. Yao, S. Deng, H. Chen, N. Zhang, Llms for knowledge graph construction and reasoning: Recent capabilities and future opportunities, arXiv preprint arXiv:2305.13168 (2023).

[15] S. Carta, A. Giuliani, L. Piano, A. S. Podda, L. Pompianu, S. G. Tiddia, Iterative zero-shot

llm prompting for knowledge graph construction, arXiv preprint arXiv:2307.01128 (2023).

[16] J. Han, N. Collier, W. Buntine, E. Shareghi, Pive: Prompting with iterative verification improving graph-based generative capability of llms, arXiv preprint arXiv:2305.12392 (2023).

[17] J. Raad, C. Cruz, A Survey on Ontology Evaluation Methods, in: Proceedings of the International Conference on Knowledge Engineering and Ontology Development, part of the 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management , Lisbonne, Portugal, 2015. URL: https://hal.science/hal-01274199. doi:10.5220/0005591001790186.

[18] A. Gangemi, V. Presutti, S. Staab, R. Studer, Ontology Design Patterns, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 221–243. URL: https://doi.org/10.1007/978-3-540-92673-3_10. doi:10.1007/978-3-540-92673-3_10.

[19] X. Lv, F. Duan, J.-j. Jiang, X. Fu, L. Gan, Deep metallic surface defect detection: The new benchmark and detection network, Sensors 20 (2020). URL: https://www.mdpi.com/1424-8220/20/6/1562. doi:10.3390/s20061562.

[20] K. S. Jones, A statistical interpretation of term specificity and its application in retrieval, J. Documentation 60 (2021) 493–502. URL: https://api.semanticscholar.org/CorpusID:2996187.

[21] G. A. Miller, Wordnet: a lexical database for english, Commun. ACM 38 (1995) 39–41. URL: https://doi.org/10.1145/219717.219748. doi:10.1145/219717.219748.

[22] M. L. Zepeda-Mendoza, O. Resendis-Antonio, Hierarchical Agglomerative Clustering, Springer New York, New York, NY, 2013, pp. 886–887. URL: https://doi.org/10.1007/978-1-4419-9863-7_1371. doi:10.1007/978-1-4419-9863-7_1371.

[23] J. Ni, G. Hernandez Abrego, N. Constant, J. Ma, K. Hall, D. Cer, Y. Yang, Sentence-t5: Scalable sentence encoders from pre-trained text-to-text models, in: S. Muresan, P. Nakov, A. Villavicencio (Eds.), Findings of the Association for Computational Linguistics: ACL 2022, Association for Computational Linguistics, Dublin, Ireland, 2022, pp. 1864–1874. URL: https://aclanthology.org/2022.findings-acl.146. doi:10.18653/v1/2022.findings-acl.146.

[24] H. N. Fotzo, P. Gallinari, Learning "generalization/specialization" relations between concepts: Application for automatically building thematic document hierarchies, in: Coupling Approaches, Coupling Media and Coupling Languages for Information Retrieval, RIAO '04, Le centre des Hautes Etudes Internationales d'Informatique Documentaire, Paris, FRA, 2004, p. 143–155.