UNIVERSITÄT TRIER

Automatic Generation of Useful Knowledge Graphs with Large Language Models

Master Thesis

to obtain an academic degree of
Master of Science (M.Sc.)

Universität Trier
FB II
Natural Language Processing

1. Supervisor:	Prof. Dr. Achim Rettinger
2. Supervisor:	Prof. Dr. Christof Schöch

Submitted on 28.03.2025 by:

Sarah Ackerschewski
Porta-Nigra-Platz 4
54292 Trier
s2shacke@uni-trier.de
Matr.-Nr. 1663770

Abstract

Knowledge graphs (KGs) structure real-world knowledge and are used to retrieve information across various domains. Yet the manual construction of such KGs is costly and time-consuming, requiring the work of domain experts. Large language models (LLMs) show outstanding performance in Natural Language Processing (NLP) tasks and thus offer a promising alternative for generating KGs automatically. Evaluating the quality of KGs also remains underexplored, mostly focusing on either one, accuracy or structure. This thesis explores how well different LLMs, such as GPT-4o, GPT-4o mini, and GPT-4 Turbo, can create high-quality KGs using a two-step approach by first extracting an ontology and then generating the full graph. Additionally, it introduces a new evaluation framework that looks beyond semantic accuracy, considering the structure of the ontology and the graph, as well as real-world usability in question-answering tasks.

The results show that GPT-4o produces the most structured and coherent KG, especially when ontology extraction is treated as a separate step and added as additional input to the KG generation step. However, challenges remain, such as structural inconsistencies and complexity, hallucinations, and incompleteness of knowledge. A comparison with LangChain's LLMGraphTransformer highlights how different generation strategies impact KG quality, emphasizing the importance of prompt engineering and step-by-step refinement. While GPT-4o performed best overall, GPT-4 Turbo struggled with ontology extraction, and GPT-4o mini had lower accuracy. It also highlights the importance of a well-thought-out evaluation framework that includes the inspection of various quality criteria, as focusing on only one or two criteria can result in misevaluations.

Whereas LLMs show great potential for KG generation, issues like hallucinations and high computational costs still pose challenges. Future research could focus on improving the prompt further, refining extraction techniques, and exploring open-source models to make LLM-based KG generation more practical and reliable.

Acknowledgements

I would like to express my deepest gratitude to Marin Jukic at textelligence for providing guidance and support as my external advisor. His expertise and insightful feedback were invaluable in shaping this thesis.
Furthermore, I am extremely grateful to textelligence as a whole for providing the necessary funding and access to OpenAI LLMs in this research. Their generous support made it possible to explore my research question with advanced models, which significantly contributed to the depth and scope of this work.
I am also thankful to Professor Rettinger and Professor Schöch for their great insights and knowledge on the topic.
Finally, special thanks to my family and friends as their belief in me, along with their moral support and feedback, kept me motivated throughout this thesis.

Contents

List of Figures

List of Tables

1. Introduction

Since the emergence of Large Language Models (LLMs) such as GPT-3.5 and GPT-4, much research has been done applying LLMs in various fields. LLMs' ability to deal with natural language and their simple application make them valuable resources in improving the automatic construction of knowledge graphs [36]. Even though LLMs have the ability to deal well with natural language processing (NLP) tasks such as text summarization, they often struggle with factual knowledge and tend to hallucinate information [36]. On the other hand, knowledge graphs (KGs) capture real-world objects and their relations. Thus, they are helpful for downstream tasks that require structure, such as recommendation systems, search engines, or question-answering [7, 42, 56]. Therefore, they contain the factual knowledge that LLMs are sometimes missing.
However, the construction of these KGs is often done completely or partially manually by domain experts, which is an expensive and time-consuming task [19]. Since domain experts could be biased when only having a one-sided perspective on the topic or could make mistakes in general, manual KG construction is also not ideal, considering how time and budget-intensive it is. Especially, generating a KG from unstructured data, such as texts, is cumbersome since it needs to map content from a data source into a KG, and traditional KG construction methods process each data type separately [35, 56]. Traditional machine learning (ML) and deep learning (DL) approaches for KG construction often lack labeled training data, making heuristic methods a necessity for mapping unstructured text to its structured KG representation [19]. Even though LLMs do not require gold labels, generating KGs from unstructured text is a challenging task for these models as well. LLMs are known for their hallucination issues, i.e., making up facts and producing wrong statements when the model does not know the correct answer [26]. Therefore, the quality of the KG can suffer if the LLM hallucinates. Still, using LLMs to create KGs is a promising option to mitigate the problem of needing domain experts to create a knowledge graph. Expert knowledge might be already contained in the LLM due to the training data, which is also in favor of using LLMs to construct knowledge graphs [39]. In this case, extensive and meticulous prompt engineering could reduce hallucinations and help generate a high-quality KG.

Presenting unstructured text in a structured way is beneficial for offering simple readability of the data and thus simplifying access to the information within the text [9, 28]. Consequently, more and more studies have shifted their focus to the automatic generation of KGs using machine learning techniques and, even more recently, LLMs. Here, the idea of using LLMs to create a KG to reduce the efforts of manual generation indicates that LLMs and KGs could enhance each other with reasoning capabilities and structure [36]. In this context, the question of whether

LLMs can construct ”good” or ”useful” KGs can be raised.

Previous research focuses mainly on constructing KGs, but evaluation of the KG is mostly restricted to accuracy. Yet, the quality of KGs is essential for ensuring reliable databases and high-level outcomes in downstream task [7]. Furthermore, the quality is not only restricted to the correctness of the content, i.e., accuracy. Triples in a knowledge graph should be syntactically correct to be useful. Consistency and completeness also play a huge role in quality measurement because they ensure that the KG is free of duplicates and represents the complete knowledge [7]. Although there is no clear definition of a “useful” or “good” KG in the literature, some studies emphasize the “fitness of purpose” [7, 54]. This would mean measuring the quality of the KG based on how well it works for the respective task [7]. Some evaluation frameworks focus on the quality dimensions that can be looked at when evaluating a KG. Nevertheless, specific quality metrics are seldom introduced in these studies. Thus, evaluating the KGs is a difficult research area that remains not completely solved yet [17].
This thesis focuses on building an automatic KG generation system using current LLMs, here GPT-4o, GPT-4o mini, and GPT-4 Turbo, hosted by OpenAI [31–33]. Furthermore, the term ”useful” will be investigated under the assumption of meaning ”fit for use”. Here, various metrics will be introduced and used to measure the different quality dimensions that a graph can be looked at, as defined by [54] and others [7, 42, 55].

To illustrate the overall workflow, Figure 1.1 presents the KG generation approach of this thesis. Wikipedia articles will be given to a large language model to first construct an ontology. The ontology and the articles will then be given to another LLM to build the knowledge graph. Lastly, the generated knowledge graph will be evaluated with an evaluation framework consisting of structural quality metrics, which look at the underlying structure, and content quality metrics to measure the accuracy of the graph.
The purpose of this study is to find out whether it is possible to create ”good” (or ”fit for use”) knowledge graphs from unstructured data, such as text, with the LLMs stated above. In addition, it is investigated whether a two-step knowledge graph generation approach, consisting of ontology extraction and KG generation, results in a higher quality graph than a one-step approach, here the LLMGraphTransformer approach by LangChain [24]. Moreover, adding hierarchy and property extraction steps within the ontology generation process might further improve the quality of the KG. For that, this study introduces a new approach to knowledge graph construction. Only using LLMs, the ontology is extracted and given as additional input to the KG generation model as suggested in [29]. However, the ”basic” two-step KG generation method is extended with a thorough hierarchy and property extraction process within the ontology generation step. Here, the hierarchy between the entities extracted in the first ontology extraction is also created with an LLM. Afterwards, the hierarchy of entities is completed with properties generated with the same LLM. These substeps are added to the ontology extraction process to mitigate the problem of small output windows. But it is also hypothesized that the quality of the ontology improves with more steps, considering that the ontology quality might have an impact on the knowledge graph quality. The one-step knowledge graph approach

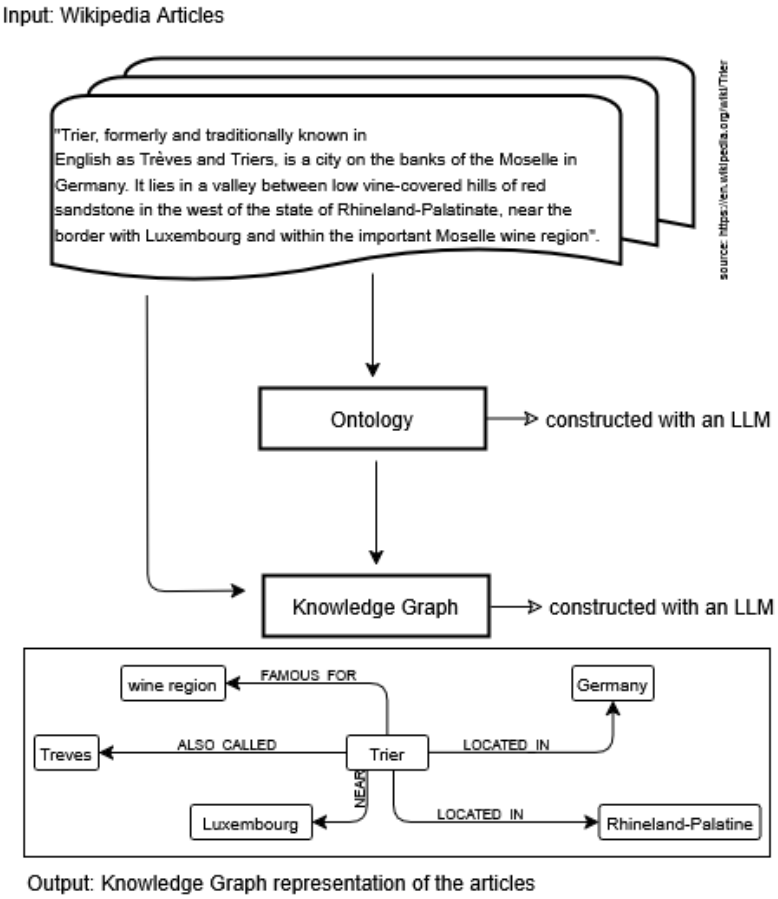

Figure 1.1: General two-step approach from unstructured texts to a knowledge graph using LLMs.

for comparison will be done with the existing method by LangChain, i.e., using the LLMGraphTransformer class [24].
Another new aspect of this work is the direct application of an evaluation framework on the created KG. Prior studies focused either on the construction or the evaluation task. This study will directly evaluate the knowledge graph created with structural and content quality metrics. The selected metrics are not restricted to the correctness quality dimension but also measure completeness, consistency, and conciseness.
This thesis is structured as follows: first, the background of KGs and the automatic construction of KGs with standard and LLM approaches is explored. Previous research about the evaluation of KGs will be looked at more closely to determine what makes a graph ”good” or ”useful”. Moreover, the methodology section will outline the method applied to generate the KG with LLMs from OpenAI. It will also delve more specifically into the data and models used to explore the research questions. Additionally, the evaluation framework and its metrics will be investigated closely to answer the question of the ”usefulness” of a KG. The results section presents the findings of this research, including the metrics from the framework. Finally, the discussion will analyze the results further and derive future research based on the progress and shortcomings of this study. [54].

2. Background

Understanding the background of LLMs and KGs and their applications is crucial before conducting new research in this area. Therefore, the following chapter looks into previous research about knowledge graph definitions, how they can be constructed and evaluated.

2.1 Knowledge Graphs

The term "knowledge graph" first appeared in a blog post by Google in 2012 ([43]) but has evolved with the increase of research in this area [14]. [43] presented the benefits of knowledge graphs in the context of Google. The old matching algorithm to retrieve search results was improved since a KG represents real-world relationships between entities. In that same sense, the representation of relations helps with ambiguities, summarization, and finding related question-answer pairs. Since [43] did not offer a formal definition, the definitions mostly referenced in papers are by [14], and [46]. While the definition by [46] (see 2) emphasized the structure of a KG, [14] (see 1) posed a more general definition focusing on the functionality of a KG.

1. "A knowledge graph acquires and integrates information into an ontology and applies a reasoner to derive new knowledge." [14]

2. "A knowledge graph is a multi-relational graph composed of entities and relations which are regarded as nodes and different types of edges, respectively." [46]

Generally, the consensus is that a KG represents real-world knowledge that can provide structure and model relationships in the form of triples. [35]. Triples are often in the form of a Resource Description Framework (RDF) [50]. RDF is a general data model that represents the data as triples of the form <*subject, predicate, object*> or <*head entity, relation, tail entity*>, respectively [50]. Due to this structure, a knowledge graph not only represents real-world knowledge but also offers contextual information [36].
Some researchers distinguished between the terms "knowledge graph" and "knowledge base". [50] described a KG as a specification of a knowledge base, which is a set of rules, facts, and assumptions. [19] claimed that knowledge bases additionally contain inference. However, most papers use the terms interchangeably, which will also be the case for this thesis.
[36] described four types of KGs, encyclopedic, commonsense, domain-specific, and multi-modal KGs. While encyclopedic KGs capture general real-world knowledge, domain-specific KGs are narrowed down to one topic. Commonsense KGs, such as ConceptNet and ATOMIC, focus on concepts and their relations. Multi-modal KGs contain not only textual data but also multimedia-like images [36].

2.2 Knowledge Graph Construction Pre-LLM

Automatic KG generation was tackled mostly as a task consisting of many subtasks. The subtask's names varied from study to study but mostly overlapped content-wise. [19] revealed two steps for KG construction, entity discovery and relation extraction. Entity discovery was further split into subtasks to extract the relevant entities from a text. First, they said that entities are labeled commonly by using a Sequence2Sequence model in a named entity recognition (NER) task. Then, fine-grained types are mapped onto the entities (entity typing) and entity mentions in the text are linked to the corresponding entities (entity disambiguation/linking). Optionally, multiple KGs can be merged with entity alignment. In the relation extraction step, the semantic links between the discovered entities are labeled [19].
[56] added one more step between the subtasks defined by [19]. Between the entity discovery and relation extraction subtasks, the authors added a coreference resolution step, i.e., finding and grouping an entity and its expressions to prevent duplicates.
[22] also specified three subtasks for KG generation, but these varied slightly from the ones mentioned before. They described knowledge extraction, entity mapping, and data integration into the KG [22].
Even more, steps were defined by [52], where entity discovery and NER are seen as one subtask, then entity linking, relation extraction, and finally, event extraction, which names the relations in a more fine-grained manner.

Before the discovery of (large) language models, such as BERT in 2018 [11], KG generation approaches made use of "traditional" ML models and techniques.
[22] proposed a KG creation framework, T2KG, that generates a KG from natural language text using similarity-based and rule-based approaches for mapping the relations. The construction task was tackled in five steps, entity mapping, coreference resolution, triple extraction, triple integration and finally predicate mapping. First, entities were extracted from Wikipedia articles, aligned to a uniform resource identifier (URI), and then grouped with coreference resolution. Triple extraction was conducted by deriving the relations and their arguments, as defined in linguistics [22]. The goal of the triple integration step was to reduce unconformities and duplicates by grouping identical entities with the chains detected in the coreference step from which then the most common expression was chosen as the representative entity for this group and replaced all other entities of the group in their triples [22]. Finally, the relations from the triples were mapped to the correct relations in the DBpedia KG with a vector-similarity metric and rules. The final mapping was chosen with a priority to candidate pairs created by rules and if such a rule did not exist, then based on similarity [22]. Their KG creation approach resulted in an F1-score of 50,69% and proved to be better than the baselines. [22] also showed that more knowledge can be integrated into an existing KG.
[53] used a Bi-GRU supervised extraction model with multiple attention and a modified cross-entropy loss function to construct a domain-specific KG. The Wikipedia classifications were chosen as the domains, their web pages crawled, and the texts embedded with Glove word embeddings. The embedded texts were then given as input to the framework [53]. They evaluated the construction with the mean of the related paths of degrees 100, 200, and 300, which showed that using the Bi-GRU model with the multiple attention and the improved loss function had the highest

mean with 0.79 compared to other combinations of the framework, e.g., only the Bi-GRU model or the Bi-GRU only with the modified loss function. [53] further verified their results by using a dataset by [40], which consists of 522k training and 172k test sentences. The proposed method was compared to other established state-of-the-art approaches and proved to generate better results in the precision-recall graph.

2.3 Knowledge Graph Construction with LLMs

KG construction approaches advanced over the years since the rise of Language Models (LM) that show context-aware knowledge acquired from pretraining [8]. Since KGs often lack natural language understanding and are difficult to build as they are often incomplete and not generalizable enough for specific tasks, LLMs are introduced as a possible solution to these problems [36]. LLMs also offer easier adaptation to new subtasks due to fine-tuning, transfer learning, and few-shot learning [35]. [23] also highlighted the robustness of LLMs in various KG-related tasks, such as KG completion, ontology refinement, and question-answering tasks.
While some studies distinguished between pre-trained language models (PLMs) and large language models (LLMs), other studies used the terms interchangeably. [51] claimed that LLMs are PLMs with additional parameters. PLMs are models trained in an unsupervised manner with a large corpus and can be fine-tuned to be used for various tasks[1].

Early research studied the implicit knowledge that might already be stored by the LLM, a so-called internal knowledge graph. [39] investigated how much knowledge is already stored in ELMo and BERT with a **LA**nguage **M**odel **A**nalysis (LAMA) probe. They manually converted knowledge from three sources (Google-RE, ConceptNet, and SQuAD) and aligned the T-REx knowledge automatically to cloze statements for a masked token prediction task. If the ground truth token was ranked highly among the other candidates, it was suggested that the model shows more factual knowledge [39]. The results indicated that BERT, especially BERT-large, in commonsense reasoning and in retrieving factual knowledge, depending on the source.
More recently, [9] examined what is stored in the internal KGs of GPT-3 and how this knowledge is represented by extracting said knowledge as a KG using prompt engineering. They identified relations of seed head entities from Wikidata. Then, the relations and head entities were paraphrased to generate more relations because the performance of the LLM possibly increases [9]. Given the head entity and the relation, the model was prompted to generate the tail entities. In this context, the authors also emphasized in the prompt that the LLM should output "Don't Know" for unsure answers to guarantee that the graph is factual [9]. The method yielded a precision value of 83.3%, which exceeded the precision of a pure greedy algorithm by about 30%, indicating what kind of implicit knowledge is stored in the LLMs [9].

In this context, [26] investigated the reasoning ability of the internal KG learned during the pretraining of LLMs. Their first research question explored how accurately LLMs can remember the implicit knowledge with a tail entity prediction and

[1](Remark) Since most studies used the term LLM for models that other definitions would characterize as PLM, I will also use the term LLM for all types of pre-trained language models.

a relation prediction task. Both tasks were performed without any additional context, so the LLM could only rely on the information from its internal KG [26]. The tail entities or relations of 100 random triple entries from DBpedia were masked depending on the prediction task at hand. The LLMs put to the test were text-davinci-003, ChatGPT, and GPT-4. The results were then evaluated with hard and soft accuracy. Hard accuracy only accepted the triple output if it was the same as in DBpedia, while soft accuracy accepted triples if the relation or tail entity was factually correct. Results showed that all models achieved high soft accuracy, i.e., produced mostly factually correct relations/entities but not the exact relations/entities as in DBpedia [26]. Overall, the LLMs recalled information about tail entities from their internal KGs better than relations. [26]'s second research question investigated whether LLMs can reason with their internal knowledge. The LLM was instructed to predict the relation between two entities given additional context from Wikipedia [26]. The authors used the same models as for the first research question. GPT-4 produced the best output because it followed the instructions more closely than the other two models, i.e., showing better reasoning capabilities [26].
Another more recent study by [21] investigated the relation between the knowledge in an LLM and the knowledge in its pre-training. Multiple questions were important to the authors, i.e., how the LLM can answer questions about the content of the knowledge it stores, what kind of knowledge this is, and how the knowledge that is rare in the pre-training dataset can be seized more effectively. For this, they considered a closed question answering (QA) evaluation of the LLM in a few-shot setting. Using various pre-training datasets, e.g., OpenWebText and Wikipedia, and QA datasets, such as TriviaQA, entities were linked, and the LLM was leveraged to map documents to question-answer pairs [21].

Automatic KG generation with LLMs in only one step might not work well due to the possible lack of commonsense knowledge [12]. Therefore, most approaches use LLMs in the various subtasks of the construction task.
In the early stages of leveraging LLMs for KG generation, [2] transformed a commonsense acquisition task into a KG construction task. They used existing knowledge triples for training to provide the KG structure to the model, COMET, which was based on the transformer architecture of GPT. COMET was tested with the ATOMIC and the ConceptNet datasets and compared with an LSTM baseline [2]. The results showed that COMET performs well with a BLEU-2 score of 95.25 compared to the score of the LSTM with 60.83. COMET also produced a low perplexity score (4.32) showing that GPT can learn representations that are transferable to generating natural language commonsense knowledge [2].

[15] saw KG construction as a two-step task. They used the CCNR (Classical Chinese NER-RE Dataset) to extract the named entities with a model based on BERT, such as BERT-base, RoBERTa-base, Guwen-BERT, and BERT-ITM. The NER model served as the encoder for a relation extraction model, which then used BERT with Pooling to obtain fixed-size vectors of input sentences and entities and capture their semantic meaning for predicting relationships[15]. Since Guwen-BERT was pre-trained on a classical Chinese corpus, it outperforms the other models in both tasks. Their RE model containing semantic information also performs better than others, showing that [15] improved the existing KG generation pipeline.

[30] highlighted the importance of ontology alignment to structure knowledge in a domain. They leveraged BERT "CLS", BERT mean token, SBERT, and Fasttext for a text similarity task thus aligning two job ontologies, ESCO and O*NET. The ontology layers were treated as a "bag of occupations" containing labels and descriptions [30]. The matching was done on the schema level, i.e., dealing with concepts and not instances, and at the element level, where entities were considered [30]. Since there was no ground truth, they used the mean reciprocal rank (MRR) to measure if the matches were correct. Additionally, [30] had a subset of the output annotated manually. The results showed that Fasttext embeddings have a high MRR but low coverage of occupations, while SBERT had a high MRR and more coverage. BERT matched with a low MRR but had a higher coverage than SBERT and Fasttext [30]. However, the authors claimed that CLS tokens were not useful sentence embeddings and that, overall, SBERT was the most promising ontology alignment system with high-quality embeddings.

[13] explored using reinforcement learning for finetuning a T5 model, large and base, for KG generation. They tested two tasks, text-to-graph and graph-to-text, to linearize the graph by applying self-critical sequence training (SCST) for both tasks. The model processed the triples by linearizing them in order of the list given by the datasets WebNLG+2020 and TEKGEN. The graph-to-text model using T5 large showed the best performance with SCST training, while text-to-graph performed worse in the reinforcement setting and instead yielded better results with cross-entropy, showing that reinforcement learning is an option to be explored in KG generation tasks [13].

[16] proposed an automatic KG generation approach trying to give the LLM minimum information input, i.e., an "initial" prompt and a few example entity pairs. The initial prompt was rephrased so that the prompts varied syntactically but still inferred the same semantic relation. Then, the entity-pair examples were sampled and added to the prompt [16]. Entities that fulfilled all the prompts were searched and the knowledge for the KG was selected by enumerating all possible pairs of entities and picking the top-k entity pairs with the highest confidence scores. Candidate entity pairs were also chosen by minimum individual log-likelihoods weighted and averaged across different prompts. [16] took relations from ConceptNet and LAMA and created new relations in none of these datasets. Their framework used RobertaNET and, for the instructions, GPT-3. Based on human evaluation from AmazonTurk, they found that LLMs encode knowledge better than smaller ones and that better pretraining improved knowledge learning and storage. The suggested framework was able to extract knowledge of new relation types as well as pre-existing knowledge [16].

[28] proposed Grapher, a KG construction system where a pre-trained LLM was fine-tuned for entity extraction, and then the relations of these entities were identified. They formulated a Sequence2Sequence task for the pre-trained model, here T5 large. An option to use an LSTM/GRU similar head to construct these relations as a sequence of tokens was added for the edge generation. Otherwise, it was possible to predict the relations with a classification head. WebNLG+ 2020, TEKGEN and NYT corpus were the datasets used for training and testing. [28] evaluated their model with precision, recall, and F1-score based on the exact, partial, and strict match. The triple had to be the same, but the type was irrelevant for an exact match, while the same triple and element type were required for a strict match.

When candidate and reference triples only matched in some regards, it was considered a partial match [28]. Overall, the edge generation with a GRU-based head yielded better results than the edge classification head if the dataset was big enough for training the GRU head. Furthermore, Grapher outperformed state-of-the-art KG construction approaches [28].
[29] introduced a benchmark for relation extraction with an ontology as additional input. Their approach, Text2KGBench, took an ontology to specify concepts, relations, domains, etc., and a text corpus containing the facts that the ontology expresses. Furthermore, training examples were added as input consisting of pairs of sentences and facts [29]. The ontology and the training examples were added as guidance for the model, here Vicuna-13B and Alpaca-LoRA-13B, regarding what relations and facts to extract. Wikidata-Tekgen and DBpedia-WebNLG were chosen as the datasets because they contain multiple ontologies. The precision, recall, and F1 scores calculated with the ground truth triples were low for the different model/dataset combinations due to object/subject hallucinations. The ontology conformance, i.e., the percentage of LLM output triples that conform with the input ontology, on the other hand, was higher for all combinations demonstrating that Text2KGBench still performed well [29].
AutoKG, proposed by [6], extracted nodes from text blocks based on keywords with GPT-4 and GPT-3.5 and an ada-embedding. They focused on a simplified version of a KG without triples, where the keywords could correspond to nodes and the undirected weight value of the keyword represents the edges. They clustered text blocks and their corresponding embeddings through k-means and spectral clustering. The LLM then filtered the keywords from a specified number of text blocks [6]. To construct the graph, an initial graph was generated with a weight matrix determined by the similarity between the embedding vectors of the text blocks. The relation between keywords and the text blocks was built based on how close a text block was to a keyword, thus creating positive examples if they were close to a keyword and negative ones if far away [6]. When text blocks were classified as positive, their association weight, i.e., relation, was calculated with Laplace learning semi-supervised. [6] verified the effectiveness of their KG construction approach with a hybrid search, which not only included text blocks relevant to the query but also information from the KG, thus improving the understanding of the LLM.
[5] prompted GPT-3.5 iteratively to extract candidate triplets from phrases and perform entity/predicate resolution and schema inference. Entity resolution was conducted to combine entities referring to the same concept, whereas predicate resolution combined relations that referred to the same predicate between the entities. Then, they cluster based on the label similarity of the entity or relation pairs. The model was prompted to return semantically identical entities and relations from the clusters. Given the entities and relations identified in the entity/predicate resolution step, the KG schema was inferred [5]. At each iteration, the model performed hypernym generation and hierarchical agglomeration, which combined the hypernyms and relations. This helped remove duplicate entries. The experiments were conducted with the Sardegna Turismo website, which was manually annotated for evaluation. Their approach managed high precision scores for all tasks and an F1 of 95.92% for the entity generation task, proving that a zero-shot setting and open domain work well with GPT-3.5 [5].
More recently, [25] suggested the integration of retrieval-augmented generation (RAG)

for KG retrieval. They used prompt engineering to effectively instruct the LLM to first extract the entities. Then, subgraphs were constructed based on the entities and an input question. The relations and entities were expanded with a beam search and filtering. Finally, a reasoning prompt was applied so the KG explained each step of its reasoning, while also encouraging it to not only use the implicit knowledge but also form new links between the nodes [25]. Tested on the QALD10, Creak, Mintaka, CWQ, and ExplainCPE datasets, the framework outperformed standard input-output prompts and CoT prompts showing more reliable and accurate reasoning [25].
In [23] LLMs were leveraged to create ontologies using competency questions (CQs) as ontologies are the basis for a KG. They used literature research of publications in biodiversity research using deep learning approaches as their data. The CQs were first generated with ChatGPT-3.5 and then evaluated by human experts [23]. Next, the ontology was created with the overall concepts as entities and relations extracted from the CQs. The models used were LLaMA-2-70B, Mixtral-8-70B, and Falcon-40B, and the final ontology was built with the PROV-O ontology [23]. For the KG generation, the LLM was instructed to use the CQ answers with their corresponding questions and the LLM-generated ontology to map entities, relations, and concepts to the KG. The authors evaluated the KG with LLMs that should return scores between 0 (no alignment) and 10 (complete alignment with ground truth). Furthermore, human annotation of five research articles was conducted. Overall, 42 disagreements out of 200 CQ answers between the human annotators and the LLM indicated that the proposed method created a semi-consistent KG [23].
[12] also constructed an ontology and the KG. However, the authors wanted to create a more fine-grained KG covering only a specific topic instead of a broad scope of themes, therefore, they focused on EV batteries and the Hamas attack on Israel as the two domains. The ontology construction was done in two steps. First, the entities were identified from Wikipedia pages, where *"Battery(Electricity)"* and *"Vehicles"* were the starting points from which more entities were captured by collecting subtrees from Wiki Category [12]. Wrong relations were detected and removed based on parent-child similarity with an all-MiniLM-L6-v2 pre-trained transformer. After the entity ontology construction, the relation ontology was built using a prompt specific to the themes and adding the entities for which a relation is searched for [12]. The model outperforms the baselines in the entity ontology construction task for both datasets. [12] showed that their KG, ThemeKG, was more fine-grained for the EV battery topic than WikiData.

2.4 Evaluation of Knowledge Graphs

Most of the previously mentioned studies measured the quality of their constructed KGs with common evaluation metrics such as accuracy or F1-score. However, the quality of a KG should also be assessed with various other dimensions to guarantee usability in the downstream application of the KG [50]. Therefore, the following studies investigated how KGs can be evaluated in a more fine-grained way.

[54] provided the basis for most research on the topic by introducing linked data quality. The authors described that the quality of data is based on how "fit for use" it is and introduced 18 quality dimensions, which are different factors that influence data quality. The definition of "fit for use" and the dimensions were a breakthrough

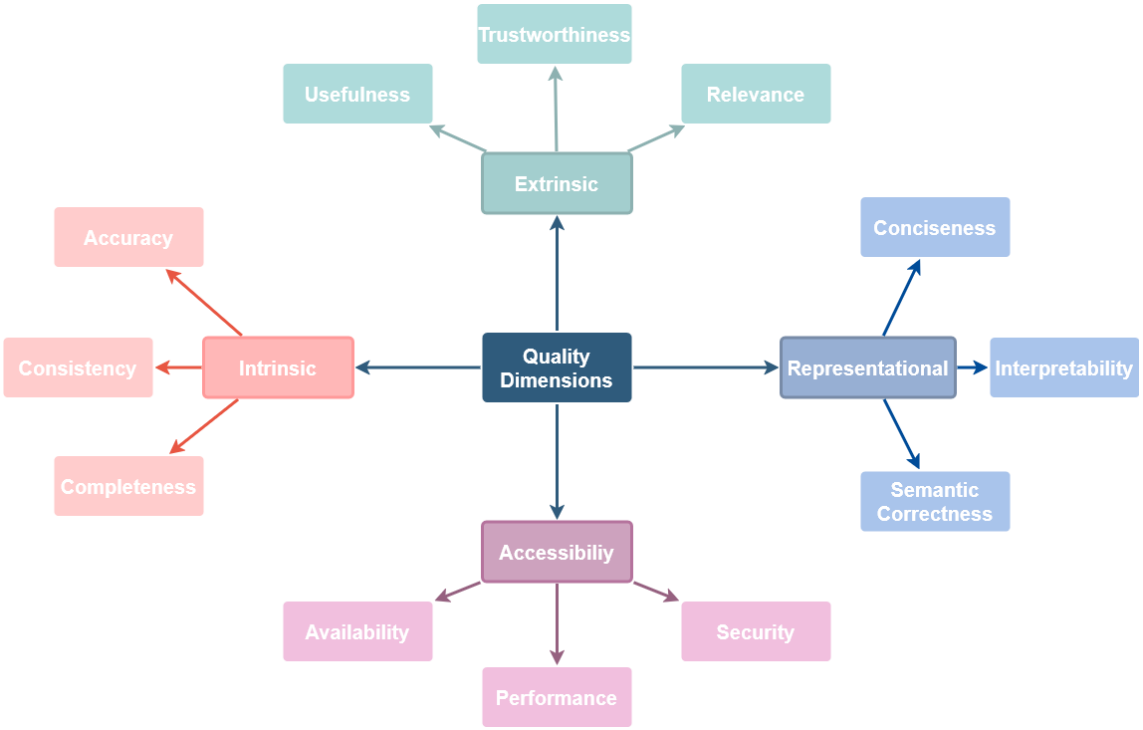

Figure 2.1: Taxonomic Representation of the Quality Dimensions as defined in [54]

in the field and were reused and/or defined in the context of knowledge graphs. The 18 identified quality dimensions were categorized into four main quality aspects. Figure 2.1 illustrates the taxonomy of the quality dimensions and to which of the four main quality classes they belong to. Intrinsic quality measured the quality of the data 'independent of context' with dimensions such as syntactic and semantic accuracy/validity. On the other hand, contextual quality dimensions like relevancy and completeness took the context into account [54]. Additionally, representational quality examined the format and meaning of the data to check for ambiguity and understanding, as well as interoperability. Lastly, the accessibility of data was considered with availability, licensing, and interlinking dimensions.

In [55], the quality dimensions of [54] were extended, and a framework of 34 quality criteria was applied to five publicly available KG datasets: DBpedia, Freebase, OpenCyc, Wikidata, and YAGO.
The experiment showed that all KGs provided syntactic valid RDF documents. Wikidata proved to be a very trustworthy KG that also has the highest schema completeness compared to the others [55]. Due to being editable by the community, it makes consistent checks for schema restrictions. DBpedia had the highest column completeness, a low coverage of labels, and an acceptable schema completeness. YAGO had high trustworthiness on the statement level and showed acceptable schema completeness, while also achieving the best results with OpenCyc for interlinking [55]. Opposite to Freebase and OpenCyc, the other three KGs were easily human-readable due to multiple RDF serialization formats. Overall, the application of the 34 quality criteria showed the range of quality of the knowledge graphs and demonstrated the applicability of these criteria for further research [55].

[7] referred to the research of [54] from 2015 for measuring the quality of linked data. The four categories of dimensions were of interest here: Accessibility, Intrinsic, Consistency, and Representational. The authors specified that a KG is of high quality if it is "correct, comprehensive and fresh" [7]. Thus, the intrinsic category was of special importance because it not only validates the KG on a syntactic and semantic level but also emphasizes consistency, conciseness, and completeness [7]. The authors proposed a framework for evaluating the quality of a KG, i.e., if the KG is "fit for purpose". The basis of this framework was the dimensions and their respective metrics. Overall, among the 18 quality requirements to indicate whether a KG is "fit for use" were dimensions such as context, conciseness, scalability, organized structure, and more [7]. These requirements were developed from the various applications that KG can be used in. The framework chooses to evaluate all, or some dimensions based on the feasibility of each metric [7].

A survey by [50] in 2022 also explored other dimensions of quality assessment. They claimed that other quality management methods are not feasible because KGs have heterogeneous data, are noisy, incorrect triples would be more difficult to detect and most methods would be too complex for huge knowledge graphs. According to them, a KG must be "fit for use", i.e., the quality can only be evaluated depending on the context. Two main dimension types were of interest to the authors, intrinsic dimensions, which can be applied to any KG, and extrinsic dimensions, which are more application-dependent. The focus of the research was on the intrinsic dimensions including accuracy, consistency, completeness, timeliness, and redundancy. The survey focused on the different methods of quality assessment. Amongst others, outlier detection, based on statistics and machine learning classification, was used for assessing the quality of dimensions such as completeness [50]. An outlier was defined as an error, such as wrong triples and missing type statements in KG quality assessment. A possible approach to outlier detection was to assign each statement or triple a confidence score and set a threshold to identify outliers ([38], as cited in [50]. Embedding-based methods were also stated as statistical approaches, where the head and tail entities, as well as the relations of the graph, are converted into embeddings. The embeddings can then be used for tasks such as link prediction, triple classification, etc. but cannot be used for quality evaluation as they do not necessarily consider the semantics [50]. Rule-based approaches were also reviewed by the authors. Making use of predicate logic and symbolic reasoning in combination with manual annotation or other statistical measures can enhance detecting errors and incompleteness in KGs. [50] finalized their survey by discussing the importance of combining these methods in future works instead of using each method on their own.

[42] introduced structure-based metrics focusing on the structure of the KG and its underlying ontology. This evaluation was chosen because they claimed that a KG is "good" if the classes and properties defined in the ontology are used. The instantiated class and property ratio were suggested to see which classes/properties are used in the KG, as "empty" classes and properties decrease the conciseness of the KG and, thus the quality [42]. The complexity of the ontology structure was also measured with the class instantiation, which assesses how fine-grained the ontology is by looking at how many subclasses each class has and if they are included in the graph. Many subclasses, which are also utilized by the ontology and graph, i.e., having a

high class instantiation score, indicated that the KG is more specific [42]. The subclasses were also measured with the subclass property acquisition and instantiation metrics. The subclass property acquisition score indicates if the subclass is specific to the subclass and is not used in the superclass making the property more exclusive and thus better for the quality of the ontology. The subclass property instantiation again indicated how many properties only specific to the subclass are actually being used [42]. Finally, [42] introduced the Inverse Multiple Inheritance (IMI) measuring the overall complexity of the graph. It calculates the average multiple inheritance with the average number of superclasses. They applied these structural metrics to already existing KG graphs such as DBpedia, Wikidata, Freebase, YAGO, and Google KG, and a newly introduced KG, Raftel, that consists of the aforementioned KGs. The classes, properties, and relations were modified so that the ontology was consistent and concise [42]. DBpedia and Raftel showed a higher class instantiation score indicating that they are fine-grained KGs. Raftel was also identified as the least complex KG according to the high IMI score. Therefore, [42] not only introduced structural metrics to measure the quality of a KG but also a new baseline KG.

Another quality assessment framework was presented by [18]. Here, the authors differentiated between quantitatively (QN) and qualitatively (QL) measured metrics, i.e., if user evaluation (QN) or calculation (QL) was necessary. Domain experts were needed to select the KG settings, the relevant quality dimensions, and the appropriate quality metrics [18]. The assessment in the framework was conducted by calculating all metrics with the correct settings and weights separately. For each quality dimension, the aggregated score of the metrics specified for that dimension was computed. Finally, the aggregated score of all quality dimension scores was computed and represented the quality of the knowledge graph [18].

2.5 Other Knowledge Graph Tasks

A common subtask of KG construction is KG completion (KGC), where the validity of triples is classified, and missing entities and relations are detected [3]. KGC is used for validating and expanding existing KGs by completing existing KGs through relation and node predictions [19].
[3] introduced KGValidator, a framework for evaluating KG completion models, i.e., identifying an unseen triple as correct or incorrect. The KGC predictions had to be in the form of a list containing the knowledge triples (head, relation, tail). The LLM then had to predict whether the given triple was valid [3]. They also used Wikidata as the reference KG in the RAG pipeline with the predicted triples and given to the validator. If a triple had no corresponding Wikidata information, an agent was used to extract the additional information from the web [3]. Experiments were conducted with GPT-3.5.-Turbo and GPT-4-Preview on various KGC datasets, such as Wiki27k. GPT-3.5 showed bad performance when only utilizing its internal world knowledge. The performance was improved with the contextual information and web searches, which led back to its struggle with long-tail memory. GPT-4 had a strong performance in all configurations, so only with its internal knowledge and also with additional context. The model configurations all worked better with extra information, as there were ambiguous triples that could only be disambiguated with additional context, thus improving the accuracy in these settings [3].

On the other hand, KGs are also used to improve LLMs. Even though LLMs proved to outperform other deep learning models in NLP tasks, they cannot often provide factual knowledge and produce so-called hallucinations [36]. They defined hallucinations as statements generated by the LLM that are incorrect because the LLM cannot remember the knowledge that was memorized during training [36]. [26] distinguished between content and ontology hallucinations in their relation and tail entity prediction experiments. Content hallucinations were described as relations produced by the model that do not exist or are inaccurate factually, while ontology hallucinations were defined as invalid paths, e.g., if wrong types of entities are linked [26]. As a solution, they proposed to reduce these hallucinations with refined prompt engineering, in this case, by providing context from Wikipedia, which improved the results [26]. This was also indicated by [6], who claimed that adding natural language knowledge from external databases can decrease hallucinations.
Related to hallucinations, there is another shortcoming of LLMs, namely their black-box behavior [6]. The reasoning behind making decisions is often not clear in LLMs, which makes it difficult to interpret them and determine why they sometimes produce incorrect answers [36] [6]. Using KGs can add structure to LLMs while also offering simple readability of the data [9].
Another disadvantage of LLMs is their lack of domain-specific and new knowledge since they are mostly trained on cross-domain corpora [51]. This can be difficult to solve as the infusion of new knowledge would require re-training the entire model, but KGs can help as external knowledge sources to keep the LLM current [36],[51]. [51] explained that the infusion of KGs into LLMs can be done before the training of the LLM by combining the corpora and the KG content, during the training with the encoder, or after the training, which is most common for adding domain knowledge.

Some works presented in this section dealt with KG generation with LLMs, such as Llama and GPT-4, yet a thorough quality assessment of an LLM-constructed KG from unstructured data was not done with more advanced models which since then were introduced. A study considering multiple quality dimensions for evaluating a KG generated with LLMs was not found during the research of this thesis, and thus, might not exist yet. Therefore, this work will investigate the generation of knowledge graphs from unstructured texts with LLMs. Then, the quality of the KG is evaluated by a more fine-grained quality assessment framework.

3. Methodology

This section will introduce a new method to generate knowledge graphs using LLMs. First, it will cover the data used and how it is preprocessed, as well as the models that will be utilized to create a knowledge graph. Then, it will focus on the two-step knowledge graph generation process. Step one describes the ontology extraction step, which yields a hierarchical entity structure with property names and relationships with descriptions and possible start and end entities. The second step explains the KG generation step in detail, i.e., how the ontology is used to extract the information from the articles to get a final KG. For comparison, another approach for KG generation is described as well. Finally, a KG evaluation framework will be established to measure how well the KGs are constructed by the LLMs. The implementation of these methods and the evaluation metrics can be found in the respective GitHub repository[1].

3.1 Data

A lot of datasets containing triples of KGs already exist, but there is no dataset that consists of unstructured texts and the representative KG. Nevertheless, there are DBpedia and Wikidata datasets, both constructed from Wikipedia articles. Therefore, this study makes use of scraped Wikipedia articles to generate a KG so that the automatically generated KG can be compared to the existing KGs in Wikidata and DBpedia.
Using the Wikimedia Dump webpage[2], the Wikimedia dump from 20 May 2024 was used for the experiments, which was the most recent Wikimedia dump when the experiments were planned. It is currently not available on the website anymore, because newer dumps have been published since then. The 20240520 dump can be found in a Google Drive folder[3] and not in the GitHub repository due to the large size of the file. Overall, the data contained 290.247 articles. The wiki-dump-reader[4], a python library specific for importing the wiki dumps, was used for reading the downloaded Wikipedia articles. It returned the title, the article content and additional links, which were referenced in the text.

Since the article content sometimes only referenced another article with a *"Redirect"* and contained titles, the articles had to be cleaned first. For this, all articles with the content *"Redirect"* were dropped from the dataset. Furthermore, non-English characters were removed from the text as they were difficult to handle otherwise.

[1] https://github.com/sarahackerschewski/knowledge-graph-generation-LLMs
[2] https://dumps.wikimedia.org/
[3] https://drive.google.com/drive/folders/1iQSfY9YcW1MsOXBcy6we56wAuQJ3B0Q2
[4] https://pypi.org/project/wiki-dump-reader/

The article content was structured with a heading that started and ended with '=='. These headings were removed, and the different text units were merged into one text. This reduced the length of the text and was done to make the text more accessible to the model in later steps because it was more comprised and took less input tokens. In a final step, 1000 articles were sampled randomly from the dataset with a seed of 25 to be used as the data for the task at hand. These articles had an average of about 93 tokens in the cleaned texts and an average of about 309 tokens in the original texts, where all tokens divided by a whitespace were counted as a separate token.

3.2 Models

Different models were used to test the automatic generation of the knowledge graph and the various factors influencing a KG. In the literature, models by OpenAI, especially GPT-3.5 and GPT-4, were used due to their high performance in text understanding and reasoning capabilities[6, 12, 26, 51]. Since then, OpenAI has published further improved models with better performance on various tasks. Thus, the models GPT-4 Turbo, GPT-4o, and GPT-4o mini were chosen for the following experiments to enhance the generation of KGs.
GPT-4 Turbo was introduced as an OpenAI model with an extended context window of 128K tokens and with a knowledge cutoff in December 2023 [31]. The increased context window could offer the opportunity to use an ontology as an additional input for the KG construction task. Furthermore, OpenAI presented GPT-4o as a multimodal large language model, including improved performance on textual data [33]. This also made GPT-4o interesting for the ontology extraction and KG generation task. GPT-4o is quite large, however, OpenAI also published GPT-4o mini to offer high performance for less of a price, though GPT-4o mini is still weaker than GPT-4o [32]. Therefore, GPT-4o mini was also used for the construction of the KG to investigate if the performance was enough to construct a useful knowledge graph. The OpenAI Assistant API contained the prompts for each task. In the Assistant dashboard, it was possible to switch between the models. The runs for each model were then accessed via the OpenAI API. Each model was run with the same parameters, i.e., a top P of 1 and a temperature between 0.2 and 0.25, depending on the task.

Access to an OpenAI key and the funding of the model usage were provided by *textelligence GmbH*[5] as the external advisor of this thesis.

3.3 Ontology Extraction

As most research suggested, the KG generation task was split into two different primary tasks, ontology extraction and knowledge graph extraction [19]. First, the ontology was extracted so that it could be used as additional information for the LLM when creating the KG. This task was further broken down to deal with the model output token limit of 4000 for each of the models used.

Here, an ontology was not created in the same sense as described in previous research above (see 2). The ontology was defined as a JSON with hierarchical entities that

[5]https://textelligence.de/ - Contact Person: M.Eg. Marin Jukic (m.jukic@prstg.de)

have properties assigned to them, which also should be inherited. This simplified ontology was chosen over known ontology structures, such as RDF, to make the results more comparable to the KG of the second approach used, the LangChain LLMGraphTransformer. Furthermore, it was hypothesized that the resulting ontology and graph would be better with a specific ontology just for the articles at hand than a more general one, e.g., an RDF schema. The following shows the example ontology structure, which was provided to the ontology extraction LLM as well:

```
{
  "entities": {
    "Entity1": {
      "properties": {
        "property1": "datatype",
        "property2": "datatype"
      },
      "childrenEntities": {
        "SubEntity1": {
          "properties": {
            "property1": "datatype",
            "property2": "datatype",
            "property3": "datatype",
            "property4": "datatype"
          }
        },
        ...
        }
      }
    },
    "Entity2": {
      "properties": {
        "property7": "datatype",
        "property8": "datatype",
        "property9": "datatype"
      },
      "childrenEntities": {
        ...
      }
    },
    ...,
    "Other": {
      "Entity3": {
        "properties": {
          "property10": "datatype",
          "property11": "datatype",
          "property12": "datatype"
        }
      },
      ...
    }
  },
  "relationships": [
    {
      "type": "RELATIONSHIP_TYPE",
      "description": "short description of the relation",
      "source": "Entity1",
      "target": "Entity2"
    },
    ...
  ]
}
```

The ontology extraction task consists of entity and relation extraction using prompt engineering. For this, an OpenAI Assistant[6] was set up and called by the code to extract the entities and relationships for the given data. In the Assistant dashboard,

[6]https://platform.openai.com/docs/assistants/overview

the model was prompted to generate a plain ontology as simple as possible for a list of Wikipedia articles. The relation had to consist of the type, the head/source entity, the tail/target entity, and a short description of the relation to give more details about the type and help identify duplicates later. The complete prompt can be found in the Appendix A.1.1. The model was also specified on the dashboard. Here, for each model, a temperature of 0.25 and a top P of 1 were chosen.
Requests to the assistant were made in batches of 10 articles per list, and each response of the model was saved in a joint JSON file. The hundred "ontologies" had to be merged into one. Furthermore, cleaning the ontologies was necessary as well. The entities were inconsistent with their writing conventions, and the final ontology was prone to duplicates due to the multiple batches and merging tasks. For unifying the writing conventions of entities containing whitespaces, underscores, and/or hyphens, the convention *"OneTwoThree"* was chosen as it reflects how most entities were represented in the research and would reduce problems that could arise when using, e.g., whitespaces. This resulted in *"Stock Exchange"* becoming *"StockExchange"*, *"Military_Installation"* becoming *"MilitaryInstallation"* and *"Re-admission*" becoming "*ReAdmission"*, for example. The final sizes of the entity lists for each LLM can be seen in Table 4.1.

The cleaning of the relationships was more complex. The same problems regarding the conventions of entities also occurred in the source and target entities of the relations and had to be applied. Additionally, upon quick manual assessment, a lot of duplicates were discovered. A relation often included source/target tuples that conveyed the same relation with different expressions, e.g., the relation *"WAS_BORN_IN"* was used by the source/target tuples *("Person", "DateOfBirth")*, and *("Person", "BirthDate")*. To fix this, the Cosine similarity between each tuple of a relation was calculated using OpenAI Embeddings, and only one of the most similar tuples was kept in the ontology. A threshold of 0.55 was chosen for the similarity after trying out different thresholds, such as 0.8, 0.75, 0.7, 0.65, 0.6, 0.55, and 0.5. The 0.55 threshold was able to reduce the duplicates the most, while not deleting correct relations, like the 0.5 threshold. This way, the number of relations in the GPT-4o ontology was reduced from 1044 to 505.

In the next step, a hierarchy was built to find the relations between the entities extracted before. Here, an entity could act as a parent entity and have child entities, be a child entity, or stand alone in the ontology, similarly to super- and subclasses, e.g., defined in [42]. The hierarchy extraction was done in batches of 5, where each batch included 100 entities. Two prompts were created with the OpenAI Assistants. The first prompt (Appendix A.1.2) instructed the model to create an "initial" hierarchy from the first batch. The initial hierarchy was extended using the second prompt (Appendix A.1.3), which included the batch and the hierarchy of the previous output each time, resulting in one final hierarchy. The run thread of the assistant was not deleted until the last batch was run to make sure that the previous contents of input and output were taken into consideration. The response of the last run was taken as the final hierarchy.
The hierarchy was then split into 13 or 8 batches, depending on the model with which the hierarchy was created, each batch containing a different number of entities due to their varying structure (see figure 3.1. Each batch was given to another

assistant to extract the properties/attributes of each entity. The prompt (Appendix A.1.4) highlighted the concept of inheritance to ensure that children entities have the same properties as their parent but can also extend these properties. Here, the thread was also kept running until the last batch was processed. This ensured consistency and reduced duplicates because the model could process each batch on its own while still remembering what was processed in earlier batches. The batches were then merged into one dictionary of entities.

In the concluding step, the entities were combined with the relations extracted in the first step, resulting in the final ontology. The whole process was conducted with each model (see 3.2) for comparison. The property extraction step with GPT-4 Turbo yielded mainly bad results, hence, this step was repeated with GPT-4o because the incomplete GPT-4 Turbo ontology produced too few properties to continue ontology construction. Figure 3.1 depicts the ontology extraction process and all its substeps for better comprehension. As can be seen in the figure, the process varied slightly for the GPT-4 Turbo, because GPT-4o had to be used for the properties but else the process remained the same. Overall, three ontologies were created using the three models, or in GPT-4 Turbo ontology's case using two of the models, i.e., entity, relation, and hierarchy extraction using GPT4 Turbo and property extraction using GPT4o.

3.4 Knowledge Graph Construction

After extracting an ontology for the data, the next step was generating the knowledge graph. The LLM was provided with the articles and the ontology from the previous step to produce <*head entity, relation, tail entity*> triples. Using an OpenAI Assistant again, the prompt (see Appendix A.1.5) specified how the KG should be constructed, i.e., trying to be consistent and not redundant, as well as the JSON output format for the nodes and relations. To keep an overview of the generated nodes and relations, they are returned separately, and the relations refer to the unique IDs of the entity nodes instead of including the whole node in the triple. A relation had to contain the type or name of the relation, a startNode and an endNode referencing the ID of an entity node to mark the subject and object entity, respectively, and a short description of the relation. In the case that the node was not created before, the model was also allowed to add a string of the entity instead of an ID.
Each article was processed on its own to ensure that the information from the different articles did not overlap. The JSON containing the results of the 1000 batches was merged by adapting the IDs so that the numbering was continued instead of starting from "*1*" again each batch. Likewise, the IDs in the startNodes and endNodes were adjusted with the new unique IDs, resulting in the final KG. Additionally, the nodes that had the same name were assigned to the same ID so that duplicates were kept minimal.
All three models were run with a top P of 1 and a temperature of 0.2, which was decided after testing the prompt in the dashboard interface of OpenAI, also called the "playground" option of the Assistants API.

As an alternative approach, a knowledge graph was generated in only one step from

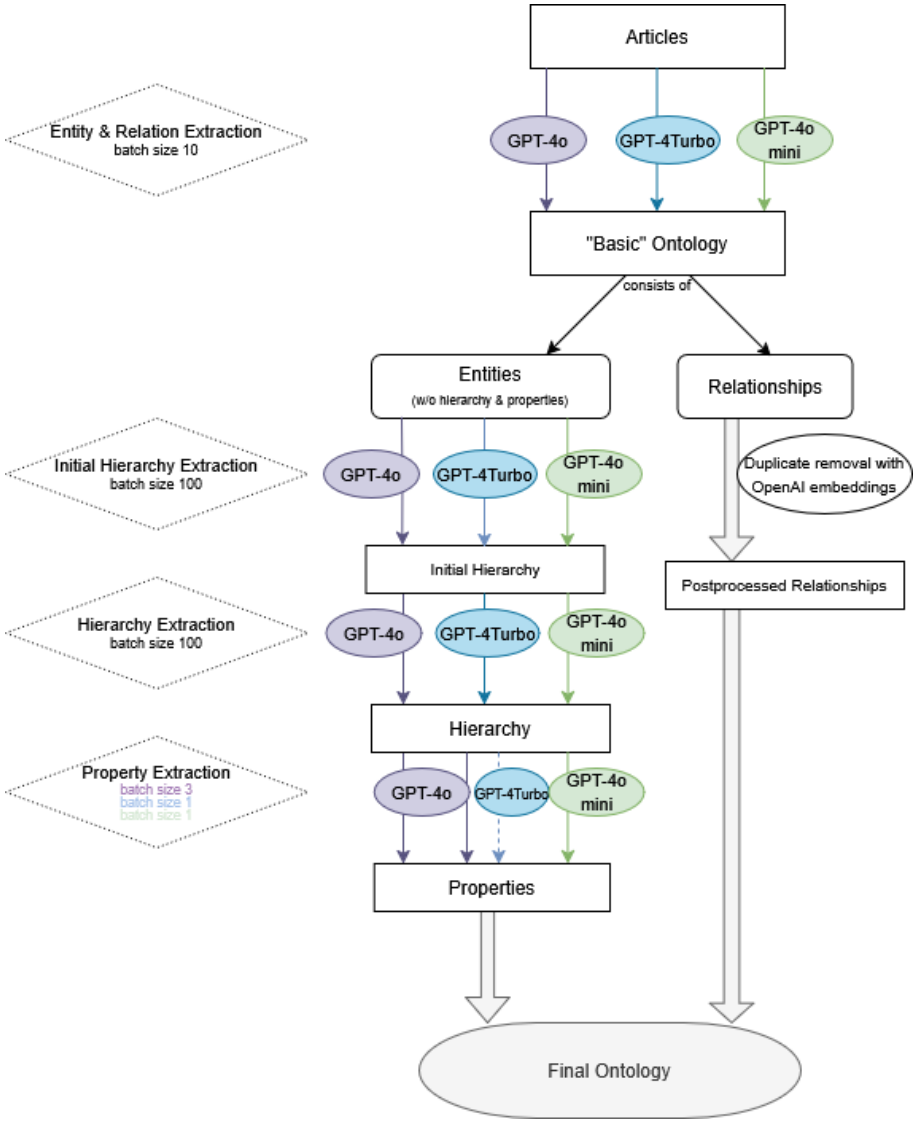

Figure 3.1: Detailed process from texts to an ontology using one model consistently for each step. Exception for property extraction with GPT-4 Turbo, which did not work and GPT-4o was used instead.

the same cleaned Wikipedia articles, so without including the ontology as input for the KG generation. Thus, one of the research questions, investigating whether the graph benefits from extracting the ontology beforehand, could be explored. This one-step approach was done using LangChain because the framework already offers a document-to-kg approach with the class LLMGraphTransformer[7]. The approach was implemented using a guide published by LangChain (see [24]).
Each article was converted to a *Document* type also defined by LangChain and then given to the LLMGraphTransformer in batches of one to generate graph documents, which are structured representations of the documents [24]. By leveraging an LLM, in this case GPT-4o with a temperature of 0.2, entities and their relationships were extracted and returned. GPT-4o was chosen as it seemed the most popular model among the three OpenAI models selected for the previous experiments. Similarly,

[7]https://api.python.langchain.com/en/latest/graph_transformers/langchain_experimental.graph_transformers.llm.LLMGraphTransformer.html

the temperature of 0.2 was chosen to match the temperature for the automatic KG generation with the OpenAI Assistants to make the results more comparable.
The resulting graph documents of each batch were automatically added to a Neo4j graph. Additionally, they were also appended to a JSON in a similar structure to the batch results of the approach using Assistants. Then, the JSON batches were merged into a final knowledge graph JSON, also removing duplicate nodes and relations.
Since the resulting JSON containing the knowledge graph was about 234 MB, it could not be uploaded to the GitHub repository but instead was added to the Google Drive also used for the complete dataset but in a separate folder[8]. The ontology could be accessed simply through the graph schema of the Neo4JGraph and could be added to the GitHub repository. Thus, there was no need for a separate ontology generation step and cleaning of the ontology file. For simplicity and clarity, the graph created with the LLMGraphTransformer using GPT-4o was called LangChain graph in the rest of the thesis.

3.4.1 Neo4j

The knowledge graphs generated by the two-step system were also uploaded to Neo4j to create a graph that can be queried with Cypher, the query language of Neo4j, like the graph created with the LLMGraphTransformer. This was done "manually" by creating a Neo4j Aura instance and passing it a Cypher query that uploads the nodes and relations from the JSON files containing the KG[9]. The JSON file had to be saved in the Google Drive folder[8] so that Neo4j had access to the file.
All three graphs were uploaded using the same two Cypher commands, one for the node upload and one for the relationships upload. Only the links to the file were adapted. The Cypher commands can be found in Appendix A.2. The GPT-4o KG file had to be changed slightly because the node for the star *"MY Apodis"* had a nested property dictionary that could not be handled by Cypher. Here, the property *"outerLayer"* was removed but its inner dictionary was kept, so that *"hydrogenShell": "thin"*, *"heliumLayer": "1.5 to 20%"*, *"core": "extends to 60% of the stellar radius"* were still preserved as properties just without its nested upper layer.

3.4.2 Triple Extraction

The JSON representation could offer more insight into the entities and relationships with properties and descriptions. Nevertheless, a knowledge graph in the typical sense contains triples of type <*subject, relation, object*> and not a dictionary entry [36, 42, 54]. Therefore, a function was implemented that converted the JSON relationships to triples in the correct format. In order to get the correct subject and object from the properties of the respective IDs referenced by the JSON triple, a simple check was made to see which property names were relevant subjects and objects. These were then accessed and represented the entity, e.g., the value of the *"name"* or *"title"* property keys. More difficult were date values, as they could be represented in various formats. Hence, the datetime library was utilized to extract dates from the string value. For example, the date *"10.01.1989"* was represented

[8]https://drive.google.com/drive/folders/17G5QtdsHmvZnxRfRE5G2S2BJt4DBwISh?usp=drive_link

[9]https://neo4j.com/docs/getting-started/data-import/

as *"year": 1989, "month": 1, "day": 10*, while others were represented with missing year, month or day information or even as one string in different ways. These were all converted to the unified date representations "*dd/mm/yyyy*", "*mm/yyyy*", and "*yyyy*" depending on the values filled in the entity. Dates that represented a range were mostly kept as they were to not overcomplicate the matching process because they only occurred seldom in the GPT-4o mini KG.

3.5 Evaluation

The generated knowledge graph was evaluated on different levels to ensure good quality. In the context of KGs and linked data, quality was mainly defined as "fit for use" [7, 54]. Using the Wikipedia articles created a general, not domain-specific, KG. Thus, the "use" of this KG is to provide the content of the different articles and connect the knowledge between the articles, if possible, so that it could be used for question-answering tasks. The evaluation was done on two different levels: structural quality and knowledge or content quality. Various quality metrics and other approaches were used to assess the quality of these levels.

3.5.1 Structural Quality Evaluation

First, the structural level was inspected to see whether the entities and relations were used appropriately. For this, four quality metrics were taken from [42]. Quality data metrics for KGs generally measure the ratio of different occurrences ([54]), which is also the case for the metrics from [42]. "Entities" and "classes" are used interchangeably in the following.

When measuring the quality of a knowledge graph, most research referred to quality dimensions so that various aspects of the KG could be considered [55]. Since not all dimensions fit the graphs generated in this paper, only a subset was chosen for evaluation.

One of the quality dimensions defined by [54] was syntactic validity and semantic accuracy. While semantic accuracy was investigated in the knowledge/content quality evaluation, syntactic validity was looked at in the context of structure. Here, the syntactic validity was checked by inspecting whether all items of the KG triple were filled. This was done with a simple search of empty quotation marks ('') in the results of the triple generation 3.4.2. If there were empty brackets in a triple, it implied that either the head or tail entity or relation type was not filled out by the model, thus being a syntactically invalid triple.

Redundant classes and properties also reflect the bad structural quality of a knowledge graph [7]. Instead, a graph should be concise and complete. To ensure the conciseness of the knowledge graph, the definition and usage of entities and their properties had to be investigated. For the entities, Instantiated Class Ratio (ICR) was calculated to see whether the classes defined in the ontology are being used. This metric from [42] was used for ontology and KG extraction (see 3.1 & 3.2, respectively), as the ontology was also constructed by an LLM, Some of the entities defined in the ontology might also not be utilized in the relationships of the ontology, and because the ontology was used as a base for the KG generation, it was an

important aspect to look at. Similarly, the Instantiated Property Ratio (IPR; 3.3)as computed to determine how the properties from the ontology are used in the KG. Since the properties are only used in relations in the KG, there is no ontology IPR. Subclass Property Acquisition (SPA; 3.4) identified the average number of properties of a children entity added to the properties the entities inherited from their parent entity. If no new properties were added to the child entities, the ontology would not be as concise as possible because the child entities would be redundant. Here, a second version of the SPA is introduced to exclude entities that have no hierarchy. Finally, the Inverse Multiple Inheritance (IMI; 3.5) was utilized as a measure of simplicity. The more parent entities an entity has, the more complex is the inheritance of properties and the less simple the knowledge graph would be.

In the following, the used metrics are represented with their formulas, which were taken from [42]. Here, N represents the number or count of something. IC and IP refer to the instantiated classes *(C)* and properties *(P)*, respectively. *nsup* stands for the number of superclasses or parents a class has.

- Instantiated Class Ratio (ICR)

$$ICR(Ontology) = \frac{N(IC_o)}{N(C_o)} \tag{3.1}$$

$$ICR(KG) = \frac{N(IC_{kg})}{N(C_{kg})} \tag{3.2}$$

- Instantiated Property Ratio

$$IPR(KG) = \frac{N(IP)}{N(P)} \tag{3.3}$$

- Subclass Property Acquisition

$$SPA(Ontology) = \frac{\sum N_i(P_{child} - P_{parent})}{N(C)} \tag{3.4}$$

- Inverse Multiple Inheritance

$$IMI(Ontology) = \frac{1}{\frac{\sum_{i=1}^{N_c} nsup(C_i)}{N_c}} \tag{3.5}$$

Furthermore, [42] also introduced two class-specific metrics: Class Instantiation and Subclass Property Instantiation. These were not considered since ontologies were not domain-specific, and thus, the KG produced too many classes.

In addition to these metrics, further statistical evaluations were conducted. To extend the investigation of the class usage except for the ICR, it was examined how ontology-derived entities were used as labels in the KG and how many new entities were added during the KG generation step. Similarly, ontology-derived relationship types and newly added relationships were investigated to measure the relationship usage, which was not dealt with before. These statistics were included in the evaluation framework to help understand the structural metrics more.

3.5.2 Knowledge Quality Evaluation

The quality of the structure of the KG and its underlying ontology is valuable to check if the construction worked well. Yet the correctness of the content cannot be assessed by the structural quality metrics alone. Instead, an evaluation framework has to include accuracy measures to capture the knowledge quality. However, the Wikimedia dump did not contain gold triples that could be used to compare the generated knowledge graph with a real KG. In order to still be able to calculate the accuracy, KGs like Wikidata ([45]) and DBpedia ([1]) are based on Wikipedia articles. Both KGs were queried with a SPARQL query to yield the relevant triples from the articles used to generate the KG from this thesis. The resulting triples of links were accessed to reveal the actual entities and relationships instead of IDs.

The accuracy of the generated KG could then be calculated with the "gold triple". In some papers, different variations of accuracies were computed to investigate in which regards the KG was well constructed and in which not [26, 28]. The exact and partial accuracies were introduced but also varied in their definitions. Here, variants of partial and exact accuracy were redefined to fit this specific task and research question.
The exact accuracy (EA) overall remained the same, i.e., calculating the ratio of correct triples in all triples generated. However, after some testing, the EA (and even the partial accuracy) would have falsely returned 0 as the conventions of Wikidata and DBpedia of relations varied from the conventions of the generated KG. To fix this, a mapping of frequent relations was constructed manually and was applied to the gold triples to yield the same relationship name conventions, e.g., *"date of birth"* from Wikidata became *"BORN_ON"*, matching the generated KG. The same was done before calculating the partial accuracy. Also, DBpedia often returned a triple, where the predicate only returned *"Wikidatalink"*, with no possibility to trace back to which Wikidata relation it links. These relations could not be mapped and thus could only be compared via partial accuracy.

Referring to [28], the partial accuracy (PA) was implemented as the ratio of triples matching the gold triples either in subject (subj.) and object (obj.), or subject and relation (rel.).

- Exact Accuracy (EA)

$$EA = \frac{N(\textit{correctly predicted triples})}{N(\textit{gold triples})} \tag{3.6}$$

- Partial Accuracy (PA)

$$PA = \frac{N(\textit{(correct subj. \& obj.)}|\textit{(correct subj. \& rel.)})}{N(\textit{gold triples})} \tag{3.7}$$

Moreover, some examples classified as correct and some that were classified as incorrect were chosen for manual evaluation. By looking at an Excel file produced during the content quality evaluation, about 100-150 random "predicted triples" per KG were checked if a match with the "gold" triples from Wikidata and DBpedia was

Question	Cypher query by GPT-4o	Cypher query modified
1. Which people have an English nationality?	MATCH (p:Person)-[:HAS_NATIONALITY]->(n:Nationality {name: "English"}) RETURN p.name	MATCH (e)-[:HAS_NATIONALITY]->(n:Nationality {name: "English"}) RETURN e.name;
2. When and where was Mihail Andricu born?	MATCH (p:Person {name: "Mihail Andricu"})-[:BORN_IN]->(place:Location) OPTIONAL MATCH (p)-[:BORN_ON]->(date:Date) RETURN p.name, date.value AS birthDate, place.name AS birthPlace	MATCH (p {name: "Mihail Andricu"})-[:BORN_IN]->(place), (p)-[:BORN_ON]->(date) RETURN p.name, place.name AS birthPlace, date.day + "-" + date.month + "-" + date.year AS birthDate;
3.What is the Yser Testament?\label{question3}	MATCH (d:Document {title: "Yser Testament"}) RETURN d.description	MATCH (d {title: "The Yser Testament"})-[r]->(object) RETURN d, type(r) AS relationship, labels(object) AS objectLabels, properties(object) AS objectProperties;

Table 3.2: Questions for the Question-Answering evaluation of the graphs. Cypher query by GPT-4o shows the queries generated using GPT-4o and Cypher query modified how these were adjusted to fit the graph scheme, here the GPT-4o graph, as an example.

successful and, if so, whether the match is correct. Furthermore, it was investigated whether the unmatched "predicted" triples should have been matches and for what reason. This was done to help identify the causes for the misclassifications and lead to improvement for future KG generation research.

3.5.2.1 Question-Answering Quality

To take the content quality evaluation one step further, the graph was queried with questions to see whether the KG was "fit for use". Here, the use case in "fit for use" was a simple question-answering task using GPT-4o via the ChatGPT interface to generate Cypher queries for the questions and modify these queries so that they match the graph schema. Even though an existing approach to convert natural language into Cypher, query the graph and convert the answer back to natural language by LangChain exists, it could not be applied since the graphs were too big to add them as context to the GraphCypherQAChain [24]. Therefore, this other, more simplified approach with manual adaptation was chosen. The answers were also not converted back into natural language but kept in their table format, unlike the output of the GraphCypherQAChain.

Each graph was tested with the same three questions that should be answerable given the Wikipedia articles they were all based on.

1. Which people have an English nationality?
2. When and where was Mihail Andricu born?
3. What is the Yser Testament?

First, a broad question regarding multiple articles was asked (see Question 1). Then, it was investigated how well the knowledge in the graph for one random article was. For this, one question (Question 2) was "easier" in the sense that it contained the relationship type ("born_in/on"), and the other one (Question 3) was more difficult because it broadly asked about the entity. The questions were chosen by drawing three random articles from the dataset of 1000 and thinking of a question for each of them. These questions should help inspect how well the automatically generated graphs were "fit for use" on different difficulty levels.

The prompt given to the ChatGPT interface to generate the queries was as follows:

Please convert these questions into Cypher queries:
- Which people have an English nationality?
- When and where was Mihail Andricu born?
- What is the Yser Testament?

Table 3.2 shows the questions, the Cypher query generated by GPT-4o and how the queries were modified to work with the existing graph schema. Depending on each graph, the queries had to be slightly modified. The GPT-4o mini graph, for example, did not create a triple containing the *"BORN_IN"* information for question 2. Problems like these were handled for each graph generated with the approach above and for the LangChain graph. These modifications and the full response of the Cypher queries can be found in appendix A.3.1 and A.3.3, respectively. An assessment of the results of the queries can be found in the next chapter (4.2.2.1).

3.5.3 Quality Dimensions

The here-presented evaluation framework was constructed to assess the quality of the KGs on various levels. Considering the four main quality aspects and their respective quality dimensions defined by [54], almost all aspects were covered in this framework. As seen earlier, Figure 2.1 illustrates the taxonomy of the quality dimensions. The intrinsic quality was addressed by validating syntactic validity, and consistency with the metrics introduced by [42]. Manual evaluation of the generated triples ensured broad accuracy for these triples checking for incorrect and hallucinated triples. On a representational basis, conciseness was also included in the quality metrics by [42]. Additionally, statistics about the leverage of classes and relationships were collected. Semantic correctness on a contextual level was measured by comparing the predicted triples to the triples in the Wikidata and DBpedia KGs. Finally, the extrinsic quality was established with the question-answering task, investigating how "fit for use" the LLM-generated triples were. Accessibility of the graphs was not explored, as the KGs were not published or hosted.
Table 3.3 describes the general quality assessment process. It summarizes the metrics or measures used to validate the specific quality dimensions. The final column clarifies which of the main quality aspects were addressed with the respective quality dimension from the quality aspect.
Overall, the evaluation framework should assess the quality of a graph in a more fine-grained manner than previous research about KG construction and KG quality assessment.

	Step	Metrics/Measures	Quality Dimension
1.	Structural Quality	Ontology: ICR, SPA/SPA V2, IMI KG: ICR, IPR	Intrinsic: Consistency Representational: Conciseness
2.	Syntactic Validity	Empty subject, predicate or object	Intrinsic: Accuracy
3.	Statistics	Ontology: hierarchy-levels, no. entities/relations KG: used entities/relations, no. nodes/relations, most used entities/relations	Intrinsic: Consistency Representational: Conciseness
4.	Semantic Accuracy	Exact accuracy, partial accuracy *(additional manual assessment)*	Representational: Semantic Correctness *(Extrinsic: Trustworthiness)*
5.	Question-Answering	Cypher query for three questions	Intrinsic: Consistency Extrinsic: Usefulness

Table 3.3: Structure of the evaluation framework, describing which metrics were used and which quality dimension from [54] were addressed.

4. Results

In view of the research questions posed in the introduction, the ontologies and KGs generation were evaluated using the evaluation framework to investigate how "good" or useful the KGs are. In this context, a knowledge graph was defined as useful if it performed well across the various quality dimensions, such as accuracy, syntactic validity, completeness and consistency. These dimensions did not only ensure that the KG was structurally sensible and semantically correct and meaningful but also practically applicable for tasks such as question-answering tasks. The following chapter presents the results of the metrics used to assess the performance of the generated graphs across these dimensions.

4.1 Ontology Extraction

The ontologies extracted from the unstructured text worked differently well for each model. Table 4.1 shows how many entities and relations were extracted from the Wikipedia articles, as well as the quality metrics specified for the ontology extraction in the evaluation framework.
The number of entities and the number of relationships in the table represented the sizes of the ontologies. It could be seen that the number of entities produced was similar in the GPT-4o and GPT-4 Turbo ontologies, with 496 and 499 entities, respectively. GPT-4o mini created the least entities of all models and approaches because it resulted in only 203 classes. While the number of entities in the LangChain schema was also quite low, with 326 entities, it contained 2305 relationships, which were five times more relations than the GPT-4o ontology with 423 relationships, making it the most relation-dense ontology. GPT-4 Turbo and GPT-4mini generated the least number of relationships, with 346 and 326 relations, respectively. Compared to GPT-4o, which had the highest relationship count from the ontologies with the new approach, GPT-4 Turbo extracted 18.2% fewer relationships, and GPT-4mini extracted 22.9% fewer relationships.

Concerning the four structural quality metrics, GPT-4o mini outperformed the other models in three of the metrics. In the ontology created by GPT-4o mini, both subclass property acquisition versions were the highest, with an average 2.1 or 2.2 (Version 2) properties, which were added to the properties inherited from the parent entity. In comparison, GPT-4o only acquired about 1 or 1.5 properties as calculated by the original SPA and the second SPA version, respectively. GPT-4 Turbo was somewhere between the other two models, with 1.9 additional properties on average. The LangChain schema did not offer a hierarchical representation of the classes, which is why the SPA versions and the inverse multiple inheritance could not be computed. The IMI also was the highest for the GPT-4o mini ontology,

	GPT-4o	GPT-4 Turbo	GPT-4o mini	LangChain
No. Entities	496	499	203	326
No. Relationships	423	346	326	2305
ICR	**0.6996**	0.6052	0.6897	0.5215
SPA	1.0081	1.8717	**2.1035**	
SPA V2	1.4971	1.9022	**2.1897**	
IMI	0.8158	0.6439	**0.8388**	

Table 4.1: Results of the structural quality evaluation for the ontologies.

indicating that this was the least complex ontology as not many entities had multiple parent classes. However, the difference to the GPT-4o ontology IMI was not as large as the difference between the GPT-4o mini's SPA and SPA V2 compared to the other ontology as GPT-4o mini had an IMI of 0.839 and GPT-4o's IMI was 0.816. Only the ontology of GPT-4 Turbo performed significantly worse with an IMI of 0.644, thus being the most complex ontology of all the generated ontologies.

[42] not only introduced the metrics used in this thesis but also applied these metrics to existing knowledge graphs and Raftel, their KG based on Korean data with the structure of Wikidata. Though the data in the existing KGs was also filtered to the Korean labels, [42]'s results were used for comparison to the here-presented graphs and ontologies for a more diverse evaluation. The SPA of Raftel, Wikidata, and DBpedia suggested that more properties were acquired additionally to inherited ones on average than the automatically generated KGs. Raftel was still in line with the graphs introduced, as it had an IPR of 6.54. However, Wikidata and DBpedia contained more than 40 and 63 properties on average besides the inherited properties, respectively. Nonetheless, the complexity of the GPT-4o and GPT-4o mini ontologies was quite similar to the graphs analyzed in [42]. Their IMIs ranged from 0.962 to 0.975, thus, the least complex graph, Raftel with 0.975, was better by approximately 0.136 points. GPT-4 Turbo performed about 33% worse compared to Wikidata, where the IMI was the lowest of all three looked at in [42] with 0.92.

In addition to the structural evaluation, a short manual inspection showed that duplicate removal for relationship types worked quite well, and most redundancies could be removed. Entities also contained a few duplicates, such as *"Occupation"* and *"Profession"* (found in the GPT-4o and GPT-4 Turbo ontologies). However, applying cosine similarity to the OpenAI embeddings did not work for the entities as it did for the relationships, so they were kept in the ontology.
Overall, looking at the table and the metrics, the ontology extracted with GPT-4o mini was the best ontology structure-wise, though GPT-4o also did not perform badly. It was clear, however, that GPT-4 Turbo yielded the worst ontology results. The complexity of the hierarchies and properties was investigated with the IMI and subclass property acquisition metrics. Furthermore, a broad inspection of the structure of hierarchy and properties was conducted. It showed that the models were capable of finding hierarchical relationships between the entities. The number of main parent entities of all graphs varied between 8 and 13 entities. GPT-4o had the most main classes with 13 entities, while GPT-4o mini and GPT-4 Turbo both had

	GPT-4o	GPT-4 Turbo	GPT-4o mini
Other	149	136	103
Occupation	61	50	0
Organization	24	30	0
Location	31	26	15
Person	7	17	14
Event	24	17	7
Document	31	18	0
Organism	0	0	7

Table 4.2: Highest number of childrenEntities a parent class can have. Among these classes, the top five classes have the highest count for each graph.

8 parent classes.
Looking at the hierarchical structure of all three ontologies extracted automatically with the presented approach revealed that GPT-4o and GPT-4 Turbo showed similar structures, which makes sense considering that GPT-4 Turbo had to be combined with GPT-4o to create the entity properties. Both falsely extracted the parent-child pair *"Person"* - *"Occupation"*, which resulted in *"Occupation"* being the entity with the most child entities as it contained all professions in the data. Table 4.2 highlighted the five classes with the highest children count in all ontologies with a hierarchical structure, i.e., all ontologies except for LangChain. *"Other"* and *"Occupation"* prevailed in GPT-4o's and GPT-4 Turbo's ontologies with a difference of 13 and 11 entities more in the GPT-4o ontology. The *"Other"* class was also the most entity-dense class in the GPT-4o mini ontology. 103 entities could be accounted to that class, although these were falsely classified as actual *"childrenEntities"* of *"Other"*, which was not allowed according to the prompt for hierarchy extraction (see A.1.3). Otherwise, *"Location"* and *"Person"* are the most filled classes, with 15 and 14 sub-entities, respectively. GPT-4o mini did not only add *"childrenEntities"* to the *"Other"* class, but also these child entities were hierarchical sometimes, such as *"SportsTeam"* with its child entity *"Team"*. The same error also occurred in the *"Other"* class of the GPT-4 Turbo ontology, where, for example, *"Chemical"* had a sub-entity *"ChemicalCompound"*. Only GPT-4o aligned with the specifications. Interestingly, GPT-4o mini produced 7 child entities for *"Organism"* while not filling *"Document"*, *"Occupation"*, and *"Organization"*, which were prominent parent classes in the GPT-4o and GPT-4 Turbo ontologies. Thus, it could be seen that those two ontologies overlapped more than with GPT-4o mini.

Overall, these results provide insights into key quality dimensions such as conciseness, completeness, and syntactic validity, which will be further analyzed in the Discussion (chapter 5). Nevertheless, it can be generally said that the ontology extracted with GPT-4o mini was the best structure-wise, though GPT-4o also did not perform badly. It was clear, however, that GPT-4 Turbo yielded the worst ontology results regarding all structural metrics. Concerning the size of the ontologies, GPT-4o and GPT-4 Turbo produced the most entities, while LangChain generated the most relationship-dense ontology. Further manual inspection showed that hierarchical structures were generally well-formed, though some models misclassified relationships, particularly in the *"Other"* and *"Occupation"* classes.

	GPT-4o	GPT-4 Turbo	GPT-4o mini	LangChain
No. Nodes	4586	2938	5329	5781
No. Relationships/Triples	3965	2722	4794	5250
ICR	0.7823	0.5251	**0.936**	***0.9785***
IPR	0.903	0.8501	**0.955**	0.7990

Table 4.3: Results of the structural quality evaluation for the knowledge graphs.

4.2 KG Generation

Using the ontologies extracted from the articles and the Wikipedia articles themselves, the knowledge graphs were generated. The KGs of the individual models performed quite differently in the structural and knowledge quality evaluations.

4.2.1 Structural Evaluation

The variation of the structures of the KGs could already be seen by looking at the number of nodes and relationships produced by each graph. Table 4.3 sums up all the relevant results for the knowledge graphs. Of all models, GPT-4o with the LangChain prompt from the LLMGraphTransformer produced by far the most nodes and relationships, with 5781 nodes and 5250 relations. GPT-4 Turbo, with the suggested approach, created the smallest graph by producing only roughly half of these nodes and relations. The graph generated with the introduced method, which had the highest number of nodes and relationships, was GPT-4o mini with 5329 nodes and 4794. The KG extracted with GPT-4o and the presented approach yielded 13.9% fewer nodes and about 17.3% fewer relations than the GPT-4o mini graph.

The structural quality metrics showed that the graph-based instantiated class ratio (ICR) was the highest for the LangChain graph, followed by the GPT-4o mini graph, where 97.85% and 93.6% of the entity types in the ontology were used in the nodes generated, respectively. Since both graphs also had the lowest of distinct entity types in their ontologies but the most nodes in their graphs, this suggested that almost every ontology step extracted class could be used in the nodes. In contrast, GPT-4o had a lower ICR of 0.78, and GPT-4 Turbo performed the worst, with an ICR nearly 50% lower than the top-performing models.
Similarly, most properties defined in the ontology of GPT-4o mini were used in the respective knowledge graph. The instantiated property ratio of the GPT-4o mini graph was 0.955. However, the LangChain graph had by far the lowest IPR with 0.799, indicating that although the defined classes were used quite often, it could not be said the same about the usage of the properties. On the other hand, the GPT-4o and the GPT-4 Turbo graphs' IPRs performed about 11.5% and 6% better, respectively, compared to the LangChain IPR. Thus, even though their ICRs were worse than the one of the other graphs, the properties were utilized better than Langchain's in addition to the better usage as opposed to the usage of the classes of the ontologies.

These results were also contrasted with the results of the structural quality evaluation by [42]. Looking at the ICRs of the automatically generated graphs, Raftel

	GPT-4o	GPT-4 Turbo	GPT-4o mini
Used Classes	375	259	189
New Classes	95	58	309

Table 4.4: Number of classes from the ontologies used in the graph vs. number of classes newly created in the KG generation step.

outperformed all except for the one of the LangChain KG with 0.941. Wikidata and DBpedia, however, defined many more classes in their respective ontologies than could be used in the graph. Their ICRs were lower than the ICRs of all the other graphs, with one exception, which is explained shortly. Especially Wikidata had a very low ICR with 0.004 for the Korean data and 0.334 for all language data. The Korean "version" of DBpedia only yielded an ICR of 0.47, but if all language data was considered, the resulting ICR of 0.54 was bigger than the 0.53 ICR of the GPT-4 Turbo graph. By contrast, the use of the properties in the Wikidata and DBpedia graphs was 100%, the same as for the Raftel graph. Wikidata had an IPR of 1 for both the Korean data and for all languages. For all language data, DBpedia also had an IPR of 1 and an IPR of 0.99. This showed that all the properties defined in both ontologies were used in the actual graphs. Although the IPRs of the automatically generated graphs were partially quite high as well, like the GPT-4o mini IPR of 0.955, they could not outperform the graphs analyzed in [42].

In regard to the usage of the entity labels created in the ontology extraction steps, not only if they were used with the ICR but also how often they were used was of interest. As presented in table 4.4, the models did not fully utilize the ontology provided in the prompt to generate the labels or needed to generate more new labels because they were not covered in the ontology. Especially GPT-4o mini had a high concentration of newly created entities. From the 203 created classes, 93% were used in the graph labels. However, overall 498 distinct labels could be found in the nodes, thus suggesting that 309 new classes were added by GPT-4o mini.
Interestingly, even though GPT-4o leveraged 75.6% and GPT-4 Turbo leveraged only 52% of the created entities, fewer new entity labels had to be added during the KG generation step. GPT-4o added only 95 new entities, meaning that less than half the amount of entities were additionally added than by GPT-4o mini. The number of new labels added to the graph by GPT-4 Turbo was even lower, with 58 new classes found. This lined up with the number of entities created in the ontology extraction step. GPT-4 Turbo and GPT-4o extracted twice as many entities compared to GPT-4o mini, thus providing more coverage and options for the KG generation model to use.

Furthermore, the KGs distribution of the label count was estimated, which can be found in figure 4.1. The figure contained the top five most-used class labels for each graph and how often they were used in the other graphs. So some classes were not in the top 5 of GPT-4o, for example, but its count was still presented because it was in the five most used labels of another graph. Looking at the most frequently used labels in the GPT-4o graph nodes, the top five labels and their occurrences were as follows: *"Date"* (633), *"Person"* (459), *"City"* (329), *"Organization"* (178)

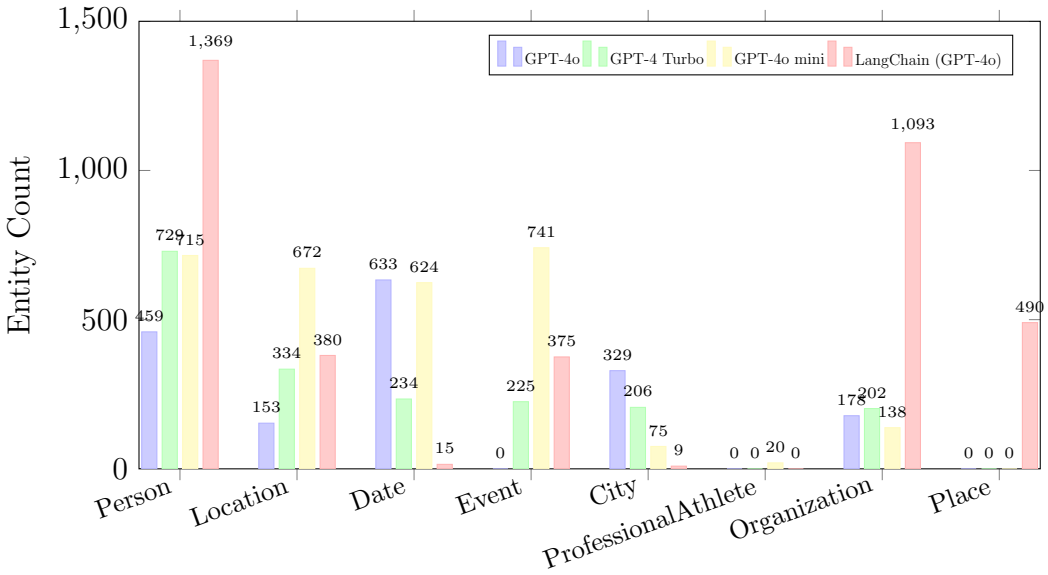

Figure 4.1: Entity Type Distribution across Models. Represents the top five most used entities for each model and their counts compared to the other models

and *"Location"* (153). Even though the top used labels in the nodes of the GPT-4 Turbo often overlapped, their occurrences differed. *"Date"* only occurred 234 times in the nodes, while *"Person"* had the highest frequency with 729 occurrences, which is approximately 13% more occurrences than the top label in the GPT-4o KG. The other top labels were *"Location"* (334), *"Event"* (225) and *"City"* (206). The label *"Organization"* was not found in the five most frequent labels in GPT-4 Turbo's KG like it was in GPT-4o.
This was also the case for the GPT-4o mini graph, which otherwise had some other overlapping labels in the top five labels. Here, the most frequently used label in the graph nodes was *"Event"* (741), i.e., it had about three times more occurrences than in the GPT-4 Turbo nodes. Though, the frequency of *"Person"* was of similar size compared to the count of *"Person"* labels in the GPT-4 Turbo KG, the other labels were *"Location"* (672), *"Date"* (624) and interestingly *"ProfessionalAthlete"* (207), which not only did not occur in the other graphs but also seemed more specific than the other labels.
LangChain's most frequently used class was *"Person"*, like it was for GPT-4 Turbo. However, *"Person"* was referenced about 46% less in the GPT-4 Turbo graph than in the LangChain graph, which can also be accounted for by the vastly different number of nodes seen in table 4.3. *"Person"* was followed by *"Organization"* (1093), *"Place"* (490), *"Location"* (380) and *"Event"* (375) as the top five classes with the highest frequency. Except for *"Place"*, these entity types aligned with the most common entities of the other graphs. The number of occurrences also fit the other results, only the frequency of *"Person"* diverged from the other top classes.

The frequency distribution analysis revealed that the GPT-4o KG contained 180 distinct entity labels that appeared only once in the graph. In comparison, the GPT-4 Turbo KG had 41 fewer such single-occurrence entities, indicating a slightly different distribution pattern. The GPT-4o mini KG referenced 255 distinct entity types only once, which is noteworthy given that 309 entities were introduced during

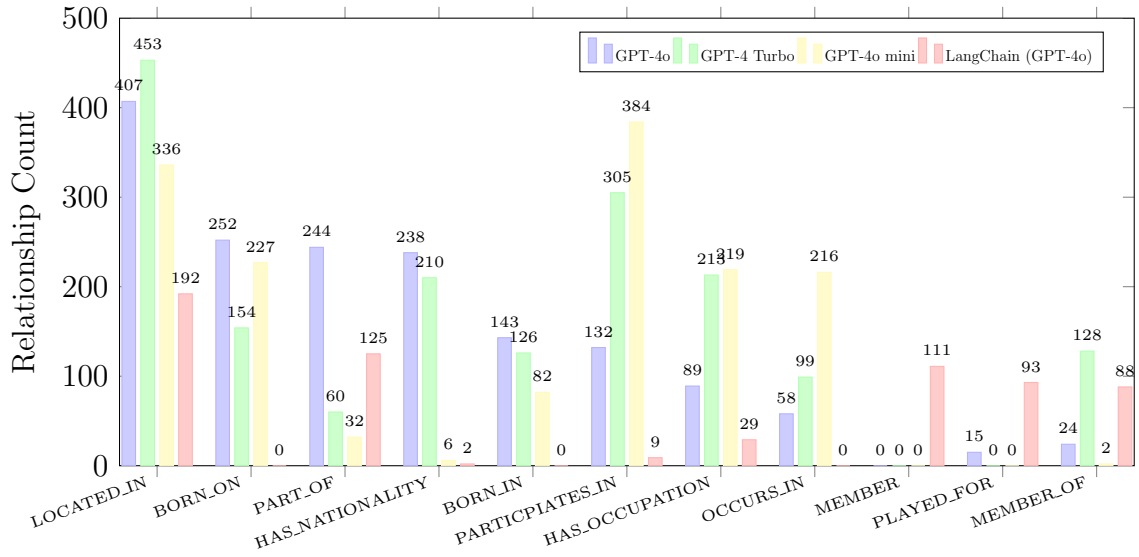

Figure 4.2: Relationship Type Distribution across Models. Represents the top five most used relations for each model and their counts compared to the other models

	GPT-4o	GPT-4 Turbo	GPT-4omini
Used Relationships	325	143	265
New Relationships	33	101	78

Table 4.5: Number of relationships from the ontologies used in the graph vs. number of relationships newly created in the KG generation step.

the KG generation process. This aligned with the overall number of nodes in the KG, suggesting a structured incorporation of additional entities. Interestingly, while LangChain produced the highest number of nodes among all KGs, it exhibited the lowest count of single-occurrence classes, highlighting a potential difference in how entity types were distributed within the graph.

In addition, the graph leverage of the relationships defined in the ontologies was also looked at. Table 4.5 presents the comparison between ontology-derived relationships and newly created relationships during KG generation. Similarly to the entity usage in the GPT-4o graph, the relationship analysis showed that 325 of the 423 ontology relationships were used in the final graph. Only 33 new relations were added to the schema during the generation of the graph. GPT-4 Turbo, however, used only half of the existing relationships and instead added 101 new ones, which did not match the entity usage described earlier. On the other hand, the GPT-4o mini KG created fewer new relations compared to GPT-4 Turbo but also compared to the number of new entities during KG generation, indicating that the relations derived during ontology extraction sufficed for the graph so that only 78 relations had to be added.

Furthermore, figure 4.2 showed that the distribution of relation types across the graphs varied slightly. In the GPT-4o graph, *"LOCATED_IN"* was the most frequent relationship type with over 400 occurrences. It was followed by *"BORN_ON"* (252), *"PART_OF"* (244), *"HAS_NATIONALITY"* (238), and *"BORN_IN"* (143) as the rest of the top five most frequent relations. These mostly aligned with the most

used entities, such as *"Date"*, *"Person"*, and *"Location"*, which could fit as subjects or objects for these predicates. This was also the case for the other graphs. GPT-4 Turbo also focused on the three entities amongst others, and these could be used with the highest occurring relations, *"LOCATED_IN"*, *"PARTICIPATES_IN"*, and *"HAS_OCCUPATION"*, which occurred 453, 305, and 213 times, respectively. The other top relationships, *"HAS_NATIONALITY"* (210) and *"BORN_ON"* (154), were also quite frequent in the GPT-4o graph, though with different distributions, as "BORN_ON" was used approximately 38% less in the GPT-4 Turbo KG than in the GPT-4o KG. This might be in alignment with the number of relations generated by both graphs that varied a lot.
GPT-4o mini created a lot more relationships than the other models in general, thus, the top five most common relationships were moderately utilized more often. Though type-wise, the relations were similar to the other most frequent ones. Notably, the top relation was *"PARTICIPATES_IN"* with a count of 384, and after that, it was followed by *"LOCATED_IN"* (336), *"BORN_ON"* (227), *"HAS_OCCUPATION"* (219), and *"OCCURS_IN"* (216).
However, the LangChain KG stood out the most with a very different distribution. Its most frequent relations were *"LOCATED_IN"* (192), *"PART_OF"* (125), *"MEMBER"* (111), *"PLAYED_FOR"* (93), and *"MEMBER_OF"* (88). Especially *"PART_OF"*, *"MEMBER"*, and *"MEMBER_OF"* were noteworthy because they all could be used to describe similar relationships and might even be duplicates. Further manual inspection showed that the LangChain KG contained more of these duplicate relationship types, e.g., due to representational inconsistencies, such as *"BIRTHPLACE"* and *"BIRTH_PLACE"*.

Moreover, the LangChain graph could be differentiated in the further frequency analysis of relationship usage. Unlike the other models, LangChain produced a much larger variety of relation types, with 835 relations that occurred only once. This suggested that while it captured a wide range of connections, many were used infrequently, potentially leading to a less structured or more fragmented graph. The graph was also generated with GPT-4o but using the two-step approach, at the same time, contained 121 unique relationships that occurred only once, i.e., only about 14.5% of LangChain's count. GPT-4 Turbo and GPT-4o mini produced even fewer once-occurring distinct relations, with 119 and 101 relations, respectively. This implied that these graphs had a more balanced distribution of the relationships than the LangChain KG.
Generally, it could be said that the labels and relationships used by each model reflected that they were all based on the same Wikipedia articles with some variation in which aspects of the texts were focused on more.

The syntactic validity of the triples was also an important quality dimension of this evaluation framework. A manual search of '' in the output Excel files showed that no entities or relations were left empty. This indicated that each model was able to fill the head entity, the relationship type, and the tail entity and resulted in a syntactic validity of 100%.

Further statistical evaluation revealed that the triples generated by each model overlapped across models. Overlapping triples were defined as triples with the same

Model 1	Model 2	Overlapping Triples
GPT-4o	GPT-4o mini	408
GPT-4o	GPT-4 Turbo	592
GPT-4o	LangChain 4o	203
GPT-4 Turbo	GPT-4o mini	345
GPT-4 Turbo	LangChain 4o	134
GPT-4o mini	LangChain 4o	135

Table 4.6: Number of overlapping triples between the graphs generated by the different models.

subject, predicate, and object. Table 4.6 showed that the triples of GPT-4 Turbo intersected the most with the GPT-4o mini triples, with 592 overlapping relationships. This was also the largest intersection between two models in all comparisons. Comparing all other models' triples with the triples of LangChain yielded the least matches, emphasizing the construction difference between the LangChain graph and the others. The worst match could be found between LangChain and both GPT-4 Turbo and GPT-4o mini, with overlaps of 134 and 135 triples, respectively, emphasizing the different naming and focus on certain entities between the approaches. The intersection of LangChain and GPT-4o was slightly better, with 203 intersected relationships, which could be traced back to the fact that the LangChain graph also utilized GPT-4o to create the graph but with a different prompt and approach. GPT-4o mini and GPT-4 Turbo matched quite well with the triples generated with GPT-4o. The overlapping triples between GPT-4o mini and GPT-4o counted 408 relations. Finally, the match between GPT-4o mini and GPT-turbo with 345 overlapping relationships was the third-highest match count of all comparisons. This is probably a result of the prompt engineering that resulted in these graphs and provided the models with the same few-shot examples.

Overall, these findings suggested that LLM-generated KGs can efficiently instantiate entity classes and properties but struggle to match the completeness and consistency of manually curated graphs like Wikidata. Furthermore, the results highlighted that GPT-4o mini performed the best in balancing graph size, ontology coverage, and property utilization. However, no model fully optimized both ICR and IPR simultaneously, leaving room for further refinement in ontology-to-KG alignment.

4.2.2 Knowledge Quality Evaluation

The results above already help indicate which graphs are generated better than others. Nonetheless, structural quality alone cannot validate a knowledge graph as useful or not. Semantic accuracy needs to be estimated as well to see whether the triples are meaningful and, thus, useful. Wikidata and DBpedia triples were used as gold triples, and exact and partial accuracy was calculated. The performance of each KG is presented in table 4.7.

GPT-4o outperformed all the other models in accuracy across metrics and the existing graphs. Its graph achieved the highest exact accuracy with the Wikidata triples, where approximately 23% of triples exactly matched the triples of Wikidata. Although the exact accuracy of GPT-4o for the DBpedia triples was worse than for the Wikidata triples, with 18% accuracy, it surpassed most of all other exact accuracy scores across models and gold triples. Interestingly, the best partial accuracy, about 0.72, was reached by GPT-4o for the DBpedia partial accuracy, not

	GPT-4o	GPT-4 Turbo	GPT-4omini	LangChain
DBpedia exact accuracy	**0.1829**	0.1477	0.136	0.0581
DBpedia partial accuracy	**0.7203**	0.638	0.6333	0.6328
Wikidata exact accuracy	**0.232**	0.1929	0.1154	0.0535
Wikidata partial accuracy	**0.5319**	0.5063	0.432	0.3272

Table 4.7: Results of the knowledge/content quality evaluation, i.e., comparing the triples with Wikidata & DBpedia triples.

Wikidata like the best exact accuracy. This indicated that even though Wikidata contained more exact matches with the KG, DBpedia generally covered more triples similar to the triples in the GPT-4o KG. This discovery also held for the other graphs. Most graphs had a higher exact accuracy with the Wikidata triples and more partial matches with the DBpedia triples, implying that though the relation labels or tail entities of DBpedia did not always match the predicted triple, a lot of the automatically generated triples were not completely wrong.

The GPT-4 Turbo KG showed a slightly worse performance than the GPT-4o KG, with an average of about 5% less accuracy in each metric. Nevertheless, it yielded better scores than the other two graphs, and with a Wikidata exact accuracy result, GPT-4 Turbo achieved the second-highest exact accuracy score overall with 0.1929. While this accuracy was less than 4% lower than the best exact accuracy by GPT-4o, it varied by 7% and 18% for GPT-4o mini and LangChain, respectively. GPT-4 Turbo performed well in partial accuracy, scoring 0.63 for DBpedia and 0.506 for Wikidata. The other two models, GPT-4o mini and LangChain, performed the worst in the knowledge quality evaluation, which mostly could be accounted for by the different relationship types GPT-4o mini and LangChain leveraged compared to GPT-4o and GPT-4 Turbo. Even though the Wikidata partial accuracy was at 0.43 for the GPT-4o mini KG and hence lower than the best two models, the DBpedia partial accuracy aligned with the results of GPT-4 Turbo with 63.3% and 63.8%, respectively. It demonstrated that the subject/object in the triples mostly fit with the DBpedia triples, although the relations might not have matched completely. Considering the number of triples generated by the GPT-4o mini graph, the accuracy could be said to be higher than GPT-4 Turbos since GPT-4o mini contained around 2000 more triples than the GPT-4 Turbo KG. The exact accuracies of 0.136 (DBpedia) and 0.1154 (Wikidata) were quite good as well if compared to GPT-4 Turbos' exact accuracies of 0.1477 (DBpedia) and 0.1929 (Wikidata).

LangChain showed the worst performance for the knowledge quality assessment. The relationship types found in LangChain's graph heavily deviated from the relations of the other automatically generated graphs, as well as from Wikidata and DBpedia. Since the relationships within the LangChain KG contained duplicates, no additional mapping of relations could be applied that mapped the Wikidata/DBpedia predicates to the LangChain ones besides the mapping, which was based on the triples generated by the other models. Due to this, the exact accuracy of the LangChain KG suffered a lot, returning a result of about 5.8% and 5.3% for DBpedia and Wikidata, respectively. Thus, an average of 62% and 67.4% fewer exact matches were found compared to the two-step approach KGs for DBpedia and Wiki-

	GPT-4o	GPT-4 Turbo	GPT-4o mini	LangChain
Nodes	26	114	101	3
Relationships	29	147	107	3

Table 4.8: Number of nodes that have cyclic relations to them and the number of overall cyclic relations in each graph.

data. Though LangChain's partial accuracies were also lower than all other partial accuracies, the divergence was not as big as that of the exact accuracies. With a Wikidata partial accuracy of approximately 0.33, it only performed 0.1 worse than the GPT-4o mini triples. The achieved partial accuracy for DBpedia triples merely varied by 0.0005 points when compared to the GPT-4o mini partial accuracy. This indicated that although the relation labels did not match well with the gold triples, the content of the triples was still relatively relevant.

The manual validation of the knowledge quality evaluation highlighted what type of triples caused misclassifications or were erroneous in general. Here, examples of these mistakes are listed with the respective graph they appeared in for better understanding.

1. *("Jake Fenlason", "HAS_OCCUPATION", "Jake Fenlason")* (GPT-4 Turbo)
2. *("São Paulo", "LOCATED_IN", "Gero Camilo")* (GPT-4o)
3. *("Li Zicheng", "PARTICIPATES_IN", "08/03/2016")* (GPT-4 Turbo)
4. *(William Norman Herbert", "HAS_OCCUPATION", "Northumberland Fusiliers")* (GPT-4 o Mini)
5. *("Major General William Norman Herbert", "BORN_IN", "Harwickshire")* (GPT-4o)

One common error of all automatically generated graphs was cyclic relationships, i.e., triples where the object was the same as the subject. This error occurred when the answer of the relation could be found in the properties of the subject node, e.g., the triple *("Jake Fenlason", "HAS_OCCUPATION", "Jake Fenlason")* found in the GPT-4 Turbo KG, where the node *"Jake Fenlason"* contained the property *"occupationTitle": "Soccer player"*. The exact number of nodes with cyclic relations and the number of cyclic relations in general were estimated using a Cypher query in the Neo4j graphs with the following Cypher query:

```
MATCH (n)-[r]->(n)
RETURN n AS subject, type(r) AS predicate, n AS object;
```

The responses showed how many nodes and relations were affected and were presented in table 4.8. Particularly, the GPT-4 Turbo generated KG demonstrated a high usage of cyclic relationships with 147 relationships referring to the same entity as head and tail entity. This might be related to the number of nodes in the KG. In the GPT-4o KG, only 26 nodes were affected, indicating that enough nodes were

created that the relation could reference. This also holds for the LangChain cyclic relations. The nodes generated in the LangChain graph were the highest number of nodes, and only three cyclic relations could be found in the KG. Nonetheless, GPT-4o mini had the second highest node count of all KGs but produced the second most cyclic relationship with 101 nodes affected. Both GPT-4 Turbo and GPT-4o mini had lower exact accuracy scores than GPT-4o, suggesting that some of the cyclic relations could be accounted for partial accuracy instead or could not be matched. An example for the latter was the triple *("Alexander Jamieson", "HAS_OCCUPATION", "Alexander Jamieson")*, which could not be matched in DBpedia. The triple *("Ania Ruiz", "HAS_OCCUPATION", "Ania Ruiz")*, on the other hand, could be partially matched with *("Ania Ruiz", "HAS_OCCUPATION", "volleyball player")*, and contained the property *"occupationTitle": "volleyball player"* showing that the connection the model, here GPT-4 Turbo, wanted to make was partially correct. Due to this, the relationship types, which mostly occurred in the cyclic relations across the graphs, were *"HAS_OCCUPATION"* and *"HAS_NATIONALITY"*, as objects of these relationships were often included in the properties of the corresponding subjects.

Furthermore, some triples showed that the models did not correctly understand the relation between the two entities. Amongst others, this was seen in example 2. The correct knowledge would have been *"Gero Camilo", "LOCATED_IN"/"RESIDES_IN", "São Paulo"*. Instead, the subject and object entities were falsely interchanged by GPT-4o.
Another common mistake can be seen in Example 3. The relationship types chosen by the models did not always reflect what the description of the relationship contained. Here, the triple description was *"Li Zicheng was prosecuted on March 8, 2016."*. The entities overlapped correctly, but the relation was wrong. Example 4 illustrates a similar mistake. The description of the relationship also did not match the relationship itself completely. The head entity and relation type suited the description *"William Norman Herbert served as colonel of the Northumberland Fusiliers"*. Yet, the tail entity was not captured correctly. Thus, only counting toward the partial accuracy, not the exact one.
Example 5 did not refer to a mistake per se. The subject, predicate, and object are correct but only partially accurate according to the gold triples. Due to the object entity being kept broad, the triple was not detailed enough to match the Wikidata and DBpedia triples exactly. The gold triples were more specific by naming the district *"Fillongley"* and not only the region *"Warwickshire"*.

All in all, it could be seen that GPT-4o achieved the highest accuracy across all evaluated metrics, with 23% exact accuracy for Wikidata and 18% for DBpedia, while also excelling in partial accuracy. GPT-4 Turbo followed closely behind, whereas GPT-4o mini and LangChain performed worse, with LangChain struggling the most due to its divergent relationship types and high rate of misclassified triples.

4.2.2.1 Question-Answering Evaluation

The final step to measure if a graph is useful was the use case. Since the graphs generated in this thesis were not based on domain-specific articles, the use case was a simple question-answering task. The responses of the graphs to the Cypher queries

	GPT-4o	GPT-4 Turbo	GPT-4o mini	LangChain
(1)	Frederick Charles Bartholomew, Reginald Edmund Compton Butterworth, Sir Philip Reginald le Belward Grey Egerton, Sir Richard Stanley Barratt	Norman Kendrew, Sir Philip Reginald le Belward Grey Egerton, 14th Baronet JP DL	Gertrude Macdonald	None
(2)	22 December 1894, Bucharest	Bucharest	22 December 1894	None
(3)	Document written by the philologist addressed to Albert I, denounces the Belgian organization Frontbeweging, occurs on 11.7.1917	Document authored King Albert I, occurs in World War I	Document (description: "An open letter addressed to King Albert I during World War I, expressing grievances of the Flemish within the Belgian Army."), addressed to Armand De Ceuninck, occurs in Yser Front,	Open Letter to the King of the Belgians Albert I, action of the Frontbeweging movement, provoked reaction from High command, related to the Flemish movement, related to the Belgian Army, related to Flamenpolitik, defended actions of the German authorities, context World War I, demanded autonomy for Flanders, grievances against the Belgian Army, defended Flamenpolitik

Table 4.9: Results of the QA task for each model. Question 3 was "summarized" by presenting the triples. The complete results can be found in Appendix A.3.3. Questions: (1) *Which people have an English nationality?*, (2) *When and where was Mihail Andricu born?*, (3) *What is the Yser Testament?*

on Neo4j varied for each graph and are presented in table 4.9 in a shortened format. The full responses (A.3.3), as well as the expected, correct responses (A.3.2), can be found in the Appendix.
Looking at the results, it could be seen that GPT-4o performed best for the first two questions. The answer to the first question *("Which people have an English nationality?")* is merely answered to 33% but found more answers than all the other graphs. GPT-4 Turbo found two of the 12 desired answers, while GPT-4o mini could only find one name. The LangChain graph showed the worst performance by returning no answers in question 1 and question 2. The results of the first question suggested that GPT-4o made the most useful connections between a *"Person"* entity and a *"Location"* entity with the relation *"HAS_NATIONALITY"*. This also aligned with the findings reported before, where it stated GPT-4o mini's and GPT-4 Turbo's number of cyclic relationships, especially concerning this relationship. Furthermore, the different graphs might have created different names for the relationship or multiple relationships representing the same relationship. For example, the LangChain KG comprises at least two relation types to express a *"Person"*s nationality, *"HAS_NATIONALITY"* and *"NATIONALITY"*. Yet, none of these relationships made a connection to the Nationality node representing Great Britain or an English nationality, which highlights the weak performance in the general knowledge quality evaluation.

Question 2 was only fully answered by one of the models, namely GPT-4o, which created the knowledge important to answer this question. GPT-4 Turbo and GPT-4 o Mini both could only establish one of these relationships. While GPT-4 Turbo was able to build the relationship between the node *"Mihail Andricu"* and his birthplace *"Bucharest"*, GPT-4o mini focused on his birthdate. Due to this, the Cypher query first generated by GPT-4o and then manually modified to fit the graphs, had to be changed even further, so that an answer could be given by the KG. These modifications can also be found in the Appendix (A.3.1). Like for question 1, LangChain could not form any of these two relationships for this node. Instead, the KG had

this information in the node properties, where both the date and place were correct. Other nodes, which also contained knowledge of birth date and birthplace, were connected by a triple. However, these relations varied in relation types, such as *"BIRTHPLACE"* or *"BIRTH_PLACE"*, which made querying for this knowledge difficult. Thus, this still suggested that LangChain performed worse for question-answering compared to the other graphs since these KGs also stored the information in the node properties and additionally built the relationships in a unified manner.

The last question should show whether the KGs were able to identify the key information from the articles and represent it properly. The correct response in A.3.2 is a slightly shortened version of the whole Wikipedia article from the Wikipedia dump. For this question, the answers of the various graphs deviated the most not only by several relations but also by content and content accuracy. Here, LangChain's response contained 11 triples referring to the *"Yser Testament"*. The relation types and subject and object nodes can be found in the article, indicating that LangChain was able to obtain important knowledge from the article, with only a little key information missing, included in the last sentence of the article.
The other KGs, on the other hand, reached their limit with this question. GPT-4o's graph returned three triples, from which two were correct: *"addressed to Albert I"*, and *"occurs on 11.7.1917"*. The third triple contained incorrect knowledge as the *"Yser Testament"* was a letter by the *"Frontbeweging"* and did not denounce it. Looking at the description of this relation, the correct information was stated in it: *"The Yser Testament criticized the actions of the Belgian Army."*. However, the object node of the triple was mapped incorrectly. Instead of *"Belgian Army"*, it connected the subject to *"Frontbeweging"*, thus, generating wrong knowledge similar to example 4 from earlier. Likewise, GPT-4 Turbo also produced an incorrect triple, which falsely stated that the letter was *"authored"* by *"Albert I."*. Still, the description of the relationship indicates that it was *"addressed to Albert I"*. This referred to the same error examined earlier (see example 3), where the relation type chosen by the model did not reflect the actual relationship. Besides the triple with the incorrect relationship, the GPT-4 Turbo model was only able to create one completely correct triple regarding the *"Yser Testament"*, i.e., that this letter was sent in World War I.
GPT-4o mini was the only model that included a description of the *"Yser Testament"* in its node properties. The content of this description expressed the key knowledge of the Wikipedia article. Concerning the information represented in the triples, equivalent to the erroneous object entity in the GPT-4o triple *("The Yser Testament", "DENOUNCES", "Frontbeweging")*, the triple of the GPT-4o mini graph describing that the letter was addressed to *"Armand De Ceuninck"* had the incorrect mapping to the node of *"Armand De Ceuninck"* instead of *"Albert I."*. The description of the relationship, however, represented the correct knowledge of the article.

These results suggested that the graph generated with GPT-4o presented knowledge in a unified structure so that easier questions, such as identifying nationalities or birthplaces, were answered more effectively than by other models. Hence, GPT-4o was able to demonstrate its ability to establish key relationships, while the other models struggled with inconsistent relation labels and incorrect entity mappings. On the other hand, the graph created with LangChain's LLMGraphTransformer showed

the ability to extract key information from an article and represent it in a knowledge graph, although its knowledge representation was less consistent across articles compared to GPT-4o's structured representation of the same Wikipedia content.

5. Discussion and Future Research

After extensive evaluation of each knowledge graph on multiple levels, the results of this study demonstrate the potential of LLMs in generating KGs. This chapter discusses the main findings considering the research questions, highlighting the strengths and limitations of the proposed approach and exploring how different models and methodologies impact structural quality, knowledge completeness, and overall usefulness in knowledge retrieval tasks.

Predominantly, the structural quality analysis indicated that an entity and relationship-dense ontology covers a broader range of concepts and nuances. However, this also increased structural complexity, which could be seen in the results of the inverse multiple inheritance (IMI). GPT-4o's ontology included more entities and relationships than GPT-4o mini's but also had a more complex hierarchy due to a child class often having multiple levels of parent entities. One of the reasons for that might be the wrongly added *"Occupation"* class GPT-4o's hierarchy, which was a child entity of the *"Person"* entity and parent entity to other classes representing the profession of a *"Person"*. This additional class between *"Person"* and, e.g., *"Artist"* or *"Author"* increases the complexity of the ontology as seen in the IMI. So if the few-shot example in the ontology extraction prompt was adjusted to make it clearer that *"Occupation"* is not a child entity of *"Person"*, the complexity could be reduced, but the entity and relation count could stay of similar magnitude.

It generally can be said that of all graphs created with the proposed method, the approach worked the best with GPT-4o. Even though most structural metrics were lower than the other LLMs' performances, the KG ICR, which measured how many entities of the ontology were used in the KG, showed a much better performance compared to Wikidata and DBpedia. In addition, its ontology ICR was the best of all generated graphs. The lower KG ICR compared to GPT-4o mini's ICR can be explained with the amount of new entity types added during the KG generation step to the ones from the ontology extraction step (see 4.4). GPT-4o mini added 309 new entity types, likely due to the size of ontology-derived entities, but 93% of these classes were used, resulting in a high ICR. GPT-4o, on the other hand, created more entities during ontology extraction, of which only 78% were actually used. Nevertheless, these made up 80% of all entity types in the final graph, while GPT-4o mini's ontology-derived classes only accounted for 38% in the KG. This suggested that the ontology extracted by GPT-4o showed better quality despite low structural quality metrics.

The usage of the ontology-derived properties was measured with the instantiated property ratio (IPR). The results indicated that most properties were instantiated

GPT-4o	GPT-4 Turbo	GPT-4o mini
”name”: ”string”, ”birthDate”: ”date”, ”birthPlace”: ”string”, ”nationality”: ”string”, ”occupation”: ”string”, ”genre”: ”string”, ”awards”: ”[string]”, ”notableWorks”: ”[string]”	”name”: ”string”, ”birthDate”: ”date”, ”birthPlace”: ”string”, ”gender”: ”string”, ”nationality”: ”string”, ”genres”: ”[string]”, ”notableWorks”: ”[string]”	”name”: ”string”, ”birthDate”: ”date”, ”birthPlace”: ”string”, ”genre”: ”string”, ”awards”: ”string”

Table 5.1: Properties for the *”Author”* class defined in each of the three generated ontologies

in the GPT-4o mini Graph, however, the differences between the KGs, especially to GPT-4o, were not significant. The high instantiation of the GPT-4o mini KG, as well as the highest subclass property acquisition (SPA), can be traced back to the fact that fewer properties were included for the entity types, which could be seen during a simple and short manual evaluation at first glance.
Table 5.1 shows the difference for one example, the properties of the *“Author”* entity, where the GPT-4o mini ontology contained fewer properties. Furthermore, the inheritance of properties did not work properly in the GPT-4o mini, as the *“gender”* property of the parent class *“Person”* was not inherited by the child class *“Author”*. This makes comparing the IPR and SPA/SPA V2 more difficult while highlighting the necessity of a diverse evaluation framework. Without looking at examples of the constructed ontologies, the metrics indicate that GPT-4o mini created the best ontology and KG. However, fewer properties were created, increasing the IPR compared to GPT-4o’s KG, which has a lower IPR but instantiated maybe the same or even better, just with a higher property rate. GPT-4o’s KG might instantiate fewer properties and add fewer properties during inheritance, measured by the SPA and SPA V2 metrics, still, it adhered to the inheritance rules given in the prompt A.1.4, exhibiting a better textual understanding of the instructions than GPT-4o mini.

GPT-4 Turbo’s ontology showed the worst performance on various levels. During construction, the property extraction had to be re-run with GPT-4o as half of the entities were not assigned any properties. This already demonstrated a lack of understanding of hierarchical structures, which were given to the model in addition to the prompt, as well as an inability to recognize which properties could generally belong to an entity. The mix of models, i.e., GPT-4 Turbo for extracting the “basic” ontology and hierarchy and GPT-4o for generating the properties, might have caused the bad performance at the ontology level.
GPT-4 Turbo’s bad performance alone, however, was noticeable when looking at the IMI. It was the most complex graph, with an IMI of 0.64 compared to 0.82 and 0.84 of GPT-4o and GPT-4o mini. The mix of models is not the only reason for that, as the metric is restricted to the hierarchy alone, which was created with GPT-4 Turbo. A glance at the hierarchy supported the low IMI, too, as it was quite unstructured. Unnecessary and partly wrong parent-child relations were included in the hierarchical representation, such as *“Person”* – *“Occupation”* –*“Composer”* – *“ComposerSociety”* as parent-child structure, indicating that *“ComposerSociety”* is

a child of *"Person"*, *"Occupation"* and *"Composer"*. The class *"ComposerSociety"*, thus, inherited all properties from its multiple superclasses, which did not only increase complexity but also produced wrong properties for the class. Also, the ICR indicated that "unnecessary" entities were created in the ontology and not used in the KG. This occurred extensively for GPT-4 Turbo's ontology and graph, showing that these are of worse quality than the others.

Furthermore, the performances of the LLMs, GPT-4o, GPT-4 Turbo, and GPT-4o mini for the other quality criteria undermined the statement that GPT-4o could generate the best knowledge graph. Looking at the cyclic relationships, each of the three graphs generated with the proposed approach and the models contained, GPT-4o contained the least. This indicated that GPT-4o was able to understand the prompt the best as it instructed the model to *"[t]ry avoiding recursive statements"* (see A.1.5). Similarly, it was the best model to understand the concept of property inheritance in the property prompt (see A.1.4).

In general, the investigated quality dimensions, syntactic validity, conciseness, completeness, and semantic accuracies were only partially fulfilled by the graphs. All models were able to create syntactic valid triples for the KGs, but performance varied for the other dimensions. The complex structure of GPT-4 Turbo's KG indicated bad conciseness. Since GPT-4o mini's graph did not adhere to the hierarchy generation instructions and included hierarchy in the *"Other"* class, unnecessary complexity was added. GPT-4o wrongly assigned all professions as children to the *"Occupation"* class, complicating the hierarchy. Even though the ontology and graph were based on the articles to represent in the KG, unused entities, relations, and properties were created, as suggested by ICR and IPR. Hence, all models had problems keeping the ontology and graph concise.

The knowledge quality evaluation, including the question-answering task, was conducted to validate semantic accuracy and completeness. The results here showed that the triple structure of the automatically generated graphs deviated from the triples in Wikidata and DBpedia. A broad mapping between the relations of Wikidata or DBpedia and the new KGs helped to achieve some exact accuracy. The low exact and partial accuracies showed that the focus of the graphs differed between the predicted and gold triples. Moreover, it could be seen that the relationship types created by the automatically generated KGs overlapped more often with Wikidata's relationships, resulting in higher accuracies when Wikidata's triples were the gold triples. The models provided by OpenAI are black-box models, and training data is kept private, so it could be the case that Wikidata was used as training data for the models, which would explain the similarities in naming the relations.

Additionally, the low exact accuracies with the DBpedia gold triples were caused partially by the missing predicate. Instead of having relationship types like Wikidata and the automatically generated KGs, DBpedia often just provided *"Wikidatalink"* as a predicate with no option to trace which relationship type was referenced. Partial accuracies were able to capture these triples if the subject and object were correct. Furthermore, triples that are correct if checked manually but do not exactly match the Wikidata or DBpedia triple, such as example 5, could be counted to the partial

accuracy. Including examples like these was important as they convey correct information but are, e.g., slightly more broad or sometimes more specific than the gold triples and would otherwise count as wrong triples.

Question-answering responses also highlighted the limitations in the completeness of the graphs, especially GPT-4 Turbo's and GPT-4o mini's KGs, which could not answer any of the questions fully. GPT-4o performed better but did not contain all the important information from the articles. The knowledge within the graphs generated with the proposed methods was kept general, and most of the time, merely information from the first few sentences was included, while the rest was left out. Thus, the completeness of the graphs was quite bad.

Semantic accuracy was not as bad but also was not ideal. Many included inaccurate and incomplete object nodes and most likely hallucinated relationship types. The results of the manual triple evaluation exhibited different mistakes in the generated triples from the different graphs. These demonstrate some of the limitations of leveraging LLMs for knowledge graph construction. Mistakes, as seen in examples 1, 2, and 4, would not happen to a domain expert. It shows the troubles of LLMs, even more advanced models, with reasoning tasks.
In examples like 1, the models might have lacked the capability to find the corresponding node in the KG or even the value in the subject node to fit the subject and relation of the triple, thus, returning the subject entity instead, as it contained the relevant object in the subject's node properties. Furthermore, the less advanced models, such as GPT-4o mini, due to its size, and GPT-4 Turbo, due to its "age", were not able to adhere well to the prompt instructions trying to restrict cyclic relations. GPT-4o also could not avoid these errors but was able to keep it minimal with less than 1% of all triples containing recursive statements. Since statements like these do not carry knowledge in it, they are not contributing to the coherence, completeness and conciseness of the graph.

Evaluating the knowledge quality of the automatically generated KGs, also revealed more information about the problem of hallucinations, which was discussed in a lot of papers before, e.g. [26]. Only manual evaluation of a random sample of the generated triples in comparison to Wikidata and DBpedia could identify errors like these because they do not fall under any of the other metrics evaluated. [26] differentiated between content and ontology hallucinations. Hallucination errors found in the graphs of this thesis predominantly fell under content hallucinations.
Especially, errors such as example 3 from the results chapter might fall under the category of content hallucinations. The model understood the relationship description well but does not find the corresponding relationship type in the ontology or newly created relation types, so it hallucinates the "correct" type instead. This resulted in semantically incorrect triple statements, even though the relation description might be correct. Similarly, errors, like example 4 in GPT-4o mini's graph, reduce the quality of a graph. Because the object node is not completed, the triple does not convey useful knowledge but is also not wrong per se. In multiple occurrences, it seemed that the models stopped generating the tail entities or misunderstood the relations between the nodes and the role of the node itself. Instead of inferring that *"William Norman Herbert"'s* profession was colonel at the *"Northumberland Fusiliers"*, the

model did not create a node that contains this knowledge. So, the model had to hallucinate in finding a tail entity. This hallucination error could be classified as an ontology hallucination as it generated an invalid path according to the ontology. The GPT-4o mini ontology specifies that the relationship *"HAS_OCCUPATION"* can only occur between a *"Person"* and an *"Occupation"*. The node *"Northumberland Fusiliers"*, however, is labeled as a *"Location"*, which in itself is also inaccurate as only *"Northumberland"* is a location, but the node refers to a regiment in the British Army. Thus, the ontology rules were disregarded, making this an error of content hallucination.

Hallucination errors occurred in all graphs across models and approaches, however, in these samples, the only errors accounted for hallucination were hallucinated relationship types. In the future, these might be handled with further tweaking of the prompt, e.g., adding a statement to not include triples, where the model is not sure if it is correct. Otherwise, further postprocessing of the triples by identifying the predicate in the relationship description and comparing it to the actual relation type chosen by the model might mitigate the errors. When hallucination errors are detected, a better relationship might be selected from the ontology or graph schema. Nevertheless, reducing the number of hallucinations in the graphs is important as these worsen the truthfulness quality criterion. The hallucinations might be a reason that (domain) experts still need to validate the resulting KGs to find hallucinations and adapt these errors to maintain the semantic correctness of the graph. Ultimately, even a perfectly structured graph holds no value if the knowledge is inaccurate and makes the KG useless.

Interestingly, the difference between the GPT-4o graph generated by the proposed approach and the LangChain graph was quite noticeable, though LangChain's approach leveraged the GPT-4o model too. The GPT-4o model with the LLMGraphTransformer performed quite badly across the structural and content levels. It created a lot of duplicate entities and relations due to inconsistent naming, such as *"WON"/"AWARDED"/"WON_AWARD"/"RECEIVED"*, which made the comparison to other graphs and knowledge retrieval complicated. Nevertheless, it had the highest node and relation density of all graphs, which was especially beneficial when answering the third question in the question-answering evaluation, outperforming the other models. GPT-4o with the two-step method, on the other hand, created consistent class and relationship labels, achieved high accuracy scores and was able to answer the questions needing a well-structured graph to be answered. Yet, the lower node and relationship density as well as the discrepancies between relation type and actual relationship description, resulted in a bad performance answering the third question.
However, these deviations between the graphs generated with the presented approach using GPT-4o and LangChain's LLMGraphTransformer with GPT-4o highlighted the good performance of the two-step KG generation approach and the importance of prompt engineering. Even though the same LLM was used by both techniques, the resulting graphs were completely different. LangChain's KG had less structure than GPT-4o's KG. This might be a result of splitting the KG generation task into multiple sub-steps, most importantly into ontology and KG extraction. On the other hand, LangChain's KG created more nodes and relations, which could convey more

knowledge, while the two-step approach with GPT-4o resulted in the second-lowest number of relations. The question-answering results supported this finding, as the GPT-4o KG surpassed all other graphs in questions that needed good structure, while the last question showed that the LangChain KG contained more knowledge of the article than the others. This was also established during the manual evaluation of the triples. The GPT-4o triples were restricted to broad knowledge and did not contain all key information needed to represent each article well enough.

These results indicate that KG generation from unstructured texts benefits from adding ontology as context. Thus confirming the hypothesis that an approach separating ontology extraction and graph construction surpasses one-step approaches. The best graph, created with GPT-4o and generated with the proposed method, performed better than the LangChain approach, which generated ontology and graph with the same model. The limitation of the GPT-4o two-step method, i.e., the incompleteness of the model and restricting triple generation to broad knowledge, might be mitigated with prompt engineering. Looking at the prompt provided by LangChain, it instructs the LLM to *"[a]ttempt to extract as many entities and relations as you can."* [1]. Adding a statement conveying this content might improve the amount of returned triples, thus also generating a graph, which covered more knowledge relevant for questions like question 3.

The question of whether LLMs could generate useful knowledge graphs was more difficult to answer. The first challenge was to find a definition of a useful knowledge graph. The usefulness of a KG was assessed by evaluating structural and knowledge quality through various metrics and statistics, combining metrics and definitions of previous frameworks. This way, a new evaluation framework was introduced that not only concentrates on one of the quality criteria but more, i.e., the structural integrity of ontology and KG, the semantic accuracy and the application in a question-answering task. All these criteria together can point towards whether the graph is more or less "fit for use" or useful.
Evaluating a KG based on merely one or two aspects can cause erroneous decisions. Without additional statistics and knowledge quality analysis in the framework, the graph would be validated using only structural metrics. In this case, the GPT-4o mini graph would have been the most useful as it showed the best structure. Content evaluation and analyzing the usage of the ontology-derived classes demonstrated that this was not the case, and metrics like ICR were the result of non-sufficient entity creation in the "basic" ontology extraction step, which led to the creation of several new classes in the KG generation step.

However, analysis is still limited as no unified definition of a useful knowledge graph exists. So, while the results in this thesis can be measured by these criteria and be evaluated as useful or not, other research can define usefulness differently. This makes none of the results comparable across papers. The here proposed definition could be a good starting point to develop a clear-cut definition for KG quality evaluation that can be universally used to mitigate the problem of inconsistent graph

[1] https://github.com/langchain-ai/langchain-experimental/blob/main/libs/experimental/langchain_experimental/graph_transformers/llm.py

validation.

A direct comparison of these findings with other existing methods, such as studies introduced in the background section (see chapter 2), could not be made, as most of the other experiments used different settings and evaluation metrics. However, the results could be vaguely compared to similar experiments. [29], for example, also constructed a knowledge graph from text but measured the quality with precision, recall, and F1-measure, which is why an exact comparison is not possible. The final conclusion, on the other hand, could be compared and demonstrated similar mistakes in the error analysis of the triples to the errors presented before. Hallucinations of objects and subjects, as well as ontology hallucinations, were identified as errors there. The precision, recall, and F1 values were also similar to the partial accuracies of this experiment. However, the results from this thesis were based on text of variable lengths, while [29]'s experiment was conducted on sentence-level, indicating that the here-proposed method worked slightly better.
A similar comparison could be done with the study by [22], which constructed a knowledge graph from Wikipedia articles with multiple sub-steps, such as entity extraction, coreference resolution, relation extraction, etc. The precision, recall, and F1-measure again suggest a similar performance to the here-presented approach. Since more substeps were used to generate the KG in [22]'s approach, it can be said that similar performance can be reached with less effort and steps leveraging LLMs.

Although no exact comparison can be made, it can be said that the quality of [12]'s KG, constructed with a two-step approach, surpassed the quality of the graphs from this thesis, which can be seen at the F1-measure of 0.78. The two generated graphs, however, were created for specific domains, batteries, and the attack of the Hamas on Israel. This indicates that domain-specific graphs might be easier to construct than general graphs, as entities and relations do not have to be created across domains. [12]'s findings also support the good performance of the two-step approach used in this thesis.
Furthermore, [42]'s work was already compared in the results chapter (4.2). Nevertheless, the comparisons made, especially between the automatically generated graphs and the Raftel graph, are not substantial. Raftel contains data in Korean, which is difficult to put in contrast with the graphs with English data. Moreover, [42] only conducted a structural analysis of the graphs, which is not ideal, as indicated earlier, because multiple quality criteria, not only structural quality, play a role when evaluating the quality of a graph.

Even though this study demonstrates the potential of LLM-generated knowledge graphs, various limitations and challenges persist, hinting at opportunities for future research. This experiment was conducted with a broad variation of domains, making few-shot prompting more difficult. If a domain or specific use-case is known, the example JSONs ontology and graph could be improved for that purpose. Not only might this increase the structural quality due to more consistency and completeness, but it might also resolve problems such as assigning wrong relationship types, but having correct relation descriptions due to missing corresponding relation labels. There is also the possibility that the complexity and overall hierarchy problems could be solved because the *"Other"* entity type would not be of relevance as

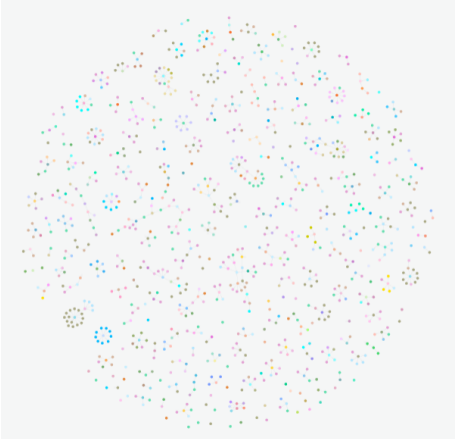

Figure 5.1: The complete Neo4j graph representation of the LLM-generated GPT-4o graph

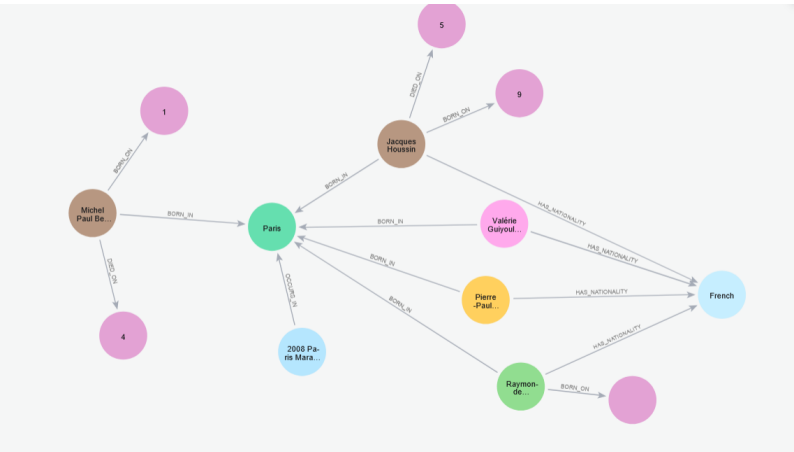

Figure 5.2: Connected knowledge across articles in the GPT-4o KG

the hierarchy of a specific domain is easier to create than one for any topic. Furthermore, additional metrics might be possible to use if the KG is restricted to one domain because not all of the structural quality metrics introduced by [42] could be applied to the here-created graphs due to their size, such as the class instantiation metric. Figure 5.1 illustrates the magnitude of classes and relations in the GPT-4o KG, showing how widely spread the graph is and that the graph is not very dense, i.e., containing multiple smaller "graphs" for each topic or article in the dataset. Only a few connections between the articles were possible, as depicted in figure 5.2, otherwise, it seemed there was not much common knowledge within the articles and thus the graph. A more specific dataset could create a more dense, and thus, maybe a more useful knowledge graph for that domain.

Considering that the LLMs in the here-proposed approach did not include knowledge of the complete articles, upcoming investigations could attempt different preprocessing techniques. Chunking approaches could separate the articles to sentence-level and use these sentences as input for ontology and KG construction. While one article is processed, the same thread could persist and be deleted as soon as a new article is handled to mitigate inconsistencies within an article, maintaining already

created entities and relationships.
Additionally, while writing this thesis, many new large language models were released, which claim to be more advanced than GPT-4o, such as GPT-o3. Future experiments could also explore the KG generation task with this model for even better performance because, as seen in [26] and in this study, more advanced models performed better.
Another factor to consider is the cost of using LLMs, such as OpenAI's GPT models. Looking at real-world use cases, it can be argued that the proposed method might be too expensive depending on the data size due to the high usage of OpenAI models and embeddings. Since OpenAI's newest and cheapest model, GPT-4o mini, did not perform across the various quality criteria, future research could explore how cost-effective open-source models, such as Llama 3[2], perform using the proposed approach.

Overall, this study highlights both the potential and limitations of using LLMs for knowledge graph construction, demonstrating that including the ontology for a more structured generation can enhance consistency and conciseness, but the approach still requires improvement. Future research should focus on integrating further prompt refinement and iterative chunking techniques to increase the accuracy and reliability of LLM-generated graphs.

[2]https://ai.meta.com/blog/meta-llama-3/

6. Conclusion

This thesis investigated the potential of large language models (LLMs) for automatically generating useful, high-quality knowledge graphs (KGs) from unstructured text. Here, a two-step approach was introduced, which first extracted an ontology with a hierarchical structure from the texts and then generated the KG from the articles, additionally using the ontology to provide structural guidance for the KG. Furthermore, an evaluation framework was introduced, which covered structural, knowledge, and question-answering quality. The results demonstrated that the two-step approach can construct a structured and coherent graph, especially when used with GPT-4o for both steps. This confirmed that the additional context provided by the ontology helped mitigate some challenges often encountered during KG construction, such as duplicates, structural complexity, and incorrect property inheritance. However, the limitations of leveraging LLMs for KG generations, like relations, object and ontology hallucinations, emphasized the ongoing challenge and which areas need further refinement.

A significant finding was the difference in the results of the here-proposed approach and LangChain's one-step approach with the LLMGraphTransformer. Even though both methods utilized OpenAI's GPT-4o, the outcomes deviated substantially. While LangChain's approach resulted in a graph with less structure and many inconsistencies and duplicates, it largely covered the content of the articles and included more important knowledge than the result of the two-step method. In comparison, the GPT-4o graph generated with the introduced approach showed fewer inconsistencies and more structure, though sometimes at the expense of the content coverage within the graph, as often the knowledge of only the first few sentences was captured by the KG.
The validation of the generated knowledge graphs using the proposed evaluation framework emphasized the need for a unified quality assessment approach to allow for meaningful comparisons between the KG generation approaches. Furthermore, the significance of including metrics for various quality dimensions to evaluate the graph quality as detailed as possible and prevent misinterpretations if only a few quality criteria were considered.

In conclusion, this study confirms that while LLMs are promising tools for automatically generating KGs, the field is still in its early stages. Future research should handle limitations, such as hallucinations and incomplete knowledge, to ensure factual correctness and usability. Exploring more advanced prompt engineering, chunking techniques, and a refined, unified quality assessment framework can enhance the quality of knowledge graphs generated with LLMs while making them more practical for real-world applications.

Bibliography

[1] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia - A crystallization point for the Web of Data. *Journal of Web Semantics*, 7(3):154–165, Sept. 2009. ISSN 15708268. doi: 10.1016/j.websem.2009.07.002. URL https://linkinghub.elsevier.com/retrieve/pii/S1570826809000225.

[2] A. Bosselut, H. Rashkin, M. Sap, C. Malaviya, A. Celikyilmaz, and Y. Choi. COMET: Commonsense Transformers for Automatic Knowledge Graph Construction, June 2019. URL http://arxiv.org/abs/1906.05317. arXiv:1906.05317 [cs].

[3] J. Boylan, S. Mangla, D. Thorn, D. G. Ghalandari, P. Ghaffari, and C. Hokamp. KGValidator: A Framework for Automatic Validation of Knowledge Graph Construction, Apr. 2024. URL http://arxiv.org/abs/2404.15923. arXiv:2404.15923 [cs].

[4] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, and T. Henighan. Language Models are Few-Shot Learners. 2020.

[5] S. Carta, A. Giuliani, L. Piano, A. S. Podda, L. Pompianu, and S. G. Tiddia. Iterative Zero-Shot LLM Prompting for Knowledge Graph Construction, July 2023. URL http://arxiv.org/abs/2307.01128. arXiv:2307.01128 [cs].

[6] B. Chen and A. L. Bertozzi. AutoKG: Efficient Automated Knowledge Graph Generation for Language Models, Nov. 2023. URL http://arxiv.org/abs/2311.14740. arXiv:2311.14740 [cs].

[7] H. Chen, G. Cao, J. Chen, and J. Ding. A Practical Framework for Evaluating the Quality of Knowledge Graph. In X. Zhu, B. Qin, X. Zhu, M. Liu, and L. Qian, editors, *Knowledge Graph and Semantic Computing: Knowledge Computing and Language Understanding*, volume 1134, pages 111–122. Springer Singapore, Singapore, 2019. ISBN 9789811519550 9789811519567. doi: 10.1007/978-981-15-1956-7_10. URL http://link.springer.com/10.1007/978-981-15-1956-7_10. Series Title: Communications in Computer and Information Science.

[8] Z. Chen, H. Mao, H. Li, W. Jin, H. Wen, X. Wei, S. Wang, D. Yin, W. Fan, H. Liu, and J. Tang. Exploring the Potential of Large Language Models (LLMs)in Learning on Graphs. *ACM SIGKDD Explorations Newsletter*, 25(2):42–61, Mar. 2024. ISSN 1931-0145, 1931-0153. doi: 10.1145/3655103.3655110. URL https://dl.acm.org/doi/10.1145/3655103.3655110.

[9] R. Cohen, M. Geva, J. Berant, and A. Globerson. Crawling the Internal Knowledge-Base of Language Models, Jan. 2023. URL http://arxiv.org/abs/2301.12810. arXiv:2301.12810 [cs].

[10] D. Dessì, F. Osborne, D. Reforgiato Recupero, D. Buscaldi, E. Motta, and H. Sack. AI-KG: An Automatically Generated Knowledge Graph of Artificial Intelligence. In J. Z. Pan, V. Tamma, C. d'Amato, K. Janowicz, B. Fu, A. Polleres, O. Seneviratne, and L. Kagal, editors, *The Semantic Web – ISWC 2020*, volume 12507, pages 127–143. Springer International Publishing, Cham, 2020. ISBN 978-3-030-62465-1 978-3-030-62466-8. doi: 10.1007/978-3-030-62466-8_9. URL https://link.springer.com/10.1007/978-3-030-62466-8_9. Series Title: Lecture Notes in Computer Science.

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

[12] L. Ding, S. Zhou, J. Xiao, and J. Han. Automated Construction of Theme-specific Knowledge Graphs, Apr. 2024. URL http://arxiv.org/abs/2404.19146. arXiv:2404.19146 [cs].

[13] P. L. Dognin, I. Padhi, I. Melnyk, and P. Das. ReGen: Reinforcement Learning for Text and Knowledge Base Generation using Pretrained Language Models, Aug. 2021. URL http://arxiv.org/abs/2108.12472. arXiv:2108.12472 [cs].

[14] L. Ehrlinger and W. Wöß. Towards a Definition of Knowledge Graphs. 2016.

[15] Q. Guo, Y. Sun, G. Liu, Z. Wang, Z. Ji, Y. Shen, and X. Wang. Constructing Chinese Historical Literature Knowledge Graph Based on BERT. In C. Xing, X. Fu, Y. Zhang, G. Zhang, and C. Borjigin, editors, *Web Information Systems and Applications*, volume 12999, pages 323–334. Springer International Publishing, Cham, 2021. ISBN 978-3-030-87570-1 978-3-030-87571-8. doi: 10.1007/978-3-030-87571-8_28. URL https://link.springer.com/10.1007/978-3-030-87571-8_28. Series Title: Lecture Notes in Computer Science.

[16] S. Hao, B. Tan, K. Tang, B. Ni, X. Shao, H. Zhang, E. P. Xing, and Z. Hu. BertNet: Harvesting Knowledge Graphs with Arbitrary Relations from Pretrained Language Models. 2022. doi: 10.48550/ARXIV.2206.14268. URL https://arxiv.org/abs/2206.14268. Publisher: [object Object] Version Number: 3.

[17] M. Hofer, D. Obraczka, A. Saeedi, H. Köpcke, and E. Rahm. Construction of Knowledge Graphs: State and Challenges, Oct. 2023. URL http://arxiv.org/abs/2302.11509. arXiv:2302.11509 [cs].

[18] E. Huaman. Steps to Knowledge Graphs Quality Assessment, Aug. 2022. URL http://arxiv.org/abs/2208.07779. arXiv:2208.07779 [cs].

[19] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S. Yu. A Survey on Knowledge Graphs: Representation, Acquisition and Applications. *IEEE Transactions on*

Neural Networks and Learning Systems, 33(2):494–514, Feb. 2022. ISSN 2162-237X, 2162-2388. doi: 10.1109/TNNLS.2021.3070843. URL http://arxiv.org/abs/2002.00388. arXiv:2002.00388 [cs].

[20] Z. Jiang, F. F. Xu, J. Araki, and G. Neubig. How Can We Know What Language Models Know? 2019. doi: 10.48550/ARXIV.1911.12543. URL https://arxiv.org/abs/1911.12543. Publisher: [object Object] Version Number: 2.

[21] N. Kandpal, H. Deng, A. Roberts, E. Wallace, and C. Raffel. Large Language Models Struggle to Learn Long-Tail Knowledge. 2023.

[22] N. Kertkeidkachorn and R. Ichise. An Automatic Knowledge Graph Creation Framework from Natural Language Text. *IEICE Transactions on Information and Systems*, E101.D(1):90–98, 2018. ISSN 0916-8532, 1745-1361. doi: 10.1587/transinf.2017SWP0006. URL https://www.jstage.jst.go.jp/article/transinf/E101.D/1/E101.D_2017SWP0006/_article.

[23] V. K. Kommineni, B. König-Ries, and S. Samuel. From human experts to machines: An LLM supported approach to ontology and knowledge graph construction, Mar. 2024. URL http://arxiv.org/abs/2403.08345. arXiv:2403.08345 [cs].

[24] LangChain. Constructing knowledge graphs, 2024. URL https://python.langchain.com/v0.1/docs/use_cases/graph/constructing/. Accessed: 2025-02-01.

[25] Y. Li, R. Zhang, J. Liu, and G. Liu. An Enhanced Prompt-Based LLM Reasoning Scheme via Knowledge Graph-Integrated Collaboration. 2024. doi: 10.48550/ARXIV.2402.04978. URL https://arxiv.org/abs/2402.04978. Publisher: [object Object] Version Number: 1.

[26] P.-C. Lo, Y.-H. Tsai, E.-P. Lim, and S.-Y. Hwang. On Exploring the Reasoning Capability of Large Language Models with Knowledge Graphs, Dec. 2023. URL http://arxiv.org/abs/2312.00353. arXiv:2312.00353 [cs].

[27] M. Masoud, B. Pereira, J. McCrae, and P. Buitelaar. Automatic Construction of Knowledge Graphs from Text and Structured Data: A Preliminary Literature Review. pages 9 pages, 585025 bytes, 2021. ISSN 2190-6807. doi: 10.4230/OASICS.LDK.2021.19. URL https://drops.dagstuhl.de/entities/document/10.4230/OASIcs.LDK.2021.19. Artwork Size: 9 pages, 585025 bytes ISBN: 9783959771993 Medium: application/pdf Publisher: [object Object].

[28] I. Melnyk, P. Dognin, and P. Das. Knowledge Graph Generation From Text, Nov. 2022. URL http://arxiv.org/abs/2211.10511. arXiv:2211.10511 [cs].

[29] N. Mihindukulasooriya, S. Tiwari, C. F. Enguix, and K. Lata. Text2KGBench: A Benchmark for Ontology-Driven Knowledge Graph Generation from Text, Aug. 2023. URL http://arxiv.org/abs/2308.02357. arXiv:2308.02357 [cs].

[30] S. Neutel. Towards Automatic Ontology Alignment using BERT. 2021.

[31] OpenAI. Gpt-4 turbo and gpt-4, 2024. URL https://platform.openai.com/docs/models/gpt-4#gpt-4-turbo-and-gpt-4. Last Accessed: 2024-12-15.

[32] OpenAI. Gpt-4o mini: Advancing cost-efficient intelligence, 2024. URL https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/. Accessed: 2024-12-15.

[33] OpenAI. Hello gpt-4o, 2024. URL https://openai.com/index/hello-gpt-4o/. Accessed: 2024-12-15.

[34] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe. Training language models to follow instructions with human feedback. 2022.

[35] J. Z. Pan, S. Razniewski, J.-C. Kalo, S. Singhania, J. Chen, S. Dietze, H. Jabeen, J. Omeliyanenko, W. Zhang, M. Lissandrini, R. Biswas, G. de Melo, A. Bonifati, E. Vakaj, M. Dragoni, and D. Graux. Large Language Models and Knowledge Graphs: Opportunities and Challenges, Aug. 2023. URL http://arxiv.org/abs/2308.06374. arXiv:2308.06374 [cs].

[36] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, and X. Wu. Unifying Large Language Models and Knowledge Graphs: A Roadmap. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–20, 2024. ISSN 1041-4347, 1558-2191, 2326-3865. doi: 10.1109/TKDE.2024.3352100. URL http://arxiv.org/abs/2306.08302. arXiv:2306.08302 [cs].

[37] H. Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semant. web*, 8(3):489–508, Jan. 2017. ISSN 1570-0844. doi: 10.3233/SW-160218. URL https://doi.org/10.3233/SW-160218.

[38] H. Paulheim and C. Bizer. Improving the quality of linked data using statistical distributions. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 10(2):63–86, 2014.

[39] F. Petroni, T. Rocktäschel, S. Riedel, P. Lewis, A. Bakhtin, Y. Wu, and A. Miller. Language Models as Knowledge Bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2463–2473, Hong Kong, China, 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1250. URL https://www.aclweb.org/anthology/D19-1250.

[40] S. Riedel, L. Yao, and A. McCallum. Modeling relations and their mentions without labeled text. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2010, Barcelona, Spain, September 20-24, 2010, Proceedings, Part III 21*, pages 148–163. Springer, 2010.

[41] A. Roberts, C. Raffel, and N. Shazeer. How Much Knowledge Can You Pack Into the Parameters of a Language Model?, Oct. 2020. URL http://arxiv.org/abs/2002.08910. arXiv:2002.08910 [cs, stat].

[42] S. Seo, H. Cheon, H. Kim, and D. Hyun. Structural Quality Metrics to Evaluate Knowledge Graphs, Dec. 2022. URL http://arxiv.org/abs/2211.10011. arXiv:2211.10011 [cs].

[43] A. Singhal. Introducing the knowledge graph: things, not strings. *Official google blog*, 5(16):3, 2012. URL https://blog.google/products/search/introducing-knowledge-graph-things-not/.

[44] B. Stvilia, L. Gasser, M. B. Twidale, and L. C. Smith. A framework for information quality assessment. *J. Am. Soc. Inf. Sci. Technol.*, 58(12):1720–1733, Oct. 2007. ISSN 1532-2882.

[45] D. Vrandečić and M. Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, Sept. 2014. ISSN 0001-0782, 1557-7317. doi: 10.1145/2629489. URL https://dl.acm.org/doi/10.1145/2629489.

[46] Q. Wang, Z. Mao, B. Wang, and L. Guo. Knowledge Graph Embedding: A Survey of Approaches and Applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, Dec. 2017. ISSN 1041-4347. doi: 10.1109/TKDE.2017.2754499. URL http://ieeexplore.ieee.org/document/8047276/.

[47] S. Wang, Z. Wei, J. Xu, T. Li, and Z. Fan. Unifying Structure Reasoning and Language Model Pre-training for Complex Reasoning, July 2023. URL http://arxiv.org/abs/2301.08913. arXiv:2301.08913 [cs].

[48] X. Wang, L. Chen, T. Ban, M. Usman, Y. Guan, S. Liu, T. Wu, and H. Chen. Knowledge graph quality control: A survey. *Fundamental Research*, 1(5):607–626, Sept. 2021. ISSN 26673258. doi: 10.1016/j.fmre.2021.09.003. URL https://linkinghub.elsevier.com/retrieve/pii/S2667325821001655.

[49] J. Webber. A programmatic introduction to Neo4j. In *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*, pages 217–218, Tucson Arizona USA, Oct. 2012. ACM. ISBN 978-1-4503-1563-0. doi: 10.1145/2384716.2384777. URL https://dl.acm.org/doi/10.1145/2384716.2384777.

[50] B. Xue and L. Zou. Knowledge Graph Quality Management: a Comprehensive Survey. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2022. ISSN 1041-4347, 1558-2191, 2326-3865. doi: 10.1109/TKDE.2022.3150080. URL https://ieeexplore.ieee.org/document/9709663/.

[51] L. Yang, H. Chen, Z. Li, X. Ding, and X. Wu. Give Us the Facts: Enhancing Large Language Models with Knowledge Graphs for Fact-aware Language Modeling, Jan. 2024. URL http://arxiv.org/abs/2306.11489. arXiv:2306.11489 [cs].

[52] H. Ye, N. Zhang, H. Chen, and H. Chen. Generative Knowledge Graph Construction: A Review, Sept. 2023. URL http://arxiv.org/abs/2210.12714. arXiv:2210.12714 [cs].

[53] H. Yu, H. Li, D. Mao, and Q. Cai. A domain knowledge graph construction method based on Wikipedia. *Journal of Information Science*, 47(6):783–793, Dec. 2021. ISSN 0165-5515, 1741-6485. doi: 10.1177/0165551520932510. URL http://journals.sagepub.com/doi/10.1177/0165551520932510.

[54] A. Zaveri, A. Rula, A. Maurino, R. Pietrobon, J. Lehmann, and S. Auer. Quality assessment for Linked Data: A Survey: A systematic literature review and conceptual framework. *Semantic Web*, 7(1):63–93, Mar. 2015. ISSN 22104968, 15700844. doi: 10.3233/SW-150175. URL https://www.medra.org/servlet/aliasResolver?alias=iospress&doi=10.3233/SW-150175.

[55] A. Zaveri, D. Kontokostas, S. Hellmann, J. Umbrich, M. Färber, F. Bartscherer, C. Menne, A. Rettinger, A. Zaveri, D. Kontokostas, S. Hellmann, and J. Umbrich. Linked data quality of DBpedia, Freebase, OpenCyc, Wikidata, and YAGO. *Semant. web*, 9(1):77–129, Jan. 2018. ISSN 1570-0844. doi: 10.3233/SW-170275. URL https://doi.org/10.3233/SW-170275.

[56] L. Zhong, J. Wu, Q. Li, H. Peng, and X. Wu. A Comprehensive Survey on Automatic Knowledge Graph Construction. *ACM Computing Surveys*, 56(4): 1–62, Apr. 2024. ISSN 0360-0300, 1557-7341. doi: 10.1145/3618295. URL https://dl.acm.org/doi/10.1145/3618295.

[57] Y. Zhu, X. Wang, J. Chen, S. Qiao, Y. Ou, Y. Yao, S. Deng, H. Chen, and N. Zhang. LLMs for Knowledge Graph Construction and Reasoning: Recent Capabilities and Future Opportunities, Feb. 2024. URL http://arxiv.org/abs/2305.13168. arXiv:2305.13168 [cs].

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich diese Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die aus fremden Quellen direkt oder indirekt übernommenen Gedanken als solche kenntlich gemacht habe. Die Arbeit habe ich bisher keinem anderen Prüfungsamt in gleicher oder vergleichbarer Form vor-gelegt. Sie wurde bisher auch nicht veröffentlicht.

Trier, den 28. März 2025

A. Appendix

A.1 Prompts

The following sections show the exact prompts used for the ontology extraction and KG generation task.

A.1.1 Ontology Extraction Prompt

****Objective:**** You are given a list of articles from Wikipedia. Your task is to analyze and understand the articles to create an RDF-based Neo4j ontology structure. This ontology will later be used to generate knowledge graphs. You must extract entity types and their relationships, presenting the output in the defined JSON structure below. Be as general as possible when defining entities—avoid specific names, organizations, and other unique identifiers.

****Input:**** You will receive a list of strings.

****Output:****

```
{
    "entities": [
        "Entity1",
        "Entity2",
        ...
    ],
    "relationships": [
        {
            "type": "RELATIONSHIP_TYPE",
            "description": "short description of the relation",
            "source": "Entity1",
            "target": "Entity2"
        },
        ...
    ]
}
```

****Example Input:****

```
["Guillermo del Toro Gómez (born 9 October 1964) is a Mexican filmmaker,
author, and artist.",
"The 2024 Summer Olympics were an international multi-sport event held from
26 July to 11 August 2024 in France."]
```

****Example Output:****

```
{
    "entities": [
        "Person",
        "Event",
```

```
        "Location",
        "Occupation",
        "Date"
    ],
    "relationships": [
        {
            "type": "HAS_NATIONALITY",
            "description": "links a person to their nationality",
            "source": "Person",
            "target": "Location"
        },
        {
            "type": "BORN_ON",
            "description": "links a person to their date of birth",
            "source": "Person",
            "target": "Date"
        },
        {
            "type": "HAS_OCCUPATION",
            "description": "links a person to their occupation",
            "source": "Person",
            "target": "Occupation"
        },
        {
            "type": "OCCURS_IN",
            "description": "Links an event to its specific time range",
            "source": "Event",
            "target": "TimePeriod"
        },
        {
            "type": "OCCURS_IN",
            "description": "Links an event to its location",
            "source": "Event",
            "target": "Country"
        }
    ]
}
```

****Instructions:****

1. **Understand the given content:** Inspect and analyze the input content.
2. **Extract the entities:** Identify entity types present in the text.
3. **Extract the relationships:** Determine relationships between the extracted entity types.
4. **Format proper output:** Structure the extracted entities and their relationships in the required JSON format.

Please provide the extracted content in the specified JSON structure.

A.1.2 Initial Hierarchy Extraction Prompt

****Objective:****
You are given a list of entities from an ontology. Your task is to create a hierarchy and assign the entities to them. This means that a parent entity (e.g., `Person`) can have child entities (e.g., `Author`, `Actor`, ...). You should create a hierarchical structure that organizes all entities into broader or narrower categories.

For the leftover entities, which could not be assigned, return them separately under the key `"Other"`. Do not repeat any of the entities. Structure the output in the given JSON format.

****Input:****
You will receive a list of strings, which are the entities.

****Output:****
```
{
  "Entity1": {
    "childrenEntities": {
      "SubEntity1": {},
      "SubEntity2": {}
    }
  },
  "Entity2": {
    "childrenEntities": {...}
  },
  "Other": {
    "Entity3": {}
  }
}
```

****Example Input:****
```
["Person", "Book", "BalletDancer", "Company", "Dancer", "Festival",
"Document", "Event", "Painter"]
```

****Example Output:****
```
{
  "Person": {
    "childrenEntities": {
      "Dancer": {
        "childrenEntities": {
          "BalletDancer": {}
        }
      },
      "Painter": {}
    }
  },
  "Event": {
    "childrenEntities": {
      "Festival": {}
    }
  },
  "Document": {
    "childrenEntities": {
      "Book": {}
    }
  },
```

```
  "Other": {
    "Company": {}
  }
}
```

****Instructions:****

1. **Understand the given content:** Carefully read and analyze the provided list of entities.

2. **Identify the hierarchy:** Determine the hierarchical relationships between the entities, identifying which entities are parent entities and which are child entities.

3. **Check entities:** Ensure all entities from the input list can be found in the output hierarchy. However, make sure there are no duplicate entries in the output.

4. **Format the output:** Structure the entities in the specified JSON format, ensuring clarity and consistency in the hierarchy.

Please provide the extracted content in the specified JSON structure.

A.1.3 Hierarchy Extraction Prompt

****Objective:****
You are given a list of entities from an ontology and an existing hierarchy. Your task is to extend the hierarchy with the given entities and create new ones if necessary (don't forget to assign the entities to the hierarchies). This means that a parent entity (e.g., `Person`) can have child entities (e.g., `Author`, `Actor`, ...). You should create a hierarchical structure that organizes all entities into broader or narrower categories.

For the leftover entities, which could not be assigned, return them separately under the key `"Other"`. Do not repeat any of the entities. Structure the output in the given JSON format.

****Input:****
You will receive a list of strings, which are the entities, and an existing hierarchy.

```
hierarchy = {
  "Entity1": {
    "childrenEntities": {
      "SubEntity1": {},
      "SubEntity2": {}
    }
  },
  "Entity2": {
    "childrenEntities": {...}
  },
  "Other": {}
}
```

****Output:****

```
{
  "Entity1": {
    "childrenEntities": {
      "SubEntity1": {},
      "SubEntity2": {}
    }
  },
  "Entity2": {
    "childrenEntities": {...}
  },
  "Entity3": {
    "childrenEntities": {
      "SubEntity1": {},
      "SubEntity2": {}
    }
  },
  "Other": {
    "Entity4": {},
    "Entity5": {}
  }
}
```

****Example Input:****

```
["BalletDancer", "Painter", "ITCompany", "MusicFestival", "Cat", "Building"]
hierarchy = {
  "Person": {
    "childrenEntities": {
```

```
      "Dancer": {}
    }
  },
  "Company": {},
  "Event": {
    "childrenEntities": {
      "Festival": {}
    }
  },
  "Document": {
    "childrenEntities": {
      "Book": {}
    }
  },
  "Other": {
    "House": {}
  }
}
```

****Example Output:****

```
{
  "Person": {
    "childrenEntities": {
      "Dancer": {
        "childrenEntities": {
          "BalletDancer": {}
        }
      },
      "Painter": {}
    }
  },
  "Company": {
    "childrenEntities": {
      "ITCompany": {}
    }
  },
  "Event": {
    "childrenEntities": {
      "Festival": {
        "childrenEntities": {
          "MusicFestival": {}
        }
      }
    }
  },
  "Document": {
    "childrenEntities": {
      "Book": {}
    }
  },
  "Building": {
    "childrenEntities": {
      "House": {}
    }
  },
  "Other": {
    "Cat": {}
  }
}
```

****Instructions:****

1. **Understand the given content:** Carefully read and analyze the provided list of entities and the provided existing hierarchy.

2. **Identify the hierarchy:** Extend the existing hierarchy by determining the hierarchical relationships between the entities, identifying which entities are parent entities and which are child entities.

3. **Check entities:** Ensure all entities from the input list can be found in the output hierarchy. However, make sure there are no duplicate entries in the output.

4. **Format the output:** Structure the entities in the specified JSON format, ensuring clarity and consistency in the hierarchy.

Please provide the extracted content in the specified JSON structure.

A.1.4 Property Extraction Prompt

****Objective:****
You are given a dictionary of entities from an ontology, which represent the hierarchy between the entities. Your task is to add fitting properties to the parent entities and the child entities, which describe the entity (for example, `Person` can have the properties: `name`, `surname`, `gender`, `birthdate`, ...).

The child entities inherit the properties of the parent entity and can also have additional specific properties (for example, the child entity `Author` may have the properties: `genre`, `awards`, `reputation`, ...). Inheritance can only happen from the parent entity to the child entity.

The `"Other"` category does not have properties itself, but each contained entity gets a `"properties"` key.

****Input:****
You will receive a dictionary of entities.

```
{
  "Entity1": {
    "childrenEntities": {
      "SubEntity1": {},
      "SubEntity2": {}
    }
  },
  "Entity2": {
    "childrenEntities": {...}
  },
  "Other": {
    "Entity3": {}
  }
}
```

****Output:****

```
{
  "Entity1": {
    "properties": {"property1": "datatype", "property2": "datatype"},
    "childrenEntities": {
      "SubEntity1": {
        "properties": {
            "property1": "datatype",
            "property2": "datatype",
            "property3": "datatype",
            "property4": "datatype"
        }
      },
      "SubEntity2": {
        "properties": {
            "property1": "datatype",
            "property2": "datatype",
            "property5": "datatype",
            "property6": "datatype"
        }
      }
    }
  },
  "Entity2": {
```

```
    "properties": {
        "property7": "datatype",
        "property8": "datatype",
        "property9": "datatype"
    },
    "childrenEntities": {...}
  },
  "Other": {
    "Entity3": {
      "properties": {
          "property10": "datatype",
          "property11": "datatype",
          "property12": "datatype"
      }
    }
  }
}
```

****Example Input:****

```
{
  "Person": {
    "childrenEntities": {
      "Employee": {},
      "Customer": {},
      "Dancer": {
        "childrenEntities": {
          "BalletDancer": {}
        }
      }
    }
  },
  "Event": {
    "childrenEntities": {
      "Festival": {}
    }
  },
  "Book": {
    "childrenEntities": {
      "Novel": {}
    }
  },
  "Other": {
    "Company": {}
  }
}
```

****Example Output:****

```
{
  "Person": {
    "properties": {"name": "string", "birthDate": "date", "birthPlace": "string"},
    "childrenEntities": {
      "Employee": {
        "properties": {
            "name": "string",
            "birthDate": "date",
            "birthPlace": "string",
            "employeeID": "string",
            "salary": "number",
```

```
            "phoneNumber": "string"
        }
      },
      "Customer": {
        "properties": {
            "name": "string",
            "birthDate": "date",
            "customerID": "string",
            "purchaseHistory": ["string"],
            "phoneNumber": "string"
        }
      },
      "Dancer": {
        "properties": {
            "name": "string",
            "birthDate": "date",
            "birthPlace": "string",
            "experience": "number"
        },
        "childrenEntities": {
          "BalletDancer": {
            "properties": {
                "name": "string",
                "birthDate": "date",
                "experience": "number",
                "balletStyle": "string"
            }
          }
        }
      }
    }
  },
  "Event": {
    "properties": {"name": "string", "date": "date", "location": "string"},
    "childrenEntities": {
      "Festival": {
        "properties": {
            "name": "string",
            "date": "date",
            "location": "string",
            "duration": "number",
            "theme": "string"
        }
      }
    }
  },
  "Book": {
    "properties": {
        "title": "string",
        "author": "string",
        "publicationDate": "date",
        "genre": "string"
    },
    "childrenEntities": {
      "Novel": {
        "properties": {
            "title": "string",
            "author": "string",
```

```
            "publicationDate": "date",
            "genre": "string",
            "subGenre": "string",
            "pageCount": "number"
        }
      }
    }
  },
  "Other": {
    "Company": {
      "properties": {
          "name": "string",
          "phoneNumber": "string",
          "clientList": ["string"]
      }
    }
  }
}
```

****Instructions:****

1. **Understand the given content:** Carefully read and analyze the provided list of entities.

2. **Assign properties:** Assign appropriate properties to each entity. Ensure that child entities inherit properties from their parent entities and add any additional properties specific to them.

3. **Format the output:** Structure the entities and their properties in the specified JSON format, ensuring clarity and consistency in the hierarchy.

Please provide the extracted content in the specified JSON structure.

A.1.5 KG Extraction Prompt

Objective: You are given a list of articles from wikipedia and an ontology in json format. Your task is to analyze the articles and make use of the entities and the relations given in the ontology to create a knowledge graph. It's important that you only consider each article on its own and don't connect knowledge between the articles, i.e. only create relations between nodes from the same article. Use the more specific labels if existing and use the properties of these labels for the entity!
Identify entities, assign them labels, fill the properties according to label and the ontology, and establish relationships. Try to use the ontology as much as possible. Make sure the types of the startNodes and endNodes in the relations are consistent with relations in the ontology and are only numbers referring to the entity. A relationship must have a startNode and endNode, so do not forget to add them. Also check that the start/endNodes in the relations can be found in the "nodes" determined before. Try avoiding recursive statements, where the startNode and the endNode refer to the same entity. If the endNode can be found in the properties of the startNode but not in the predefined nodes, fill the endNode with the property string, e.g. "occupation":"writer" will result in the triple ("example_entity", HAS_OCCUPATION, "writer")

Input: A list of Wikipedia articles and an ontology in JSON format.

Output:

```
{
    "nodes": [
        {
            "id": "unique_id_1",
            "labels": ["EntityType1"],
            "properties": {
                "name": "Entity1",
                "description": "Brief description of Entity1",
                "attribute1": "Value1",
                "attribute2": "Value2",
                ...
            }
        },
        ...
    ],
    "relationships": [
        {
            "type": "RELATIONSHIP_TYPE",
            "startNode": "unique_id_1",
            "endNode": "unique_id_2",
            "properties": {
                "description": "Description of the relationship",
                "additional_property": "value"
            }
        },
        ...
    ]
}
```

Example Input:

```
list_of_articles = ["Guillermo del Toro Gómez (born 9 October 1964) is a Mexican filmmaker,
author, and artist.", "The 2024 Summer Olympics were an international multi-sport event
held from 26 July to 11 August 2024 in Paris, France."]
```

```
ontology_json = {
    "entities": {
        "Person": {
            "properties": {
                "name": "string",
                "birthDate": "date",
                "birthPlace": "string",
                "nationality": "string"
            },
            "childrenEntities": {
                "Occupation": {
                    "properties": {
                        "occupation": "string"
                    },
                    "childrenEntities": {
                        "Author": {
                            "properties": {
                                "genre": "string",
                                "awards": "[string]",
                                "notableWorks": "[string]"
                            }
                        },
                        "Filmmaker": {
                            "properties": {
                                "films": "[string]",
                                "awards": "[string]"
                            }
                        },
                        "Artist": {
                            "properties": {
                                "medium": "string",
                                "artworks": "[string]"
                            }
                        }
                    }
                },
                "Nationality": {
                    "properties": {
                        "nationality": "string"
                    }
                }
            }
        },
        "Event": {
            "properties": {
                "name": "string",
                "date": "date",
                "location": "string"
            },
            "childrenEntities": {
                "SportEvent": {
                    "properties": {
                        "participants": "[string]",
                        "winner": "string"
                    }
                }
            }
        },
```

```
        "Location": {
            "properties": {
                "name": "string",
                "coordinates": "string",
                "country": "string"
            },
            "childrenEntities": {
                "City": {
                    "properties": {
                        "population": "number",
                        "area": "number",
                        "mayor": "string"
                    }
                }
            }
        },
        "Other": {
            "Date": {
                "properties": {
                    "year": "integer",
                    "month": "integer",
                    "day": "integer"
                }
            }
        }
    },
    "relationships": [
        {
            "type": "HAS_NATIONALITY",
            "description": "Links a person to their nationality",
            "source": "Person",
            "target": "Location"
        },
        {
            "type": "BORN_ON",
            "description": "Links a person to their date of birth",
            "source": "Person",
            "target": "Date"
        },
        {
            "type": "HAS_OCCUPATION",
            "description": "Links a person to their occupation",
            "source": "Person",
            "target": "Occupation"
        },
        {
            "type": "OCCURS_IN",
            "description": "Links an event to its location",
            "source": "Event",
            "target": "Country"
        }
    ]
}
```

****Example Output:****

```
{
    "nodes": [
        {
```

```
            "id": "1",
            "labels": ["Filmmaker", "Author", "Artist"],
            "properties": {
                "name": "Guillermo del Toro Gómez",
                "birthDate": "9.10.1964",
                "nationality": "Mexican",
                "occupation": "Filmmaker, Author, Artist"
            }
        },
        {
            "id": "2",
            "labels": ["SportEvent"],
            "properties": {
                "name": "2024 Summer Olympics",
                "date": "26.7.2024-11.08.2024",
                "location": "Paris, France"
            }
        },
        {
            "id": "3",
            "labels": ["Date"],
            "properties": {
                "year": "1964",
                "month": "10",
                "day": "9"
            }
        },
        {
            "id": "4",
            "labels": ["Nationality"],
            "properties": {
                "name": "Mexican"
            }
        },
        {
            "id": "5",
            "labels": ["City"],
            "properties": {
                "name": "Paris",
                "country": "France"
            }
        }
    ],
    "relationships": [
        {
            "type": "BORN_ON",
            "startNode": "1",
            "endNode": "3",
            "properties": {
                "description": "Guillermo del Toro Gómez was born on 9th of October 1964."
            }
        },
        {
            "type": "HAS_NATIONALITY",
            "startNode": "1",
            "endNode": "4",
            "properties": {
                "description": "Guillermo del Toro Gómez is Mexican."
```

```
            }
        },
        {
            "type": "OCCURS_IN",
            "startNode": "2",
            "endNode": "5",
            "properties": {
                "description": "The 2024 Summer Olympics were held in Paris, France."
            }
        }
    ]
}
```

****Instructions:****

1. **Understand the given content:** Analyze the input articles and ontology structure.
2. **Extract entities and relationships:** Identify entities, assign labels, fill properties, and establish relationships while maintaining consistency.
3. **Format the output:** Ensure the knowledge graph follows the specified JSON format with properly structured nodes and relationships.

Please provide the extracted content in the specified JSON structure.

A.2 Cypher queries -Neo4j Upload

A.2.1 GPT-4o KG Upload

Nodes

```
CALL apoc.load.json(
    "https://drive.google.com/uc?export=download&id=14z5bG3R2lfKJMP4-Ctx287Dwerqkk34b")
YIELD value
UNWIND value.nodes AS node
MERGE (n {id: node.id})
SET n += node.properties
WITH n, node
UNWIND node.labels AS label
CALL apoc.create.addLabels([id(n)], [label]) YIELD node AS updatedNode
RETURN updatedNode;
```

Relationships

```
CALL apoc.load.json(
    "https://drive.google.com/uc?export=download&id=14z5bG3R2lfKJMP4-Ctx287Dwerqkk34b")
YIELD value
UNWIND value.relationships AS rel
MATCH (startNode {id: rel.startNode})
MATCH (endNode {id: rel.endNode})
CALL apoc.create.relationship(startNode, rel.type, rel.properties, endNode) YIELD rel AS createdRe
RETURN createdRel;
```

A.2.2 GPT-4 Turbo KG Upload

Nodes

```
CALL apoc.load.json(
    "https://drive.google.com/uc?export=download&id=1hUIcwZhWqy04R3xh0rglUuShL-di2Euh")
YIELD value
UNWIND value.nodes AS node
MERGE (n {id: node.id})
SET n += node.properties
WITH n, node
UNWIND node.labels AS label
CALL apoc.create.addLabels([id(n)], [label]) YIELD node AS updatedNode
RETURN updatedNode;
```

Relationships

```
CALL apoc.load.json(
    "https://drive.google.com/uc?export=download&id=1hUIcwZhWqy04R3xh0rglUuShL-di2Euh")
YIELD value
UNWIND value.relationships AS rel
MATCH (startNode {id: rel.startNode})
MATCH (endNode {id: rel.endNode})
CALL apoc.create.relationship(startNode, rel.type, rel.properties, endNode) YIELD rel AS createdRe
RETURN createdRel;
```

A.2.3 GPT-4o Mini KG Upload

Nodes

```
CALL apoc.load.json(
    "https://drive.google.com/uc?export=download&id=1kA0Ee_DpULvKsJrX_C8YY0ND1fktXrZl")
YIELD value
UNWIND value.nodes AS node
MERGE (n {id: node.id})
SET n += node.properties
WITH n, node
UNWIND node.labels AS label
CALL apoc.create.addLabels([id(n)], [label]) YIELD node AS updatedNode
RETURN updatedNode;
```

Relationships

```
CALL apoc.load.json(
    "https://drive.google.com/uc?export=download&id=1kA0Ee_DpULvKsJrX_C8YY0ND1fktXrZl")
YIELD value
UNWIND value.relationships AS rel
MATCH (startNode {id: rel.startNode})
MATCH (endNode {id: rel.endNode})
CALL apoc.create.relationship(startNode, rel.type, rel.properties, endNode) YIELD rel AS createdRe
RETURN createdRel;
```

A.3 Question-Answering

A.3.1 Cypher Query Modifications

The following tables contain the GPT-4o generated Cypher queries for the questions asked in the final evaluation step contrasted with the modifications to make the query fit the corresponding graph schema.

Question	Cypher generated with GPT-4o	Cypher modified GPT-4o KG
1.	MATCH (p:Person)-[:HAS_NATIONALITY]->(n:Nationality {name: "English"}) RETURN p.name;	MATCH (p)-[:HAS_NATIONALITY]->(n:Nationality {name: "English"}) RETURN p.name;
2.	MATCH (p:Person {name: "Mihail Andricu"})-[:BORN_IN]->(place:Location) OPTIONAL MATCH (p)-[:BORN_ON]->(date:Date) RETURN p.name, date.value AS birthDate, place.name AS birthPlace	MATCH (p{name: "Mihail Andricu"})-[:BORN_IN]->(place), (p)-[:BORN_ON]->(date) RETURN place.name AS birthplace, date.day + "-" + date.month + "-" + date.year AS birthdate;
3.	MATCH (d:Document {title: "Yser Testament"}) RETURN d.description	MATCH (d {title: "The Yser Testament"})-[r]->(object) RETURN d, type(r) AS relationship, labels(object) AS objectLabels, properties(object) AS objectProperties;

Table A.1: Cypher Modifications for the GPT-4o KG.

Question	Cypher generated with GPT-4o	Cypher modified GPT-4 Turbo KG
1.	MATCH (p:Person)-[:HAS_NATIONALITY]->(n:Nationality {name: "English"}) RETURN p.name;	MATCH (p)-[:HAS_NATIONALITY]->(n:Nationality {name: "English"}) RETURN p.name;
2.	MATCH (p:Person {name: "Mihail Andricu"})-[:BORN_IN]->(place:Location) OPTIONAL MATCH (p)-[:BORN_ON]->(date:Date) RETURN p.name, date.value AS birthDate, place.name AS birthPlace	MATCH (p {name: "Mihail Andricu"})-[:BORN_IN]->(place) RETURN p.name, place.name AS birthPlace;
3.	MATCH (d:Document {title: "Yser Testament"}) RETURN d.description	MATCH (d {title: "The Yser Testament"})-[r]->(object) RETURN d, type(r) AS relationship, labels(object) AS objectLabels, properties(object) AS objectProperties;

Table A.2: Cypher Modifications for the GPT-4 Turbo KG.

Question	Cypher generated with GPT-4o	Cypher modified GPT-4o Mini KG
1.	MATCH (p:Person)-[:HAS_NATIONALITY]->(n:Nationality {name: "English"}) RETURN p.name;	MATCH (p)-[:HAS_NATIONALITY]->(n:Nationality {name: "English"}) RETURN p.name;
2.	MATCH (p:Person {name: "Mihail Andricu"})-[:BORN_IN]->(place:Location) OPTIONAL MATCH (p)-[:BORN_ON]->(date:Date) RETURN p.name, date.value AS birthDate, place.name AS birthPlace	MATCH (p {name: "Mihail Andricu"})-[:BORN_ON]->(date) RETURN p.name, date.dateValue AS birthdate;
3.	MATCH (d:Document {title: "Yser Testament"}) RETURN d.description	MATCH (d {title: "The Yser Testament"})-[r]->(object) RETURN d, type(r) AS relationship, labels(object) AS objectLabels, properties(object) AS objectProperties;

Table A.3: Cypher Modifications for the GPT-4o Mini KG.

Question	Cypher generated with GPT-4o	Cypher modified LangChain KG
1.	MATCH (p:Person)-[:HAS_NATIONALITY]->(n:Nationality {name: "English"}) RETURN p.name;	MATCH (p)-[:HAS_NATIONALITY]->(n:Nationality {id: "English"}) RETURN p.id;
2.	MATCH (p:Person {name: "Mihail Andricu"})-[:BORN_IN]->(place:Location) OPTIONAL MATCH (p)-[:BORN_ON]->(date:Date) RETURN p.name, date.value AS birthDate, place.name AS birthPlace	MATCH (p {id: "Mihail Andricu"})-[:BORN_IN]->(place) RETURN p.id, place.id AS birthPlace;
3.	MATCH (d:Document {title: "Yser Testament"}) RETURN d.description	MATCH (d {id: "Yser Testament"})-[r]->(object) RETURN d, type(r) AS relationship, labels(object) AS objectLabels, properties(object) AS objectProperties;

Table A.4: Cypher Modifications for the LangChain KG.

Table A.1 shows that little change was necessary for the queries to work with the GPT-4o KG. For question 2, the query had to return "day", "month" and "year" instead of the ""date, since the "date" was not a property in the graph. This also was adapted for the other graphs. Additionally, GPT-4 Turbo represented in Table A.2 did not have a relation "BORN_ON" from the "Person" node to a "Date" node, so only the birthplace was asked and returned. The same applied to the LangChain KG. Conversely, the GPT-4o Mini KG (Table A.3) did not contain a connection to the birthplace so only the date was returned. Most graphs did not add a description property to a "Document" entity. Thus, question 3 returned the node and all relations. This was adapted for all KGs, so comparisons of the answers could be made. A necessary modification for the LangChain graph (Table A.4) was the replacement of the "name" property with the "id" property, as this was the equivalent of "name".

This usage of GPT-4o to create the Cypher queries showed how important the graph schema is

for generating Cypher queries automatically, but it helped to understand the basics of the query language for manual adaption. Since this was only a small use case, manual adaption was no problem. In the future, chunking methods or a bigger context window could help to include the graph schema. Then, question-answering for more questions without manual modification could be possible

A.3.2 Correct Answers

Question	Correct Answer
(1) Which people have an English nationality?	Cliff Richard, Gertrude Macdonald, Norman Kendrew, Frederick Charles Bartholomew, Sir Philip Reginald le Belward Grey Egerton, Thomas Major, John Allison, Norman Savage, Reginald Butterworth, Kasey Remel Palmer, Sir Richard Moon, Sir Richard Stanley Barratt
(2) When and where was Mihail Andricu born?	22 December 1894, Bucharest
(3) What is the Yser Testament?	"Open letter addressed to King Albert I and published on 11 July 1917 during World War I. The letter's author, the philologist, set out a number of grievances relating to the treatment of the Flemish within the Belgian Army fighting on the Yser Front during World War I, especially concerning the perceived inequality of French and Dutch languages. It demanded that new legislation to equalise the status of the two languages be introduced after the war. The letter was the most famous action of the Frontbeweging and is considered an important moment in the history of the Flemish Movement. The letter expressed loyalty to Albert I and demanded autonomy [..] for Flanders within a Belgian framework. It [..] provoked an angry reaction from the High Command [...]. Within German-occupied Belgium, a large faction of the Flemish Movement were collaborating with the German authorities as part of the Flamenpolitik and the letter defended their actions.[...] Armand De Ceuninck was appointed to Minister of War in August to restore discipline."

Table A.5: The questions asked to the graph and the correct answers found in the respective Wikipedia articles

A.3.3 Answers of the KGs

A.3.3.1 Question 1 - Which people have an English Nationality?

GPT-4o


```
[
  {
    "p.name": "Frederick Charles Bartholomew"
  },
  {
    "p.name": "Reginald Edmund Compton Butterworth"
  },
  {
    "p.name": "Sir Philip Reginald le Belward Grey Egerton"
  },
  {
    "p.name": "Sir Richard Stanley Barratt"
  }
]
```


GPT-4 Turbo


```
[
  {
    "p.name": "Norman Kendrew"
  },
  {
    "p.name": "Sir Philip Reginald le Belward Grey Egerton, 14th Baronet JP DL"
  }
]
```


GPT-4o Mini


```
[
  {
    "p.name": "Gertrude Macdonald"
  }
]
```


LangChain


```
[]
```

A.3.3.2 Question 2 - When and where was Mihail Andricu born?

GPT-4o


```
[
  {
    "p.name": "Mihail Andricu",
    "birthPlace": "Bucharest",
    "birthDate": "22-12-1894"
  }
]
```


GPT-4 Turbo


```
[
  {
    "p.name": "Mihail Andricu",
    "birthPlace": "Bucharest"
  }
]
```


GPT-4o Mini


```
[
  {
    "p.name": "Mihail Andricu",
    "birthdate": "22 December 1894"
  }
]
```


LangChain


```
[]
```

A.3.3.3 Question 3 - What is the Yser Testament?

GPT-4o


```
[
  {
    "d": {
      "identity": 5,
      "labels": [
        "Document"
      ],
      "properties": {
        "author": "the philologist",
        "language": "Dutch",
        "id": "7",
        "title": "The Yser Testament",
        "publicationDate": "11.7.1917"
      },
      "elementId": "4:4b6351da-cc90-4cdd-8a33-dfb7da2f9a9d:5"
    },
    "relationship": "ADDRESSED_TO",
    "objectLabels": [
      "Person"
    ],
    "objectProperties": {
      "id": "8",
      "nationality": "Belgian",
      "name": "Albert I",
      "birthDate": "",
      "birthPlace": ""
    }
  },
  {
    "d": {
      "identity": 5,
      "labels": [
        "Document"
      ],
      "properties": {
        "author": "the philologist",
        "language": "Dutch",
        "id": "7",
        "title": "The Yser Testament",
        "publicationDate": "11.7.1917"
      },
      "elementId": "4:4b6351da-cc90-4cdd-8a33-dfb7da2f9a9d:5"
    },
    "relationship": "DENOUNCES",
    "objectLabels": [
      "Organization"
    ],
    "objectProperties": {
      "id": "11",
      "establishedDate": "",
      "location": "Belgium",
      "name": "Frontbeweging"
    }
  },
  {
    "d": {
```

```
      "identity": 5,
      "labels": [
        "Document"
      ],
      "properties": {
        "author": "the philologist",
        "language": "Dutch",
        "id": "7",
        "title": "The Yser Testament",
        "publicationDate": "11.7.1917"
      },
      "elementId": "4:4b6351da-cc90-4cdd-8a33-dfb7da2f9a9d:5"
    },
    "relationship": "OCCURS_ON",
    "objectLabels": [
      "Date"
    ],
    "objectProperties": {
      "id": "13",
      "month": 7,
      "year": 1917,
      "day": 11
    }
  }
]
```


GPT-4 Turbo


```
[
  {
    "d": {
      "identity": 6,
      "labels": [
        "Document"
      ],
      "properties": {
        "author": "unknown",
        "language": "Dutch",
        "id": "8",
        "title": "The Yser Testament",
        "publicationDate": "1917-07-11"
      },
      "elementId": "4:a65d4e5d-f486-434f-9fe9-b046f6c0a089:6"
    },
    "relationship": "AUTHORED",
    "objectLabels": [
      "Person"
    ],
    "objectProperties": {
      "id": "9",
      "name": "King Albert I"
    }
  },
  {
    "d": {
      "identity": 6,
      "labels": [
        "Document"
      ],
```

```
      "properties": {
        "author": "unknown",
        "language": "Dutch",
        "id": "8",
        "title": "The Yser Testament",
        "publicationDate": "1917-07-11"
      },
      "elementId": "4:a65d4e5d-f486-434f-9fe9-b046f6c0a089:6"
    },
    "relationship": "OCCURS_IN",
    "objectLabels": [
      "Event"
    ],
    "objectProperties": {
      "id": "10",
      "location": "",
      "name": "World War I",
      "date": ""
    }
  }
]
```


GPT-4o Mini


```
[
  {
    "d": {
      "identity": 3,
      "labels": [
        "Document"
      ],
      "properties": {
        "date": "11 July 1917",
        "author": "Unknown",
        "description": "An open letter addressed to King Albert I during World War I, expressing g
        "id": "5",
        "title": "The Yser Testament"
      },
      "elementId": "4:3af8b909-7238-4dee-8ecb-0be8a0bc3a88:3"
    },
    "relationship": "ADDRESSED_TO",
    "objectLabels": [
      "Person"
    ],
    "objectProperties": {
      "id": "8",
      "name": "Armand De Ceuninck",
      "gender": "Male",
      "birthDate": "",
      "birthPlace": ""
    }
  },
  {
    "d": {
      "identity": 3,
      "labels": [
        "Document"
      ],
      "properties": {
```

```
        "date": "11 July 1917",
        "author": "Unknown",
        "description": "An open letter addressed to King Albert I during World War I, expressing g
        "id": "5",
        "title": "The Yser Testament"
      },
      "elementId": "4:3af8b909-7238-4dee-8ecb-0be8a0bc3a88:3"
    },
    "relationship": "OCCURS_IN",
    "objectLabels": [
      "Location"
    ],
    "objectProperties": {
      "id": "9",
      "area": "",
      "name": "Yser Front",
      "coordinates": "",
      "population": ""
    }
  }
]
```


LangChain


```
[
  {
    "d": {
      "identity": 8,
      "labels": [
        "Document"
      ],
      "properties": {
        "official_title": "Open Letter to the King of the Belgians Albert I",
        "publication_date": "1917-07-11",
        "id": "Yser Testament"
      },
      "elementId": "4:f4831af7-c34d-48c5-86c7-3bd956907840:8"
    },
    "relationship": "ADDRESSED_TO",
    "objectLabels": [
      "Person"
    ],
    "objectProperties": {
      "id": "King Albert I"
    }
  },
  {
    "d": {
      "identity": 8,
      "labels": [
        "Document"
      ],
      "properties": {
        "official_title": "Open Letter to the King of the Belgians Albert I",
        "publication_date": "1917-07-11",
        "id": "Yser Testament"
      },
      "elementId": "4:f4831af7-c34d-48c5-86c7-3bd956907840:8"
    },
```

```
    "relationship": "ACTION_OF",
    "objectLabels": [
      "Movement"
    ],
    "objectProperties": {
      "id": "Frontbeweging"
    }
  },
  {
    "d": {
      "identity": 8,
      "labels": [
        "Document"
      ],
      "properties": {
        "official_title": "Open Letter to the King of the Belgians Albert I",
        "publication_date": "1917-07-11",
        "id": "Yser Testament"
      },
      "elementId": "4:f4831af7-c34d-48c5-86c7-3bd956907840:8"
    },
    "relationship": "PROVOKED_REACTION_FROM",
    "objectLabels": [
      "Organization"
    ],
    "objectProperties": {
      "id": "High Command"
    }
  },
  {
    "d": {
      "identity": 8,
      "labels": [
        "Document"
      ],
      "properties": {
        "official_title": "Open Letter to the King of the Belgians Albert I",
        "publication_date": "1917-07-11",
        "id": "Yser Testament"
      },
      "elementId": "4:f4831af7-c34d-48c5-86c7-3bd956907840:8"
    },
    "relationship": "RELATED_TO",
    "objectLabels": [
      "Movement"
    ],
    "objectProperties": {
      "id": "Flemish Movement"
    }
  },
  {
    "d": {
      "identity": 8,
      "labels": [
        "Document"
      ],
      "properties": {
        "official_title": "Open Letter to the King of the Belgians Albert I",
```

```
      "publication_date": "1917-07-11",
      "id": "Yser Testament"
    },
    "elementId": "4:f4831af7-c34d-48c5-86c7-3bd956907840:8"
  },
  "relationship": "RELATED_TO",
  "objectLabels": [
    "Organization"
  ],
  "objectProperties": {
    "id": "Belgian Army"
  }
},
{
  "d": {
    "identity": 8,
    "labels": [
      "Document"
    ],
    "properties": {
      "official_title": "Open Letter to the King of the Belgians Albert I",
      "publication_date": "1917-07-11",
      "id": "Yser Testament"
    },
    "elementId": "4:f4831af7-c34d-48c5-86c7-3bd956907840:8"
  },
  "relationship": "RELATED_TO",
  "objectLabels": [
    "Policy"
  ],
  "objectProperties": {
    "id": "Flamenpolitik"
  }
},
{
  "d": {
    "identity": 8,
    "labels": [
      "Document"
    ],
    "properties": {
      "official_title": "Open Letter to the King of the Belgians Albert I",
      "publication_date": "1917-07-11",
      "id": "Yser Testament"
    },
    "elementId": "4:f4831af7-c34d-48c5-86c7-3bd956907840:8"
  },
  "relationship": "DEFENDED_ACTIONS_OF",
  "objectLabels": [
    "Organization"
  ],
  "objectProperties": {
    "id": "German Authorities"
  }
},
{
  "d": {
    "identity": 8,
```

```
      "labels": [
        "Document"
      ],
      "properties": {
        "official_title": "Open Letter to the King of the Belgians Albert I",
        "publication_date": "1917-07-11",
        "id": "Yser Testament"
      },
      "elementId": "4:f4831af7-c34d-48c5-86c7-3bd956907840:8"
    },
    "relationship": "CONTEXT",
    "objectLabels": [
      "Event"
    ],
    "objectProperties": {
      "id": "World War I"
    }
  },
  {
    "d": {
      "identity": 8,
      "labels": [
        "Document"
      ],
      "properties": {
        "official_title": "Open Letter to the King of the Belgians Albert I",
        "publication_date": "1917-07-11",
        "id": "Yser Testament"
      },
      "elementId": "4:f4831af7-c34d-48c5-86c7-3bd956907840:8"
    },
    "relationship": "DEMANDED_AUTONOMY_FOR",
    "objectLabels": [
      "Region"
    ],
    "objectProperties": {
      "id": "Flanders"
    }
  },
  {
    "d": {
      "identity": 8,
      "labels": [
        "Document"
      ],
      "properties": {
        "official_title": "Open Letter to the King of the Belgians Albert I",
        "publication_date": "1917-07-11",
        "id": "Yser Testament"
      },
      "elementId": "4:f4831af7-c34d-48c5-86c7-3bd956907840:8"
    },
    "relationship": "GRIEVANCES_AGAINST",
    "objectLabels": [
      "Organization"
    ],
    "objectProperties": {
      "id": "Belgian Army"
```

```
    }
  },
  {
    "d": {
      "identity": 8,
      "labels": [
        "Document"
      ],
      "properties": {
        "official_title": "Open Letter to the King of the Belgians Albert I",
        "publication_date": "1917-07-11",
        "id": "Yser Testament"
      },
      "elementId": "4:f4831af7-c34d-48c5-86c7-3bd956907840:8"
    },
    "relationship": "DEFENDED",
    "objectLabels": [
      "Policy"
    ],
    "objectProperties": {
      "id": "Flamenpolitik"
    }
  }
]
```