Navigating Ontology Development with Large Language Models

Mohammad Javad Saeedizade(✉) and Eva Blomqvist

Linköping University, Linköping, Sweden
{javad.saeedizade,eva.blomqvist}@liu.se

Abstract. Ontology engineering is a complex and time-consuming task, even with the help of current modelling environments. Often the result is error-prone unless developed by experienced ontology engineers. However, with the emergence of new tools, such as generative AI, inexperienced modellers might receive assistance. This study investigates the capability of Large Language Models (LLMs) to generate OWL ontologies directly from ontological requirements. Specifically, our research question centres on the potential of LLMs in assisting human modellers, by generating OWL modelling suggestions and alternatives. We experiment with several state-of-the-art models. Our methodology incorporates diverse prompting techniques like Chain of Thoughts (CoT), Graph of Thoughts (GoT), and Decomposed Prompting, along with the Zero-shot method. Results show that currently, GPT-4 is the only model capable of providing suggestions of sufficient quality, and we also note the benefits and drawbacks of the prompting techniques. Overall, we conclude that it seems feasible to use advanced LLMs to generate OWL suggestions, which are at least comparable to the quality of human novice modellers. Our research is a pioneering contribution in this area, being the first to systematically study the ability of LLMs to assist ontology engineers.

Keywords: LLM · Ontology · Ontology Engineering

1 Introduction

Ontologies are nowadays used in many applications, spanning almost all subjects and industry domains, e.g., as schemas for Knowledge Graphs (KGs) [17] or as stand-alone models for reasoning. However, despite the long history of ontology engineering, including manual ontology engineering methodologies, modelling environments, as well as attempts to automate the process, e.g., ontology learning, ontology engineering remains a time-consuming and error-prone activity. Additionally, while being a research goal for several decades, there are still no methods or tools that allow domain experts themselves to create high-quality ontologies without the involvement of ontology engineers.

At the same time, Large Language Models (LLMs) are now exhibiting an unprecedented level of natural language understanding and generation, and due to their training data even able to produce formal output such as program code.

A. Meroño Peñuela et al. (Eds.): ESWC 2024, LNCS 14664, pp. 143–161, 2024.
https://doi.org/10.1007/978-3-031-60626-7_8

There are already tools emerging for assisting programmers in formalising their requirements into program code in various languages. Analogously, we would like to investigate what contribution LLMs could make in the area of ontology engineering, i.e., in assisting humans in translating their ontological requirements into a formal ontology language, such as OWL. The objective is to pave the way for tools that can aid domain experts and ontologists, by providing precise suggestions during the ontology creation process. As concluded in [23] LLMs are unlikely to replace neither ontologies nor ontology engineers completely, while exploiting LLMs for more effective and efficient ontology engineering is essential.

In the process of ontology development, ontologists use ontology stories [6,30] and Competency Questions (CQ) [14] to specify the capabilities and content of the ontology to be developed. Ontology stories offer descriptive narratives encapsulating the requirements of a project, while specific capabilities to answer queries can be expressed using CQs. Our investigation primarily centres on the ability of LLMs to leverage these ontology stories, as well as the CQs derived from them, for ontology creation. This sets our work apart from attempts to merely translate some textual content into an ontology, since the input does not explicitly state how to model something, but rather only specifies the desired outcome. Further, due to the W3C standard OWL currently being the predominant ontology representation language, we specifically target the generation of OWL formalisations. OWL has several possible syntaxes, while we have chosen to work with Turtle in this study due to its popularity, i.e., it is likely that LLMs have encountered this syntax frequently in their training data.

Overall, this work addresses the following research questions: *To what extent can LLMs create an ontology that meets the requirements of an ontology story? Which LLMs are suitable for this task, and what prompting techniques are most effective?* The main contributions of the work are: (1) Identifying which of the current state-of-the-art LLMs are capable of producing sufficiently well-formed OWL suggestions, (2) analysing the benefits and drawbacks of a range of prompting techniques, and (3) overall assessing the feasibility of LLMs to automatically produce modelling suggestions. The remainder of the paper is organized as follows: In Sect. 2, we discuss related work. Then, in Sect. 3, we describe the methodology and experiment setup, before presenting the results in Sect. 4, and discussing implications of the results in Sect. 5. Finally, we conclude and discuss future work in Sect. 6. Supplementary material, including stories and CQs, prompts, and the results, can be found on GitHub[1].

2 Related Work

This section presents related work, in terms of ontology engineering methods, and current tool support (Sect. 2.1), automated methods for generating OWL ontologies, i.e., ontology learning, including recent approaches using LLMs (Sect. 2.2), and finally an overview of similar work in software engineering and code generation based on LLMs (Sect. 2.3).

[1] https://github.com/LiUSemWeb/LLMs4OntologyDev-ESWC2024.

2.1 Ontology Engineering

Ontology engineering is traditionally a manual effort. Several methodologies exist, e.g. starting from early methodologies such as METHONTOLOGY [12] and NeOn [30]. More recently, agile methods [6,25,29] have become increasingly popular, reflecting the needs of many real-world settings, and in [27] the LOT methodology is presented as a compilation of experiences of many projects, as well as methodologies and tools. While some methodologies have been proposed in combination with explicit tool support, e.g., NeOn was originally proposed together with the NeOn toolkit [30] and [27] propose a set of possible tools to use, most activities are still entirely manual, such as listing of terms and relations, formalisation into an ontology language etc. Commonly, tool support constitute an ontology engineering environment, such as Protégé[2] or TopBraid Composer[3], together with some visualisation, documentation and publishing tools. Development environments guide the user in formalising the ontology by providing common language constructs as user interface shortcuts, e.g., views, buttons, wizards, etc. However, such tools still do not provide any concrete guidance on *how* to formalise the domain, e.g., in OWL.

The notion of Ontology Design Patterns (ODP) was proposed [7,13], as a way to encode modelling best practices and thus guide the ontology engineer. Additionally, collections of best practices have been published[4]. Still, these tools do not automatically connect to the requirements, e.g., expressed ontology stories [6] or use cases [27] combined with CQs [14], but the ontology engineer must, on their own, match their modelling problem to these best practices, which is not straightforward. Thus, there is no methodology or tool currently that assists the ontology engineer in the task of matching their specific requirements to modelling best practices or suggesting potential solutions in an automated manner.

2.2 Ontology Learning and Generation

Generating ontologies automatically, e.g., from natural language text, has been a topic of research for several decades, commonly denoted *ontology learning*. Initially, the field used classic NLP techniques such as POS-tagging in combination with lexical patterns or frames, to detect occurrences of ontological constructs in text. However, with methods such as [26], treating the task as one of machine translation, deep learning has also made significant contributions to this field. Additionally, early language models have been applied for various subtasks, such as extending ontologies and mapping concepts to top-level ontologies [20].

Even more recently, LLMs have been shown to be effective in the Semantic Web area owing to their capability to capture a vast array of information during their training process. LLMs4OL [4] utilized LLMs for ontology learning, where

[2] https://protege.stanford.edu/.

[3] https://allegrograph.com/topbraid-composer/.

[4] See for instance the vocabularies section of https://www.w3.org/TR/ld-bp/ or the whitepaper at https://www.nist.gov/document/nist-ai-rfi-cubrcinc002pdf for OBO ontologies.

LLMs were employed to extract relations among ontology classes or instances. However, this approach was only able to extract relations among entities and did not facilitate the complete generation of an ontology. This paradigm was then used also for tasks like taxonomy discovery, and extraction of non-taxonomic relations across diverse knowledge domains. In [11] a study is made on the performance of LLMs on similarly specific tasks, e.g., NER and relation extraction, but for the biomedical domain. Other preliminary work [21] has used fine-tuned GPT models to translate restricted natural language sentences into DL axioms. However, such specific statements do not represent realistic ontology requirements or scenarios, as targeted in our work. In addition, Text2KGBench [22] and DiamondKG [1] analyzed the capability of LLMs in generating KGs from text using predefined ontologies. Similarly, [9] extends KGs and ontologies, but starting from an existing KG schema. Further, tools are appearing for supporting the practical integration of OWL with deep learning, and LLMs, e.g., [15], but such tools do not provide any guidance specifically for the ontology generation task.

Furthermore, OLaLa [16] investigated the performance of LLMs on ontology matching and showed promising results on this task. Turning, the task around, another recent work has attempted to retrofit the ontology requirements, i.e. CQs, from existing ontologies, using LLMs [2]. While this work is very interesting, as it explores the connection between CQs and OWL formalisation, and shows promising results of generating CQs even with simple prompts, it is not treating the same kind of ontology generation task as our work. Overall, we conclude that research on using LLMs for ontology learning and KG generation in the Semantic Web indicates a promising direction. These approaches indicate the potential of LLMs also for knowledge engineering, but none of the proposed methods have so far treated the specific task targeted in this paper.

2.3 LLMs for Code Generation

Taking a broader perspective, generative AI has recently greatly influenced the generation of code, including competitions like AlphaCode [19]. GitHub Copilot is an example of AI-powered coding assistance [10], that has been shown to reduce coding time for simple tasks [33]. Code Llama [28], demonstrates the growing diversity in LLM applications for coding. Additionally, models like GPT-3.5 and GPT-4, and Bard contribute to this domain, being capable of also generating code to some extent. These developments indicate a growing interest in leveraging generative AI for automating and enhancing coding, marking a paradigm shift in software development. These approaches have inspired our work, but targeting another kind of formalisation, they are not directly transferable to ontologies.

3 Methodology

Our framework for generating ontologies from ontological requirements relies on two components, an LLM and a prompting technique, which are described

below. In addition, we describe the experiment setup and evaluation measures used to compare the outcomes to a baseline of manually constructed ontologies.

3.1 Large Language Models

In our research, it was crucial to thoroughly examine various LLMs to achieve generalisable results and avoid model-specific biases. Therefore, we use a representative selection of models spanning proprietary and open-source versions: GPT-3.5, GPT-4, and Bard from the closed-source spectrum, and Llama-7B, Llama-13B, Llama2-70B [32], Alpaca [31], Falcon-7B [3], Falcon-7B-Instruct [24], WizardLM [37], and Alpaca-LoRA [32][5] from the open source spectrum. We used the Microsoft Azure API to access the GPT models, for Bard we used the web interface, and open source models were run on T4 GPU and 12 GB RAM. By using this broad selection, we aimed to achieve a holistic understanding of the LLM landscape. However, it is important to note that the purpose is not to compare the effectiveness of the models but merely to determine if any of them can produce sufficiently high-quality OWL models, and the characteristics of such suggestions. Hence, we first performed initial experiments on a broad range of models but then quickly settled for a smaller set in subsequent experiments.

3.2 Prompting Methods

In the field of LLMs, a prompt is an input that guides the model's response generation. A prompting technique entails the strategic formulation of these prompts to maximize the efficacy of LLMs. It involves the deliberate structuring and phrasing of prompts to align with the model's training and capabilities.

In our research, we used multiple prompting techniques to test LLMs. Each prompt in our experiments had four sections, and constitutes a template text with variable sections where requirements are input (making them directly reusable for new tasks, also in other domains). The **header** introduces the task, including emphasising the need for a well-structured ontology that accurately captures the information in a given narrative. The **helper** section, explains foundational ontology concepts such as the correct usage of Turtle syntax for restrictions. It may explain syntax, distinctions of classes, properties, and restrictions. The subsequent **story** segment presents the ontology story, complete with its CQs, i.e., the ontological requirements, acting as the primary source of content for ontology creation. Finally, the **footer** anchors the prompt with cautionary advice, spotlighting common pitfalls and recurrent errors. These general mistakes range from overlooking classes, returning empty output, to erroneous definitions of attributes. The purpose of this layout is to present a clear roadmap for the LLM. In addition, LLMs require a memory to handle a context larger than their allowed context size. An external memory module can store information, such as previously generated code and past communications. Zero-shot is the only

[5] Details related to the versions and settings of these models can be found in our supplementary material.

prompting method that does not use a memory. The following details how these prompt sections are implemented for each prompting technique.

Zero-shot Prompting: This method entails a one-time interaction with the LLM, requiring no iteration or feedback loop, utilizing a prompt composed of the four components outlined in the previous section: the header, helper, story, and footer. Together the sections of the prompt provide all the information needed for the ontology construction, in a single interaction.

Sub-task Decomposed Prompting - Waterfall approach (Waterfall) : This method, adapted from [18], breaks down the ontology creation process into five stages. The first stage involves self-guidance by the LLM, followed by the extraction of classes in stage two. The third stage involves constructing a taxonomy, while the fourth stage focuses on defining object properties. Finally, in stage five, the LLM creates datatype properties. Each stage has a tailored header, helper, and footer to guide the LLM and avoid distractions. A query is dispatched to the memory at each stage to retrieve previously generated OWL code, then appending the new results, finally resulting in a complete ontology.

Sub-task Decomposed Prompting - Competency Question by Competency Question (CQbyCQ): This approach instructs the LLM to address one CQ at a time as a decomposition of the requirements. The header guides the LLM to formulate the ontology for the specific CQ, and the memory integrates the output with the previous ones. The helper, story, and footer sections remain similar to the Zero-shot approach. After modelling the final CQ the result is an integrated ontology for the entire narrative, i.e. by simply merging CQ-specific ontologies. Figure 1 illustrates a typical prompt using this prompting technique.

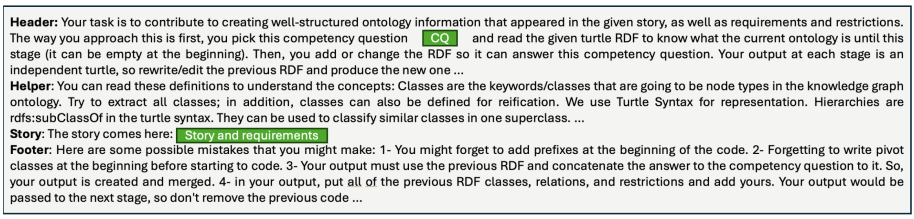

Fig. 1. An excerpt of a representative example of a prompt for CQbyCQ, with the sections of the prompt marked. Grey text is constant for all stories when using the same prompting technique, while green boxes are placeholders for the input story and CQs. The first green box contains the CQ that will be modelled, and the second green box contains the ontology story and its total set of requirements. (Color figure online)

Chain of Thoughts (CoT): In the context of the CoT [36], CoT-SC [34], and GoT [5] , the concept of 'thoughts' is a series of steps that the LLM carefully crafts for itself, where the end result is an ontology. The CoT framework instructs the LLM to create a plan based on the narrative and its associated CQs. The prompt guides the execution of only the next step at a time, requiring OWL code exclusive to that step. The outcome of each step is stored in memory and

added sequentially to the prompt. By following this approach, the ontology is developed progressively by following the generated plan.

Self Consistency with Chain of Thoughts (CoT-SC): CoT-SC [35] is similar to CoT but involves three distinct plans and three outputs. The LLM evaluates and picks the best plan. To ensure diversity of plans, exclusively for this prompting technique, temperature and penalty parameters are adjusted from 0 to 0.5. This allows for exploration of a broader range of potential ontologies.

Graph of Thoughts (GoT): We modified GoT [5] to enhance efficiency and cost-effectiveness. Similar to CoT-SC, the LLM generates three plans and OWL files representing unique ontologies. However, our GoT integrates ideas (instead of picking the best) and solutions from all three into a single ontology, providing a broader perspective for crafting a more effective ontology. This streamlines ontology generation and ensures a more nuanced representation of the story.

3.3 Experimental Setup

In our experiments, we analyse the ontologies produced using a specific combination of prompting techniques and LLMs. Initially we focus on small tasks, created specifically to trigger certain OWL constructs, before then moving on to more realistic tasks, represented as stories and CQs. The resulting ontologies are at each stage manually evaluated against a set of criteria. In the final stage, the same evaluation is also made for a set of student submissions from a master's course, serving as a baseline for comparison.

Initial Experiment. Studying all combinations of LLMs and prompting techniques on realistic tasks, to counteract stochastic variations, is too time-consuming as human experts are required to evaluate the LLM-generated results in detail. We, therefore, first performed a preliminary investigation to assess the performance of each LLM-prompting technique combination using a curated set of small tasks, to be able to already rule out a set of less promising combinations. The small tasks were created as concise narratives intended to produce a very small OWL solution but still test the ability of the LLM to produce models that are not necessarily straightforward and, in some cases, do not conform to common intuition. The preliminary experiment is organized into two phases.

During the first phase (Fig. 2, part 1), ontologies are constructed based on three distinct narratives, 6 prompting techniques, and using 11 different LLMs. Narrative 1 includes the challenge of modelling a restriction, where an instance of a class then violates that defined restriction, and hence a reasoner should detect the inconsistency. Narrative 2 includes the modelling of domain and range restrictions on properties, and when an individual uses this property, the individual should be classified according to the domain or range (even if this is, in common sense, counterintuitive). This design is to make sure the LLMs follow given instructions rather than basing their output on prior (common sense) knowledge. The third narrative simply requires the existence of a particular class definition. The difference is that in the third case, a new class or a set of restrictions must be added without explicit information in the narrative itself.

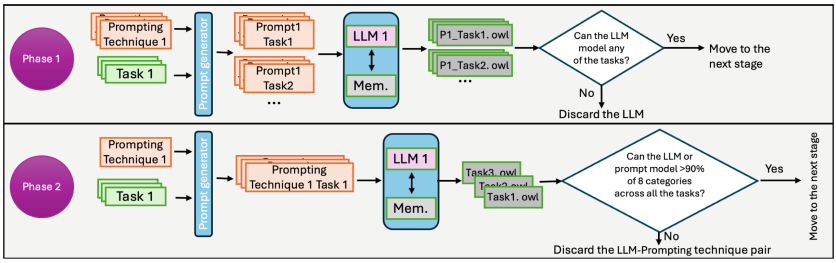

Fig. 2. The two initial experimental phases, aiming to filter out LLMs. Initially, several LLMs are assessed on small tasks. Additional criteria in phase 2 are used to filter out more LLMs, and also prompting techniques.

During this phase, minor syntax errors or other minor discrepancies will not be considered crucial since the primary goal is to pave the way for an interactive tool where such issues could be effortlessly rectified, in some cases even automatically by a syntax validation and correction process. After such syntax correction has been made, the resulting Turtle files are evaluated against one narrative-specific criteria each, similar to the ontology testing methods of error provocation and inference verification [8]. A reasoner is therefore executed over the resulting ontologies from narratives 1 and 2, and the results examined. For narrative 1, an inconsistency should be detected, and for narrative 2 the type of a specific individual is analyzed. For the third narrative, inference verification [8] is used. If the LLM was able to successfully model any one of the narratives (using any prompting techinque) it is retained for the next phase.

In the second phase, ontologies generated in phase 1, that passed the initial evaluation, are further inspected using a set of criteria derived from what content is expected based on the input narrative. We examine the OWL code generated by the models manually and assess it based on simple binary criteria to reduce the subjectivity of evaluators. The following criteria were used in this phase, where all except the last of these categories are expected in the result of each narrative: (1) Presence of an 'EquivalentClass' restriction. (2) Presence of a reification class, i.e. a class not explicitly mentioned in the narrative but needed to model an n-ary relation. (3) Correctness of the Turtle syntax. (4) Presence of domain and range for properties (at least one restriction). (5) Presence of a class hierarchy axiom (rdfs:subClassOf). (6) Semantic coherence of all class hierarchy axioms. (7) Presence of a datatype property. (8) Presence of instances (if specified by the story, one narrative does not require it). Then, by averaging the scores over these binary criteria across three tasks, we derive a score for each LLM-Prompting technique combination (c.f. Fig. 2, phase 2), and LLM-prompting technique combinations with scores over 0.9 are retained.

Main Experiment. The primary objective of our main experiment is to evaluate the quality of OWL files generated by LLMs based on realistic ontological requirements using different prompting techniques. Based on our initial exper-

iment, we have pre-selected a set of LLM-prompting combinations that seem to give reasonable results, and now they are investigated further, i.e. using the experimental setup in Fig. 3. Next, we compare the scores of the LLM-generated ontologies to the same scoring of the first and last student submissions on a course task, using exactly the same ontology stories.

Our evaluation criteria are centred on the extent to which the ontology can address the CQs. We assess this by determining whether a SPARQL query can be written, representing each CQ, for retrieving relevant data as expressed using the ontology, which has been previously suggested as a suitable ontology testing method (c.f. CQ Verification [8]). If the ontology enables successful data retrieval, it passes the test for that CQ. This is then aggregated over the CQs of each story, calculating the proportion of successfully modelled CQs. Two variants of the assessment are used, i.e. either with or without considering minor issues. Minor issues are defined as situations where (1) a single syntactic error is detected, or (2) a single object property or datatype property is missing, preventing the formulation of a correct SPARQL query by one single triple pattern. The rationale behind these being minor issues is that if all other components, e.g., classes, restrictions, and other properties, are present, then adding one single property is an easy task even for a novice modeller.

For scoring the solutions we calculate the following: Let O be the generated model, CQ_i represent i-th CQ, n be the total number of CQs, and $f(O, CQ_i)$ be a function that evaluates the model O with respect to CQ_i and returns a value from {0,1}. The total score is then given by:

$$\text{score} = \frac{\sum_{i=1}^{n} f(O, CQ_i)}{n}$$

Baseline Dataset. The three ontology stories, with associated CQs, used in the main experiment originate from a series of courses, at master's and PhD level as well as conference tutorials, (about 15–20 course instances in total). The tasks were initially developed to allow for experimenting with tool and methodology support for ontology engineering, hence, they are comparable in size (all three stories are accompanied with 14 CQs and one additional statement) and level of difficulty and cover a set of similar modelling problems (set in music, theater, and hospital domains respectively). In particular, all three stories cover a fixed set of modelling challenges, such as creating a coherent taxonomy, reification of relations, use of domain and range, time-dependent relations, and restrictions enabling instance classification, etc. For example, one story set in the theatre domain implies the reification and time indexing of the participation relation when actors are engaged in an ensemble, while in the hospital story, the participation is instead present through employment and membership in a union during a certain time. This makes the tasks also ideal for experimenting with LLMs, since they on one hand cover a broad set of modelling challenges, but also represent three domain-specific variants of the challenges.

In the experiment setup of this paper we specifically used the solutions created by student groups in a master's course in 2009 (10 groups of students in

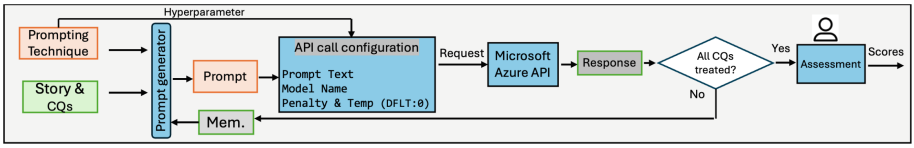

Fig. 3. Architecture of the pipeline for generating and assessing the generated ontologies in the main experiment. The prompting technique determines the hyperparameters of the LLM and creates a prompt text using the memory and a story, to send a request to the MS Azure API. At the end of the pipeline, the human evaluator assesses the ontology and a score for the prompting technique-LLM pair is generated.

total). The students were part of a master's program on information engineering and management, where all had some logic, modelling and programming background but no ontology experience. Before encountering the tasks, the students had a series of lectures on OWL and ontology engineering, and a basic introduction to OWL and the Protégé tool, with some hands-on sessions.

For each assignment (ontology story) students submitted their first attempt, received feedback, and in most cases had to improve their solutions before passing the assignment. Most groups submitted their solutions at least twice, in some cases up to 5 times, to pass. Students worked primarily in pairs (but sometimes 3, or alone). The three stories were each modelled by 10 groups, and picking the first students' submissions and the last ones gives us a dataset of 60 OWL files (i.e. 3 stories, modelled by 10 groups, and 2 solutions per group). The final submissions contained approximately 23 ± 3 classes, 24 ± 5 object properties, and 8 ± 3 datatype properties. We argue that the first submissions would be comparable to what an average junior programmer would come up with after perhaps studying some online tutorials, without task-specific guidance or feedback. While the last student submission represents an acceptable solution to the modelling problem, i.e., sufficient quality with only a few minor issues. This gives us two baselines for comparison with the LLM-generated models, in terms of quality.

4 Experimental Results

For our experiments, we first make a basic assessment of the output quality of OWL ontologies generated based on the three small narratives, allowing us to make an initial selection of LLMs and prompting techniques. We then assess the quality of the LLM output, on the three larger tasks, and compare these results against the ontologies produced by students, using these as a baseline.

4.1 Initial Experiment Results

Phase One: The initial phase of our experiment provided insightful observations regarding the performance of various LLMs in conjunction with different prompting techniques. The results indicate that the open-source and smaller

models (<20B parameters) tested, i.e., LlaMA7B, LlaMA13B, WizardLM, Falcon7B, Alpaca-LoRA, and CodeLlama-13b, lack the capability to provide accurate suggestions. These models consistently failed to produce relevant output to any of the small tasks in this phase. The generated responses were either too brief, ambiguous, only code fragments, or irrelevant text, such as trivial conversations, altogether indicating a lack of ability to perform the specified tasks.

When considering larger models, LlaMA2-70B and Bard exhibited a notable inability to generate outputs in the Turtle syntax. During the code review, we identified numerous issues such as incomplete code, incorrect usage of prefixes, irrelevant modelling for the given story, and numerous syntax errors. In stark contrast, GPT-3.5 and GPT-4 emerged as the only models capable of successfully completing the tasks. They consistently generated outputs that aligned with the expected prompt, demonstrating a considerably higher level of proficiency in ontology generation compared to the other models.

Furthermore, an analysis of the prompting techniques revealed the Sub-task Decomposed Prompting approach, the waterfall method, failed to produce satisfactory results across the three narratives, even when applied with GPT-3.5, and failed in two out of three narratives with GPT-4. This suggests that, despite refinements of the prompts after each failure of the LLM, this technique is not effective for the task at hand. Additionally, the CQbyCQ technique proved too complex for GPT-3.5, resulting in the LLM producing irrelevant output, or too brief code fragments, leading to failures across all small tasks.

In summary, the results from phase one of the preliminary experiment underscore the significant variation in performance across different LLMs and prompting techniques. Only GPT-3.5 and GPT-4 provided sufficiently high-quality results to proceed to the next phase.

Phase Two: For the retained models, the outputs were then evaluated based on the established binary criteria (see Sect. 3.3, phase two). In aspects such as the implementation of an 'owl:EquivalentClass'-restriction, presence of a reification class, and the correctness of the Turtle syntax, GPT-3.5 exhibited considerable shortcomings in comparison to GPT-4. For example, across all prompting techniques, GPT-3.5 missed almost half of the necessary reifications while GPT-4 correctly modelled 95% of them. Overall, during this phase, GPT-4 outperformed GPT-3.5 by approximately 15% on average for all criteria and prompting techniques. Conversely, in criteria involving the expression of domain and range for properties, the creation of a taxonomy using 'rdfs:subClassOf', semantic coherence in class hierarchies, the definition of datatype properties, and the provision of instances, GPT-4 and GPT-3.5 showed similar levels of competence. This suggests that while GPT-3.5 is capable in certain aspects of ontology modelling, it falls short in more complex and nuanced tasks. Regarding, prompting techniques, the waterfall and Zero-shot techniques performed poorly on GPT3.5 and GPT4, with scores below the threshold (0.9), and were therefore excluded.

Considering the overall performance in phase two, it was decided to proceed exclusively with GPT-4 as LLM and CQbyCQ, CoT, CoT-SC, and GoT prompts

in the subsequent phases of our research. Again, it is worth noting that the purpose of the study is to explore the feasibility of LLMs to assist in OWL modelling, not to compare and quantify the capabilities of different models.

4.2 Main Experiment Results

Our experiment aimed to evaluate GPT-4 with remaining prompting techniques. We ran the model three times for each technique and story, analysed all three models with respects to addressed CQs (c.f. criteria in Sect. 3.3). The analysis involved manually writing SPARQL queries that answer each CQ[6]. An example SPARQL query for a reified relation in the hospital story is shown in Listing 1.1. Scores were averaged over the results of the runs, thus reducing the effects of random variations. Then, we compared these averaged outputs from our framework against the average of the 10 student groups' submissions.

Listing 1.1. Example of typical test case in the form of a SPARQL query.

```
# CQ: What role does a certain person have within a certain union
# group at a certain point in time?
SELECT ?role ?person ?union
WHERE {
  ?membership rdf:type :UnionMembership . #Reified relation
  ?membership :memberOf ?union .
  ?membership :member ?person .
  ?membership :role ?role .
  ?membership :startTime ?start .
  ?membership :endTime ?end .
  FILTER (?start > "..."^^xsd:dateTime && ?end < "..."^^xsd:dateTime) }
```

In Fig. 4, each section depicts one ontology story and compares the performance of students with the LLMs outputs. This comparison focuses on to what extent the CQs have been addressed and presents scores as the proportion of the CQs that passed the same test. However, we also provide a modified view that ignores minor errors (annotated with 'IG'), as defined in Sect. 3.3. The latter illustrates how overlooking minor mistakes, e.g., those which even a novice ontology engineer can spot and fix, affects the overall effectiveness of the LLMs.

An observation from Fig. 4 is that by ignoring minor errors, CQbyCQ outperforms all other prompting methods and students' first submissions, while the students' last submissions typically yielded the best output. When comparing students' first and last submissions across all CQs, there was an average improvement of approximately 20%. This improvement showcases the learning process of students during multiple resubmissions and feedback. However, the CQbyCQ scores were closer in performance to the students' last submissions than their first submissions. This implies that this technique is particularly effective, aligning closely with the understanding students develop over time.

By disregarding minor issues, as shown in Fig. 4, there is a considerable increase in the quality of the LLM-generated OWL files. This improvement sug-

[6] The test is passed if a query can be formulated, i.e., no test data is used, and the complexity of the queries has not been analysed so far.

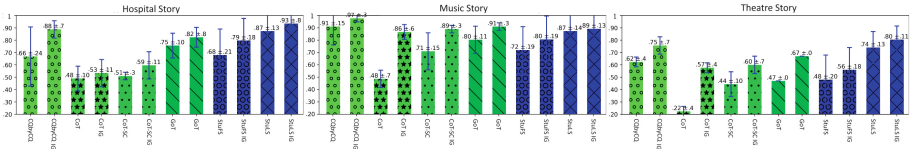

Fig. 4. Comparison of average scores (STD and AVG) of GPT-4, combined with the selected prompting techniques, against students' submissions, expressed as proportion of CQs they sufficiently modelled. 'StuFS' represents students' first submissions, and 'StuLS' students' last submissions. 'IG' indicates results when minor issues are ignored.

gests that many errors in the LLM's outputs are minor. The fact that overlooking these minor issues leads to a substantial boost in performance highlights the potential of LLMs in ontology generation. It underscores that with minimal human intervention for error correction, regarding minor issues in the suggested model, LLMs can be highly effective in assisting the modelling process.

To gain a deeper understanding of the types of CQs that posed challenges for both GPT-4 and students in their modelling efforts, we categorized the CQs into four distinct types, based on the intended modelling solution:

1. Simple Datatype Property: Includes CQs that can be addressed by adding one or more datatype properties, e.g., CQs related to the date of an event.
2. Simple Object Property: Involves defining a connection between two classes by creating an object property, e.g., providing the author of a book.
3. Reification: CQs that require the creation of an abstract class to connect other classes, often in complex situations, e.g., the role of a person in an event at a specific time is a question of reifying the 'role-playing situation'.
4. Restrictions: This involves imposing restrictions on classes or properties, e.g., at least one article is always presented at each seminar.

In Table 1 we present the performance of GPT-4 using the previous prompting techniques, as well as student submission, categorized by the four types of CQs (columns) presented above, where minor issues are again disregarded. Each row corresponds to an average score of different OWL outputs by models or students. The key findings suggest the performance of the CQbyCQ technique stands out in its ability to model simple object properties, datatype properties and apply reification. It consistently performs at a level that is better or comparable to other techniques. However, its effectiveness in creating restrictions is less reliable.

The techniques CoT, GoT, and CoT-SC established restrictions more effectively than other techniques, where CoT consistently succeeded in modelling all restrictions. However, considering their overall performance, the picture is more unclear, and additional experiments would be needed to distinguish their benefits and drawbacks in detail.

Table 1. Scores of the solutions on four different categories of CQs: datatype property (DP), object property (OP), reification (Reif) and restrictions (Rest). Minor issues ignored. Rows represent solutions by prompting techniques/students' scores.

	Theatre Story				Music Story				Hospital Story			
	DP	OP	Reif	Rest	DP	OP	Reif	Rest	DP	OP	Reif	Rest
CQbyCQ	**.93**	**.78**	.55	.33	**1.0**	**.94**	**1.0**	**1.0**	**1.0**	.94	**1.0**	0
CoT	.80	.50	.22	**1.0**	.80	.78	**1.0**	**1.0**	.53	.50	.44	**1.0**
CoT-SC	**.93**	.55	0	**1.0**	.80	.89	**1.0**	**1.0**	.53	.66	.66	.33
GoT	.86	.61	.44	**1.0**	.80	**.94**	**1.0**	**1.0**	.73	**1.0**	.66	.66
StuFS	.64	.63	.43	.10	.84	.83	.87	.30	.78	.85	.73	.60
StuLS	.92	**.78**	**.73**	.50	.94	.92	.90	.50	.94	**1.0**	.90	.60

5 Discussion

According to the findings, the utilisation of LLMs yields promising results in the development of ontology engineering support. However, the generalisability and verifiability of our results can be questioned due to the closed-source nature of the LLMs. For instance, not all hyperparameters of the model are accessible, and model updates may not necessarily improve performance. Even though the previous GPT models are still accessible through MS Azure, hence at the moment the experiments can be repeated, there is no future guarantee. Therefore, newer versions need to be evaluated before being used, especially when it comes to the performance of prompting techniques. In addition, future versions of open-source models will hopefully reach similar performance as seen by GPT-4 in this study. Another potential future direction would be to fine-tune the models to produce OWL output and treat ontology stories and CQs. However, generating broad training data for such fine-tuning might be a challenge.

Regarding validity, there is another potential concern, namely the leakage problem, i.e. the ontology stories used in the experiments already being seen in the training data of LLMs. The narratives for the initial experiment were written for sake of the experiment, but are short enough that it cannot be guaranteed they are not similar to modelling problems already available online. Ontology stories of the main experiment were publicly available, e.g., on course websites, but to the best of our knowledge, no modelling solutions have been publicly available, which significantly reduces the risk of potential leakage. However, further experimentation is necessary to confirm the main experiment results to mitigate this risk, e.g., by applying the method to novel use cases in new domains.

A potential drawback to this effort concerns expenses. Our experiments used realistic, but still small-scale input, and should the length of the stories increase GPT-4 would become a pricey tool to use. Moreover, fine-tuning GPT-4 could be even more expensive. Despite its extensive context size, GPT-4 is also still prone to forgetting certain aspects of the context. This poses the risk of inadequate modelling for large stories at a high cost.

Nevertheless, we still believe that ontology engineering will soon undergo a similar transformation as we have seen for programming. Where on the one hand, ontology engineering will be made accessible to much broader groups, and on the other hand, modelling will be done at an unprecedented speed, also by experienced ontology engineers, due to being able to automate simpler tasks in the modelling process. However, the risks involved are similar to those of using generative black-box methods in general, since the output may comprise an unknown amount of bias, creating unexpected and unwanted side effects when applying the ontologies in downstream tasks. Also, the performance may vary significantly depending on the domain. The tasks used for our experiments were set in common domains, such as music and theatre, while more specialised domains might pose more complex challenges to an LLM modelling assistant. Additionally, the generalisability of prompt structures needs further research. While our prompts can certainly be reused for new tasks and domains, this study is too small to ensure generalisability of the conclusions on prompt structure.

6 Conclusions and Future Work

In our study, we explored a set of prompting techniques, and assessed their effectiveness for generating OWL ontologies, across a range of LLMs. By comparing the ontologies generated by LLMs with student solutions, using consistent evaluation measures, we established a baseline for performance assessment. We conclude that, at the moment, only GPT-3.5 and GPT 4 produce reasonable OWL output in the first place. Further experimentation with GTP-4 revealed that, when used with the CQbyCQ prompting technique, GPT-4 outperforms the average quality of the initial submissions of students (novice ontology engineers) and had a performance similar to their final submissions after multiple feedback rounds, when ignoring minor errors. While for creating more complex OWL constructs, e.g., restrictions, other prompting techniques yield better results. This allows us to conclude that GPT-4, and a combination of prompting techniques, is currently most likely the best avenue ahead for creating an OWL modelling assistant using out-of-the-box models.

Looking ahead, our next objective is therefore to develop a Protégé plugin using such combinations and evaluate its efficacy in reducing the time taken for ontology development by expert ontologists, as well as the level of support that can be provided for novice modellers. Another avenue of future work would be to investigate fine-tuning of models for ontology engineering tasks, and to assess the performance of upcoming releases of open-source models.

Acknowledgement. This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement no. 101058682 (Onto-DESIDE), and is supported by the strategic research area Security Link. The student solutions used in the research were collected as part of a master's course taught by Assoc. Prof. Blomqvist while employed at Jönköping University.

ChatGPT was used to enhance the readability of some of the text and improve the language of this paper, after the content was first added manually. All material was then checked manually before submission.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

Appendix

Motivations, Limitations and Negative Results

Prompt Components: As mentioned in the methodology section 3, there are four sections in each prompt. At a quick glance, the header and story sections appear to be necessary since we provide a brief prompt and the story requirements. The helper and footer sections may be considered optional. However, removing the helper section causes the LLM to completely avoid modelling reifications and misplacing properties, such as putting a datatype property as a range for an object property. The helper begins by outlining strategies to establish a taxonomy, which is otherwise often ignored by LLMs.

The footer, or pitfall section, also enhances the output significantly. It offers the LLMs with common mistakes that they produce. Common errors that are mentioned as pitfalls to avoid are: (1) Providing an empty output of the given prompt. (2) Avoid the use of the Turtle syntax and instead provide a list of items in Python syntax. (3) Avoiding to provide an OWL output without establishing any taxonomy of classes. (4) In the thoughts prompting techniques, avoiding to run the complete plan (several steps) at a current step, since LLMs can ignore instructions and give the complete answer at the first step. (5) Providing explanations instead of providing the code.

Ontology Design Patterns (ODPs) serve as guides for ontology engineer to model an ontology. However, adding examples to prompts seems to degrade output performance. Despite fitting the prompt and story, 32K context LLMs tend to forget the ontology story (we tried with the 128K context GPT4-turbo model and it failed). This could be because the large context is distracting the current LLMs (this could be caused by the low performance of attention layers in LLMs). We used the term "distraction" since the model starts modelling the ODPs in the output instead of the given task.

Limitations: This study, while insightful, has several limitations. Our choice of evaluation method was additionally influenced by the time constraints faced by human experts in manually evaluating the outputs. While this approach was necessary given the available resources, it may not capture the full depth and nuances of LLM-generated ontologies compared to a more thorough, even though time-consuming, manual evaluation.

Due to their extensive branching, the tree of thoughts and the full version of the graph of thoughts techniques proved expensive. This complexity led to slower processing times and increased costs, limiting their practicality for larger-scale or time-sensitive applications.

We used the Microsoft Azure API to access GPT-3.5 and GPT-4, versions 613 trained until 2021. Consequently, our analysis did not consider any advancements or updates in these models post-2021, including the introduction of seed features in newer updates. This might limit the relevance of our findings in the context of

the latest LLM capabilities. The accessibility of hyperparameters in GPT-4 and GPT-3.5 is limited, which presented challenges in our experiment. Despite setting the temperature and penalty parameters to zero (except in plan generation for GoT and CoT-SC, where they were set to 0.5), we observed inconsistencies in the outcomes when using identical prompts. This variability underscores the significance of utilizing open-source LLMs for achieving more consistent and reliable LLM performance rather than depending on unpredictable factors.

We faced another setback in our attempt to produce a more efficient OWL code to reduce context size or general improvement of modelling. For example, in CQbyCQ, when a CQ is addressed, we simply merge it with the previous CQs instead of asking LLM to merge if this CQ has not been addressed. This choice was made since LLMs often forgot to merge classes (or properties) from the previous section, which resulted in incomplete modelling.

Lastly, we encountered another challenge by experimenting with few-shot prompting techniques. In few-shot prompting, a few examples are provided to LLMs as an example. We faced difficulty finding examples of ontology modelling that were not too similar to the ontology story, as this could potentially provide an answer to the LLM. However, this challenge may lead to a similar experiment as the one we mentioned earlier in the usage of ODPs (LLM distraction due to large context size).

Initial Experiment Result Details

Due to space limitations we were not able to present all details of the initial experiment in the main paper body, merely a conclusion summary. The detailed results of the initial experiment, phase 2, are instead reflected here. In Table 2, the LLM-Prompting scores are presented, averaged over the three tasks and 8 criteria, and a threshold of 0.9 is chosen to pass.

Table 2. After conducting the initial experiment phase two, it was decided that CoT, CoT-SC, CQbyCQ, and GoT would move to the next stage (score > 0.9). GPT-3.5 was excluded as its performance was found to be equal to or less than GPT-4.

Prompting Technique:	Zero-shot	Waterfall	CoT	CoT-SC	CQbyCQ	GoT
Score using GPT-3.5	0.77	0.86	**0.91**	0.6	0.77	0.73
Score using GPT-4	0.86	0.86	**0.91**	**0.96**	**1**	**0.92**

References

1. Alharbi, R., et al.: Exploring the role of generative AI in constructing knowledge graphs for drug indications with medical context. In: 15th International Semantic Web Applications and Tools for Healthcare and Life Sciences (SWAT4HCLS 2024) (2024). (to appear)
2. Alharbi, R., Tamma, V., Grasso, F., Payne, T.: An experiment in retrofitting competency questions for existing ontologies. arXiv preprint arXiv:2311.05662 (2023)

3. Almazrouei, E., et al.: Falcon-40B: an open large language model with state-of-the-art performance (2023). https://huggingface.co/tiiuae/falcon-40b
4. Babaei Giglou, H., D'Souza, J., Auer, S.: Llms4ol: large language models for ontology learning. In: Payne, T.R., et al. (eds.) ISWC 2023. LNCS, pp. 408–427. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-47240-4_22
5. Besta, M.: Graph of thoughts: solving elaborate problems with large language models. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 38, no. 16, pp. 17682–17690 (2024)
6. Blomqvist, E., Hammar, K., Presutti, V.: Engineering ontologies with patterns-the extreme design methodology. In: Ontology Engineering with Ontology Design Patterns. IOS Press (2016)
7. Blomqvist, E., Sandkuhl, K.: Patterns in ontology engineering: classification of ontology patterns. In: ICEIS, vol. 3, pp. 413–416. SciTePress (2005). https://doi.org/10.5220/0002518804130416. ISBN: 972-8865-19-8. INSTICC
8. Blomqvist, E., Seil Sepour, A., Presutti, V.: Ontology testing-methodology and tool. In: ten Teije, A., et al. (eds.) Knowledge Engineering and Knowledge Management. EKAW 2012. LNCS, vol. 7603, pp. 216–226. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33876-2_20
9. Caufield, J.H., et al.: Structured prompt interrogation and recursive extraction of semantics (SPIRES): a method for populating knowledge bases using zero-shot learning. Bioinformatics **40**(3), btae104 (2024). https://doi.org/10.1093/bioinformatics/btae104
10. Chen, M., et al.: Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374 (2021)
11. Chen, Q., et al.: Large language models in biomedical natural language processing: benchmarks, baselines, and recommendations (2024). https://arxiv.org/abs/2305.16326
12. Fernández, M., Gómez-Pérez, A., Juristo, N.: Methontology: from ontological art towards ontological engineering. In: Proceedings of the AAAI97 Spring Symposium Series on Ontological Engineering (1997)
13. Gangemi, A.: Ontology design patterns for semantic web content. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 262–276. Springer, Heidelberg (2005). https://doi.org/10.1007/11574620_21
14. Grüninger, M., Fox, M.S.: The role of competency questions in enterprise engineering. In: Rolstadås, A. (eds.) Benchmarking — Theory and Practice. IFIP Advances in Information and Communication Technology, pp. 22–31. Springer, MA (1995). https://doi.org/10.1007/978-0-387-34847-6_3
15. He, Y., Chen, J., Dong, H., Horrocks, I., Allocca, C., Kim, T., Sapkota, B.: Deeponto: A python package for ontology engineering with deep learning (2024). (To appear in the Semantic Web Journal)
16. Hertling, S., Paulheim, H.: OLaLa: ontology matching with large language models. In: Proceedings of the 12th Knowledge Capture Conference 2023. K-CAP '23, pp. 131–139. Association for Computing Machinery, New York, NY (2023). https://doi.org/10.1145/3587259.3627571
17. Hogan, A., et al.: Knowledge Graphs. Morgan & Claypool Publishers, San Rafael (2021)
18. Khot, T., et al.: Decomposed prompting: a modular approach for solving complex tasks. arXiv preprint arXiv:2210.02406 (2022)
19. Li, Y., et al.: Competition-level code generation with alphacode. Science **378**(6624), 1092–1097 (2022)

20. Lopes, A., Carbonera, J., Schmidt, D., Garcia, L., Rodrigues, F., Abel, M.: Using terms and informal definitions to classify domain entities into top-level ontology concepts: an approach based on language models. Knowl. Based Syst. **265**, 110385 (2023). https://doi.org/10.1016/j.knosys.2023.110385, https://www.sciencedirect.com/science/article/pii/S0950705123001351
21. Mateiu, P., Groza, A.: Ontology engineering with large language models (2023). https://arxiv.org/abs/2307.16699
22. Mihindukulasooriya, N., Tiwari, S., Enguix, C.F., Lata, K.: Text2kgbench: a benchmark for ontology-driven knowledge graph generation from text. In: Payne, T.R., et al. (eds.) ISWC 2023. LNCS, vol. 14266, pp. 247–265. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-47243-5_14
23. Neuhaus, F.: Ontologies in the era of large language models-a perspective. Appl. Ontol. **18**(4), 399–407 (2023)
24. Penedo, G., et al.: The refinedweb dataset for falcon LLM: outperforming curated corpora with web data, and web data only. arXiv preprint arXiv:2306.01116 (2023)
25. Peroni, S.: A simplified agile methodology for ontology development. In: Dragoni, M., Poveda-Villalón, M., Jimenez-Ruiz, E. (eds.) OWLED ORE 2016 2016. LNCS, vol. 10161, pp. 55–69. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-54627-8_5
26. Petrucci, G., Rospocher, M., Ghidini, C.: Expressive ontology learning as neural machine translation. J. Web Seman. **52**, 66–82 (2018)
27. Poveda-Villalón, M., Fernández-Izquierdo, A., Fernández-López, M., García-Castro, R.: Lot: an industrial oriented ontology engineering framework. Eng. Appl. Artif. Intell. **111**, 104755 (2022). https://doi.org/10.1016/j.engappai.2022.104755, https://www.sciencedirect.com/science/article/pii/S0952197622000525
28. Roziere, B., et al.: Code llama: open foundation models for code. arXiv preprint arXiv:2308.12950 (2023)
29. Shimizu, C., Hammar, K., Hitzler, P.: Modular ontology modeling. Semant. Web **14**(3), 459–489 (2023)
30. Suárez-Figueroa, M., Gómez-Pérez, A., Motta, E., Gangemi, A. (eds.): Ontology Engineering in a Networked World. Springer, Cham (2012)
31. Taori, R., et al.: Stanford alpaca: an instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca (2023)
32. Touvron, H., et al.: Llama: open and efficient foundation language models. arXiv preprint arXiv:2302.13971 (2023)
33. Vaithilingam, P., Zhang, T., Glassman, E.L.: Expectation vs. experience: evaluating the usability of code generation tools powered by large language models. In: Chi Conference on Human Factors in Computing Systems Extended Abstracts, pp. 1–7 (2022)
34. Wang, L., et al.: Plan-and-solve prompting: improving xero-shot chain-of-thought reasoning by large language models. In: Rogers, A., Boyd-Graber, J., Okazaki, N. (eds.): Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics, vol. 1: Long Papers, pp. 2609–2634. Association for Computational Linguistics, Toronto (2023). https://doi.org/10.18653/v1/2023.acl-long.147
35. Wang, X., et al.: Self-consistency improves chain of thought reasoning in language models. arXiv preprint arXiv:2203.11171 (2022)
36. Wei, J., et al.: Chain-of-thought prompting elicits reasoning in large language models. Adv. Neural. Inf. Process. Syst. **35**, 24824–24837 (2022)
37. Xu, C., et al.: Wizardlm: empowering large language models to follow complex instructions. arXiv preprint arXiv:2304.12244 (2023)