UNIVERSITÀ DEGLI STUDI DI BARI
"ALDO MORO"

Analysis and development of a prompt engineering ontology

DEPARTMENT OF COMPUTER SCIENCE

Master degree in Computer Science

Thesis in

MACHINE LEARNING

Advisor:
Prof. Claudia d'Amato

Graduating:
Simone Gramegna

Co-advisor:
Dott. Roberto Barile
Dott. Andrea Nuzzolese

Academic Year 2023/2024

Abstract

This thesis explores the development of a prompt engineering ontology to improve the understanding, organization, and application of large language models (LLMs) across various domains. By providing a structured representation of prompting techniques and their relationships with LLMs, the ontology serves as a valuable resource for guiding users in effectively interacting with these advanced systems. Large Language Models (LLMs) are AI models based on large neural networks, trained on vast amounts of text to understand and generate natural language. Those models are interesting because they are revolutionizing human-machine communication by automate complex tasks like writing, translation and coding. However, the currently available resources reviewing LLMs and prompt engineering lack and they are fragmented because both technologies are very recent, they are evolving quickly and there is no resource that put them together. We propose PEO, an ontology that models the domain of LLMs and prompt engineering in order to facilitate users in choosing the most appropriate techniques for solving a problem. PEO is designed to support a wide range of users, including researchers, developers, educators, and content creators. The research adopts a systematic approach to ontology development, following the LOT (Linked Open Terms) methodology. We adopted a systematic approach to ontology development, following the Linked Open Terms (LOT) methodology and special attention is given to ontology evaluation and publication. We evaluate the ontology according to established experimental protocols. The outcomes suggest that it is possible to formalize knowledge about large language models and prompt engineering, inferring new useful knowledge to the users. PEO can provide support to a wide range of users, including researchers, developers, educators, and content creators.

This thesis would not have been possible without the support and encouragement of many extraordinary people. I would like to express my deepest gratitude to my supervisor, Professor Claudia d'Amato, for her invaluable guidance and to my co-supervisors: Dr. Roberto Barile and Dr. Andrea Nuzzolese for their insightful contributions and technical support. I am profoundly grateful to my family for their unwavering belief in me, to my friends for their support, and to my partner for her endless love and encouragement.
A special thanks goes to ChatGPT, which has been a great useful tool during these two years.
To all of you, I am eternally grateful.

"Per aspera ad astra"

Contents

List of Figures

List of Tables

Il sentiero per il Paradiso inizia all'Inferno.

Chapter 1

Introduction

1.1 Context

Artificial intelligence (AI) has become an increasingly integral part of our lives, having an undeniable impact on today's society. AI was defined, for the first time, in 1955 at Darthmounth Research project as problem of "making a machine behave in ways that would be called intelligent if a human were so behaving"[45]. It was defined, for the first time, in 1955 at Darthmounth Research project as problem of "making a machine behave in ways that would be called intelligent if a human were so behaving" [45]. The development of AI supported by the increasing availability of computational resources lead to a wide range of applications in multiple domains ranging from industry, healthcare, and business to education [15]. Artificial intelligence is not only applied to these sectors but can also be found in common applications such as social media, digital assistants, recommendations, online searches, and facial recognition [74]. These are just some of the applications we interact with in our daily lives. Voice assistants are an example of an interactive application of artificial intelligence. The first voice assistants began to appear starting in 2011, when Apple introduced Siri on its new iPhone mode and later, other assistants were introduced to the market, including Amazon Alexa (2014) and Google Assistant (2016) [87]. Among the countless applications, also voice assistants appeared, starting with Apple Siri (2011) and followed by Amazon Alexa (2014) and Google Assistant (2016) [87]. Voice assistants rely on natural language processing technologies that have become increasingly complex over the years. They have evolved from rule-based models to more advanced architectures and representations, enabling them to process and generate text with greater accuracy. In fact, such voice assistants have been surpassed by ChatGPT, a chatbot released by OpenAI (November 2022). ChatGPT, in which GPT stands for Generative Pre-trained Transformer a family of large language models created by OpenAI that uses deep learning to generate human-like, conversational text. ChatGPT relies on the Generative Pre-trained Transformer GPT family of Large Language Models (LLMs). LLMs are LMs based on deep learning architectures and trained at a very large scale and they represent a significant leap forward compared to the previous state-of-the-art: a variety of complex tasks can be framed as simple prompting, i.e., creating a simple input text for solving a specific task. This feature is crucial because instead of adapting the training set of the model, it

is possible to adapt the input given to the model. In contrast, LM typically require fine-tuning on domain-specific data or additional training to adapt to new tasks, whereas adapting the input through prompt engineering allows for flexible task customization without modifying the model's parameters. LLMs solve with remarkable performance tasks including coding, mathematical reasoning, and even image generation (when integrated with other architectures e.g., DALL-E 3) [92].

The advent of advanced LLMs creates not only new opportunities for AI, but also introduces significant challenges. One of the main challenges in using these models is creating and optimize prompts (questions posed to the model by the humans) [69] that provide the model with the right instructions to generate accurate and relevant responses. Prompt engineering specifically addresses this challenge and focuses on defining the interactions and outputs of large language models, whose core purpose is to create optimal prompts for a generative model[7]. Prompt engineering is a novel discipline that aims at solving this challenge by providing approaches for designing, optimizing and verifying prompts for LLMs [7]. It is gaining increasing attention, indeed, a new professional role is emerging within industries: the prompt engineers. The prompt engineer must essentially select the most appropriate prompt engineering technique for a given task, a specific large language model, and the intended goal.

1.2 Thesis Objective

With the rise of LLMs and prompt engineering, a plethora of resources emerged on the web. However, as of now, no resource exists that consolidates this knowledge in a clear, simple, and structured manner. Therefore, the objective of this thesis is to create an ontology that logically and systematically describes prompt engineering and LLMs, two concepts that, although apparently different, are inherently interconnected. The use of an ontology allows for effectively expressing the connections between entities in these two domains by employing classes, instances, and relationships. We aim for a well-structured and comprehensive ontology. Moreover, we aim to make it available openly and freely in order to a) be reachable by a wide range of users including students, researchers, and developers; b) foster interoperability and reuse. The reuse concept has been used during the development process by reusing ontology design patterns.

For the purpose, we propose the Prompt Engineering Ontology (PEO): a modern ontology developed using state-of-art methodologies and implemented in OWL: a standardized language based on Description Logics.

Concerning the validity of PEO, we intend to answer the following research questions (RQs):

- RQ 1: Does PEO provide comprehensive and consistent knowledge on LLMs and prompt engineering?
- RQ 2: Can we use PEO to infer additional knowledge?

For answering the RQs, we perform an experimental study grounded on established experimental protocols.

1.3 Thesis Structure

The subsequent chapters of this thesis are structured as follows. Chapter 2 illustrates the foundations of the thesis. Specifically, it reports the main concepts related to ontologies and ontology engineering methodologies and the state-of-the-art in LLMs and prompt engineering that is our target domain.In Chapter 3, "Ontology design", the ontology design phase is described in detail, including an in-depth explanation of the various stages of the design process. Chapter 3 details the design of our proposal, PEO. Chapter 4 details the conceptualization and the implementation of PEO. Chapter 5 details the evaluation process of PEO. Finally, Chapter 6, concludes the thesis by summarizing the work accomplished and discussing potential future developments.

Chapter 2

Background

This chapter provides the basics essential for the thesis. Sect. 2.1 explains ontologies and ontology languages. Sect. 2.2 reviews the state-of-the-art on ontology engineering. The last two sections explain the domain of interest of PEO. Specifically, Sect. 2.3 describes LLMs, while Sect. 2.4 surveys the state-of-the-art in prompt engineering.

2.1 Ontologies

Definition

In AI, an ontology is a formal explicit description of a domain of interest for a certain purpose. The main components of ontologies are classes that represent concepts, and relationships among them, e.g., class hierarchies, class disjointness and class unions. Additionally, each class can be associated with attributes (roles) and restriction rules on the attributes (role restrictions). Just as in object-oriented programming, classes are instantiated to create instances (individuals) [63].

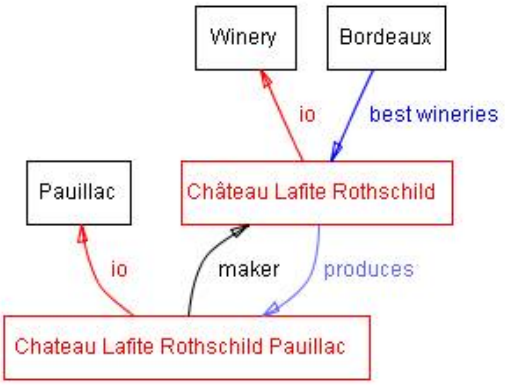

FIGURE 2.1: Example of a simple ontology in wine domain

Fig. 2.1 depicts an example of an ontology in the wine domain; it represents wines, regions, producers.

Classification

In this section, we classify ontologies along multiple dimensions [16]. Firstly, ontologies can be classified according to the level of generality of the domain of discourse; specifically, from the most general to the most specific domains, as follows:

- High-level ontologies describe very general concepts or common-sense knowledge such as space, time, objects, and actions. E.g., DOLCE ontology [13].
- Domain ontologies describe the vocabulary, theories, and fundamental principles that govern a specific domain. E.g., tourism ontology [35].
- Task ontologies: describe the vocabulary related to a specific task or activity, providing a specialization of the terms introduced in the high-level ontology. E.g., planning ontology [56].
- Application ontologies describe concepts related to a specific domain or task and are often derived through a specialization of domain ontologies and task ontologies. E.g., OntoWeb [38].

Ontologies can differ in the level of formalism adopted. Therefore, they can be classified also as follows:

- Highly informal ontologies are expressed in natural language, thus potentially leading to ambiguous definitions given the inherent ambiguity of natural language.
- Semi-informal ontologies are expressed in a rigid and structured form of natural language, improving clarity and reducing ambiguities.
- Semi-formal ontologies are expressed through formally defined artificial languages.
- Strictly formal ontologies are expressed with languages based on formal semantics.

A further classification can be made based on the expressiveness that the ontology aims to convey:

- Controlled vocabularies are finite lists of terms representing the simplest possible notion of an ontology. A typical example is a catalogue that provides only terms with an unambiguous interpretation.
- Glossaries are lists of terms along their meanings expressed through natural language statements. Being primarily created for human use, they often consist of ambiguous statements that cannot be used by automated agents.
- Thesauri add semantics to glossaries by defining the relationships between terms (such as synonymy relations). Typically, they do not provide an explicit hierarchical structure, although this can be inferred from the specification of the terms.

- Informal Is-A hierarchies are ontologies in which generalization and specialization are achieved even though there is no strict "sub-class" hierarchy. They include various ontologies available on the Web.
- Formal Is-A hierarchies are ontologies in which concepts are organized according to a strict subclass hierarchy.
- Frames are ontologies in which concepts are described in terms of their characteristic properties. Inheritance relationships allow properties specified for general concepts to be inherited by more specific ones.
- Logic based ontologies represent knowledge in a formal structured manner, enabling reasoning and inference, e.g., classification and consistency check.

Ontology Languages

Ontology languages are formal languages designed to define and represent ontologies, which describe the structure of knowledge in a specific domain of interest [64]. Knowledge in an ontology is divided into two components:

- TBox (Terminological Box): defines the conceptual structure, including classes and properties, along with axioms that describe their relationships and constraints.
- ABox (Assertional Box): contains assertions about specific instances, describing which individuals belong to particular classes and how they relate to each other.

RDF (Resource Description Framework) is a framework used to describe graph data by using triples. Triples are used to define the ABox of an ontology, each triple has three components:

$$\langle subject, predicate, object \rangle$$

The triple states that there is a relationship of type *predicate* from the entity *subject* to the entity *object*. Each triple component can be identified using a URI (Uniform Resource Identifier). The *object* can be a literal or an URI while *subject* and *predicate* must be a URI. For example, the triple:

```
<http://example.org/person/Alice> <http://example.org/
   knows> <http://example.org/person/Bob>.
```

it states that the resource identified by <http://example.org/person/Alice> knows (<http://example.org/knows>) a resource identified by <http://example.org/person/Bo

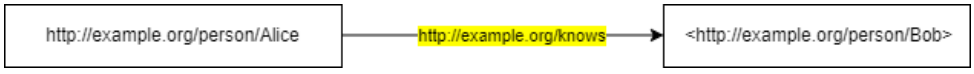

FIGURE 2.2: Relation between two individuals

Fig. 2.2 represents the RDF triple described before.
A more complex example is the following:

```
<http://example.org/person/Alice> <http://example.org/
   name> "Alice".

<http://example.org/person/Bob> <http://example.org/
   name> "Bob".

<http://example.org/person/Matt> <http://example.org/
   name> "Matt".

<http://example.org/person/Alice> <http://example.org/
   knows> <http://example.org/person/Bob>.

<http://example.org/person/Matt> <http://example.org/
   knows> <http://example.org/person/Alice>.
```

This examples involves three individuals: Alice (<http://example.org/person/Alice>), Bob (<http://example.org/person/Bob>) and Matt (<http://example.org/person/Matt>). Each one has a name, associated with the relation <http://example.org/name> connecting the corresponding literal. Alice knows Bob, this is expressed by the triple: <http://example.org/person/Alice> <http://example.org/knows> <http://example.org/person/Bob>. Matt knows Alice, this is expressed by the triple <http://example.org/person/Matt> <http://example.org/knows> <http://example.org/person/Alice>.

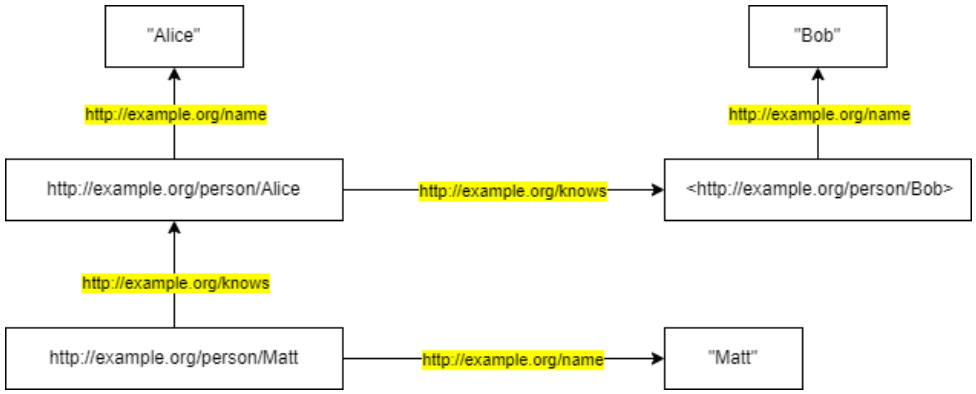

FIGURE 2.3: Relation between individual and name

Fig. 2.3 represents concepts described as an oriented labelled graph edge. OWL is a family of knowledge representation languages for ontologies, the latest version OWL2, born in 2009, is used to build ontologies representing classes, properties, individuals, and data values that are stored as Semantic Web documents [1]. OWL languages are based on DLs: a family of logic formalism derived from first order logics. Main OWL members (profiles) are:

- OWL DL: based on Description Logics ($SHOIN$), it is designed to provide the maximum expressiveness possible.
- OWL Lite: simplified subset of OWL DL, based on $SHIF(D)$, with fewer constructs for simpler applications.

- OWL Full: based on a different semantics from OWL Lite or OWL DL, it is designed to be compatible with RDF.
- OWL 2: based on $SROIQ$: a more expressive extension of previous DLs ($SHOIN$).

In general, all OWL profiles define TBox: terminological component describing the interest domain. Interest domain is modelled through classes and properties.

Name	Syntax	Description
Class	owl:Class	Represents a set of individuals
Object Property	owl:ObjectProperty	Relates individuals to other individuals
Data Property	owl:DatatypeProperty	Relates individuals to data values
Individual	owl:NamedIndividual	Represents an instance of a class

TABLE 2.1: Main OWL2 constructs

Table 2.1 contains the main constructs in OWL2. The main reason for choosing OWL2 is its capacity to support automatic reasoning, inferring new knowledge starting from what is represented in the ontology. Reasoning is important to verify the consistency of the ontology, determine subsumption relations between concepts, check whether an individual belongs to a certain class and retrieve informations. For instance, suppose we have an ontology that defines the concept of "Animal" and its subclasses, such as "Mammal" and "Bird". If we also define that "Cat" is a subclass of "Mammal", and we have an individual "Whiskers" that belongs to the "Cat" class, a reasoner can automatically infer that "Whiskers" is also a "Mammal" and an "Animal", even if these relationships were not explicitly declared.

One important aspect is the adoption of the Open World Assumption (OWA), which implies that the absence of information does not imply its falsehood so if data are not declared we do not have informations to establish its truth. This is useful in the semantic web where the knowledge is distributed across different sources and it is impossible to have access to all possible informations.

SPARQL is a semantic query language that allows to retrieve and manipulate data stored in RDF format. It is based on triple patterns, i.e., statements consisting of a subject, predicate, and object, which resemble RDF triples. These triples are used to define queries that match specific data in RDF. An example of SPARQL query is the following:

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE
  { ?x foaf:name ?name .
```

```
    ?x foaf:mbox ?mbox
}
```

This SPARQL query selects and returns the names (`?name`) and email addresses (`?mbox`) of entities (`?x`) that have a `foaf:name` and `foaf:mbox` property defined in the FOAF vocabulary.

SWRL (Semantic Web Rule Language) is a language for semantic web that can be used to express logic rules, combining OWL DL with a subset of Datalog. Rules are of the form of an implication between an antecedent (body) and a consequent (head). The intended meaning can be read as: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold [86]. An example of SWRL rule is the following:

```
hasParent(?x1,?x2)     hasBrother(?x2,?x3) -> hasUncle(?x1
   ,?x3)
```

This SWRL rule states that if ?$x1$ has a parent ?$x2$, and ?$x2$ has a brother ?$x3$, then ?$x3$ is the uncle of ?$x1$. It infers family relationships in an ontology, allowing automatic reasoning about uncles based on parents and siblings. SWRL extends OWL enabling advanced inferences but a drawback is that it can be computationally expensive and rules are not executable in all reasoners.

Ontology Repositories

Ontology repositories facilitate the discovery of ontologies and thus their reuse, main repositories are:

- W3.org [1]: a curated list of ontologies covering various domains, maintained by W3C.
- DBpedia [2]: a repository of linked data ontologies extracted from Wikipedia, enabling semantic web applications.
- AgroPortal [3]: a platform hosting agricultural and food-related ontologies for data interoperability.
- BioPortal [4]: a repository of biomedical ontologies providing standardized vocabularies for health and life sciences.
- LOV (Linked Open Vocabularies) [5]: a repository of linked data vocabularies, facilitating ontology reuse and interoperability.
- OntoHub [6]: a web-based repository for distributed ontologies, models and specifications.

[1] https://www.w3.org/wiki/Lists_of_ontologies
[2] https://archivo.dbpedia.org/list
[3] https://agroportal.lirmm.fr/ontologies?search=o
[4] https://bioportal.bioontology.org/ontologies
[5] https://lov.linkeddata.es/dataset/lov/
[6] https://github.com/ontohub/ontohub

Any user can contribute to these repository by submitting a new ontology or commenting and suggesting improvements to existing ontologies.

2.2 Ontology Engineering

Ontology Engineering is the discipline that focuses on the design, development, and management of ontologies. It involves various activities such as ontology specification, knowledge acquisition, conceptualization, formalization, implementation, evaluation, maintenance, and documentation. The goal is to create structured, machine-readable knowledge representations that facilitate interoperability, reasoning, and knowledge sharing. This process relies on formal languages like OWL and RDF and employs tools such as Protégé.[32] In this section, we describe the main state-of-art techniques and methodologies in ontology engineering. Finally we discuss some ontology design patterns that can be reused in the ontology design.

Requirements Definition and Conceptualization

The definition of requirements and the subsequent conceptualization are two fundamental tasks in ontology engineering. They aim at establishing the objectives and scope of the ontology, and the constructs within it, respectively. To define the scope and the requirements that the ontology should be able to address, Competency Questions (CQs) stand for the state-of-the-art approach. They are questions written in natural language that the ontology should be able to answer [54].

For instance, in the development of an ontology for noise pollution monitoring [26], the following CQs have been defined:

Id	Competency question	Answer
CQ1	Which is the measured noise level in a specific moment and location?	The detected noise level during April in location with coordinates 40.42, – 3.69, 648 is 67.4 dB.
CQ2	Which is the location of a specific measurement station?	The location of the measurement station “Paseo de Recoletos” is 40.42, – 3.69, 648 (Longitude, Latitude, Altitude)
CQ3	In which day interval the noise level has been detected?	The noise level Ld has been detected in the time interval between 07:00 to 19:00.
CQ4	Which was the last calibration sensor date?	The datetime, for example, 2017-04-21T14:00:00+01:00

TABLE 2.2: Competency Questions noise pollution ontology

Each CQ must correspond to a natural language answer that "simulates" the response provided by the ontology. CQs are translated into SPARQL

queries in order to be executed on the ontology. Once defined, CQs can be used for evaluating an ontology.

The conceptualization (or design) of the ontology aims at defining the classes, the relationships among them, and the individuals. There are three approaches to conceptualization:

1. Top-down development process starts with the definition of the most general concepts in the domain and subsequent specialization of the concepts.

2. Bottom-up development process starts with the definition of the most specific classes, the leaves of the hierarchy, with subsequent grouping of these classes into more general concepts.

3. Hybrid development process is a combination of the top-down and bottom-up approaches: we define the most salient concepts first and then generalize and specialize them appropriately.

Methodologies

METHONTOLOGY METHONTOLOGY [34] proposes a structured method consisting of six steps: Firstly, the specification phase aims at producing the ontology specification document, outlining the main requirements and the scope of the ontology. It can be informal, semi-formal, or formal, and is written either in natural language or using CQs. Next, the knowledge acquisition phase involves gathering all relevant knowledge related to the domain. It can be performed based on various techniques, such as text analysis, brainstorming, interviews, and reviewing existing similar ontologies.

The conceptualization phase follows, where the acquired knowledge is organized into a conceptual model. During the integration phase, the methodology encourages reusing and integrating existing ontologies rather than developing new ones from scratch. This step is crucial for ensuring consistency and coherence, as it involves checking libraries of ontologies to find definitions of terms that match the semantics identified in the conceptualization phase. The implementation phase is where the actual construction of the ontology takes place. Finally, the evaluation phase ensures the quality and correctness of the ontology. This step involves both verification, to confirm that the ontology is semantically sound, by using a reasoner, and validation.

The methodology has the following advantages:

- It proposes the use of an evolutionary prototype life cycle, which allows definitions to be modified, added or removed at any time.

- It encourages the reuse of already existing ontologies, which can speed up development and ensure consistency with other projects.

However, it also has disadvantages:

- The methodology is very detailed and rigorous, which can be time and resource consuming, especially for smaller projects, it may be excessive compared to the needs of the project.

- The methodology can be complex to master for those without experience in ontology development, as it requires familiarity with several specific techniques and tools.
- The methodology has been developed in 1997, making it outdated by today standards. Since its creation, there have been significant advancements in both ontology engineering and related technologies, such as the Semantic Web, Linked Data, and agile methodologies.
- The methodology was created before the widespread adoption of agile development practices, which are now a common approach in software and ontology engineering. This makes it less flexible for fast-paced, iterative development environments.

NeOn Another proposed methodology is the NeOn methodology [2, 3], which provides guidance for all key aspects of the ontology engineering process. The concepts underlying this methodology are: scenarios, collaborative ontology development, reuse of ontological and non-ontological resources and evolution of networked ontologies.

The ontology engineering is based on nine scenarios:

- Scenario 1: from specification to implementation The ontology network is developed from scratch, engineers should specify ontology requirements and search for potential resources to be reused.
- Scenario 2: reusing ontological resources Engineers reuse ontological resources, ontology modules and ontology statements to build ontology networks.
- Scenario 3: reusing and re-engineering ontological resources Ontology engineers reuse and re-engineer ontological resources
- Scenario 4: reusing and merging ontological resources This scenario arises when several ontological resources in the same domain are selected for reuse, and engineers wish to create a new ontological resource with the selected resources
- Scenario 5: reusing, merging and re-engineering ontological resources Ontology engineers reuse, merge, and re-engineer ontological resources. This scenario is similar to Scenario 4, but here engineers decide to re-engineer the set of merged resources
- Scenario 6: reusing ontology design patterns Ontology engineers access repositories to reuse design patterns
- Scenario 7: ontology engineers access repositories to reuse design patterns Ontology engineers restructure (e.g., modularize, prune, extend, and/or specialize) ontological resources to be integrated in the ontology network.

- Scenario 8: localizing ontological resources Ontology engineers adapt an ontology to other languages and culture communities, thus obtaining a multilingual ontology.

The scenarios presented are not a rigid workflow, they are more like a variety of pathways for developing ontologies and they cover commonly occurring situations, for example, when available ontologies need to be re-engineered, aligned, modularized, localized to support different languages and cultures, and integrated with ontology design patterns and non-ontological resources, such as folksonomies or thesauri[81].

The Neon project offers to engineers a tool called NeOn toolkit: an open-source multi-platform ontology editor which supports OWL2 and features basic editing and visualization functionalities [25].

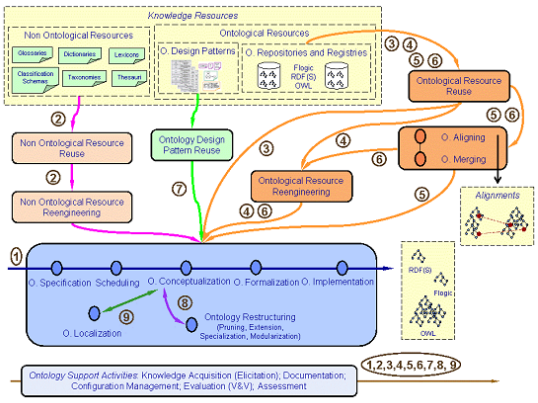

FIGURE 2.4: NeOn methodology scenarios

Fig. 2.4 represents the different scenarios in the NeOn methodology. The NeOn methodology has the following advantages:

- It identifies nine different scenarios for building ontology networks, which can be combined depending on the project's needs, this contrasts with more rigid methodologies like METHONTOLOGY.

- It supports the reuse and re-engineering of both ontological and non-ontological resources, which can save time and effort by leveraging existing knowledge bases, reducing the need to build everything from scratch.

- By basing its structure on real use cases and generalizing from them, the NeOn methodology covers practical needs encountered in ontology development projects.

However, NeOn has the following disadvantages:

- The flexibility and comprehensive nature of the methodology can make it more complex to apply, especially for teams without significant prior experience in ontology development.

- Although NeOn is designed for building large-scale networks, the actual process of merging and aligning different ontologies can be challenging, especially when ontologies come from diverse sources with differing structures and standards.

- The methodology is meant to be technology-independent, in practice, it may require specific tools to implement the processes effectively, and these tools might not always be accessible or easy to use for all teams.

XD methodology While the NeOn methodology is scenario-based, the eXtreme Design (XD) methodology [68] is based on a sequence of tasks inspired by the principles of agile development in software engineering. XD involves the stakeholders in order to gather complete requirements without making incorrect assumptions. This interaction is useful for defining CQs and user stories that detail the use of the ontology, supported by contextual statements that make implicit knowledge explicit. The principles of XD not only focus on the customer, but also provide clear guidelines to engineers regarding the design and organization of work. In particular, the design must be modular and task-oriented, focusing on the specifications defined in the CQs. The CQs are used in the development of unit tests during the testing phase, verifying the compliance of the ontology with the customer stories. Tests are generally conducted using SPARQL queries that encode the CQs. Throughout the development phase, the engineers are organized according to the pair design, where the design team is divided into pairs that must collaborate, by sharing knowledge. Based on the illustrated principles, XD defines the following tasks:

- Task 1. Get into the project context: the development process begins by aligning the engineers and domain experts, who may have different backgrounds and terminologies. The goal is twofold: to familiarize the customer with the project's methods and tools, and to provide the engineers with an understanding of the problem, scope, and initial terminology as well as establishing a collaborative environment for sharing documentation and discussing modelling issues.

- Task 2. Collect requirement stories: the customer writes stories starting from real scenarios in order to provide examples of the typical facts that should be stored in the ontology.

- Task 3. Select a story that has not been treated yet: each pair of engineers selects a story to focus on for the next iteration for releasing an ontology module. They create a new wiki page with the story's title and content based on the information from the card.

- Task 4. Transform the story into CQs: create a sequence of CQs, this task involves customer for having feedback/clarifications.

- Task 5. Match the CQ to use cases: identify candidate Content Patterns (CPs) (a specific type of ontology design patterns, see Sect. 2.2) based

on the CQs that express part of the ontology to be modelled. Matching can be done with tool support, such as keyword-based searching, or manually by engineers familiar with available CPs.

- Task 6. Select the CPs to reuse: select which of those patterns should be used for solving the modelling problem.
- Task 7. Reuse and integrate selected CPs: apply typical operations for CPs i.e. import, specialization, and composition.
- Task 8. Test and fix: validate the module by ensuring it aligns with the CQs just modelled. This process involves several steps: first, the CQ is transformed into a unit test, such as a SPARQL query, then, the module is populated with sample data based on the story, and the test is run. If the result isn't as expected, the module is revised and the test is rerun until it passes and once all tests linked to the story have been successfully completed, the engineers can move on to the next task. If any CQ remains unaddressed, they return to an earlier task to make adjustments.
- Task 9. Release module: release an ontology module having an URI shared with the whole team and on web repositories.
- Task 10. Integrate, test and fix: integrate the module with the other modules that constitute the ontology.

Once the eleven tasks have been completed, the release of the new version of the ontology takes place. The XD methodology has the following advantages:

- The methodology supports collaboration and modular ontology development, allowing teams to work on different parts of the ontology in parallel, which is especially useful for large and distributed projects.
- It employs a test-driven approach, where CQs and contextual statements serve as the basis for unit testing the ontology, ensuring that each component meets its intended purpose before moving forward.
- The methodology emphasizes constant interaction with domain experts and stakeholders to ensure that the ontology meets the intended use and requirements. This minimizes assumptions and ensures that implicit knowledge is captured.

However, there are also disadvantages:

- The methodology's iterative, test-driven nature may be overkill for smaller ontology projects, where a simpler, more direct approach could be more efficient.
- Effective use of XD relies on the availability of supporting tools, such as the NeOn Toolkit and its XD plugin. While these tools are valuable, they may limit accessibility if they are not well-maintained or widely available.

LOT methodology The Linked Open Terms (LOT) methodology [67] is a lightweight industrial-oriented methodology for ontology engineering. The methodology is compatible with software engineering methodologies; indeed, it is inspired by the agile methodology.

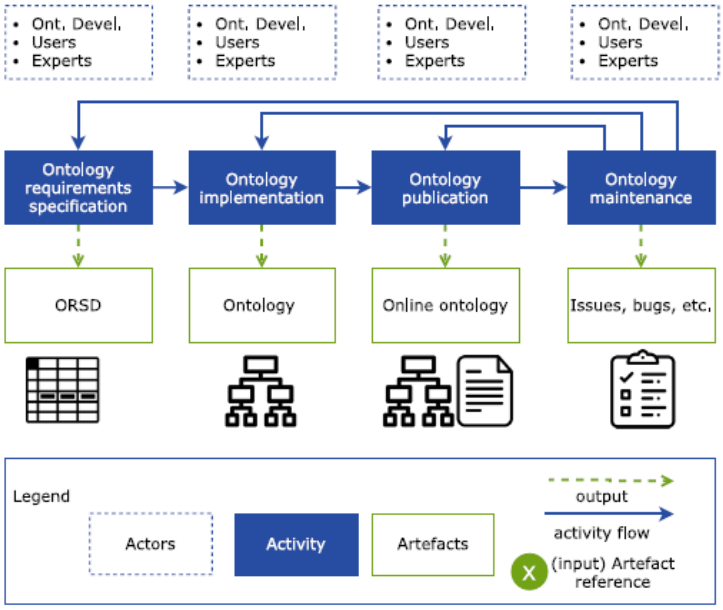

FIGURE 2.5: LOT methodology base workflow

Fig. 2.5 describes the ontology development lifecycle with four stages: requirements specification, implementation, publication, and maintenance, involving collaboration between ontologists, developers, users, and experts. Each stage outputs artefacts (e.g., ORSD, ontology, online ontology) that flow into subsequent phases for iterative refinement and usage.

This phase aims at defining the domain of interest, the goal of the ontology and the implementation language. In detail, there are seven steps as described below. Initially, use case specifications are created in collaboration with domain experts, users, and ontology engineers. These use cases describe scenarios in which the ontology will be utilized and are expressed in natural language. Next, data exchange identification is carried out, where the necessary documentation and resources related to the domain are gathered and compiled into a set of documents. The purpose and the scope of the ontology are then identified with input from users and domain experts. The purpose clarifies the reason for creating the ontology, while the scope defines the aspects of the domain that will be represented.

Subsequently, a set of functional ontological requirements is proposed, often in the form of CQs or natural language statements. These requirements outline what the ontology should be able to answer and represent. Ontology experts then review these functional requirements to ensure their correctness and completeness, addressing any ambiguities and verifying consistency with the defined purpose and scope. Following this, the Ontology Requirements Specification Document (ORSD) is formalized. This document

consolidates all the information, CQs, and functional requirements defined in the earlier phases, serving as a comprehensive reference for the ontology engineering. In some cases, an optional activity is conducted where the functional ontology requirements are further formalized into test cases, expressed in SPARQL queries. This step helps in validating that the ontology meets the specified requirements through structured testing.

In this phase, the ontology is implemented using a formal language. Firstly, it involves the conceptualization, which aims at defining the constructs in the ontology. Next, the encoding phase aims at actually implementing the ontology using an implementation language; in this phase, is useful to take into account other similar ontologies that could be reused. Once the whole ontology is completed, it can be evaluated according to different evaluation criteria.

In this phase, the ontology is published online, making it accessible and searchable by users. Before publishing the ontology online, it is necessary to identify the version to be released, document the ontology using human-readable HTML pages, and finally publish the ontology online according to the W3C guidelines [4].

The last step is the ontology maintenance. In this phase, once the ontology is published online, bug detection is performed. Moreover, the ontology can be updated according to new requirements.

Despite being introduced recently, the LOT methodology has been widely applied successfully, both in research and in the industrial field. In the industrial and manufacturing sector, the ontology SAREF4INMA [75] has been developed to describe equipment, materials, products, factories and manufacturing processes in a structured way. The industry is not the only sector where an ontology developed with the LOT method can be applied. In Spain, specifically in Madrid, an ontology has been developed to represent the Bus Public Transport system [76]. The ontology represents information on lines, routes, journey patterns and their timetables, stops on each route, information on expected bus arrival times for each stop, and information on incidents that may affect the bus routes and their journeys. Still in Spain, the LOT methodology has been successfully used in the development of a noise pollution ontology [26]. The ontology integrates data from Internet of Things (IoT) sensors in the city and establishes semantic relationships between them.

The LOT methodology has been used also in AI research field, the HALO ontology [59] has been creating for representing hallucinations in LLMs. The ontology represents LLMs, hallucinations, prompts and responses. The ontology consists of two modules and was created from a dataset of 40 known prompts designed to generate hallucinations. Each prompt was given as input to three LLMs: ChatGPT, BARD, and Claude.

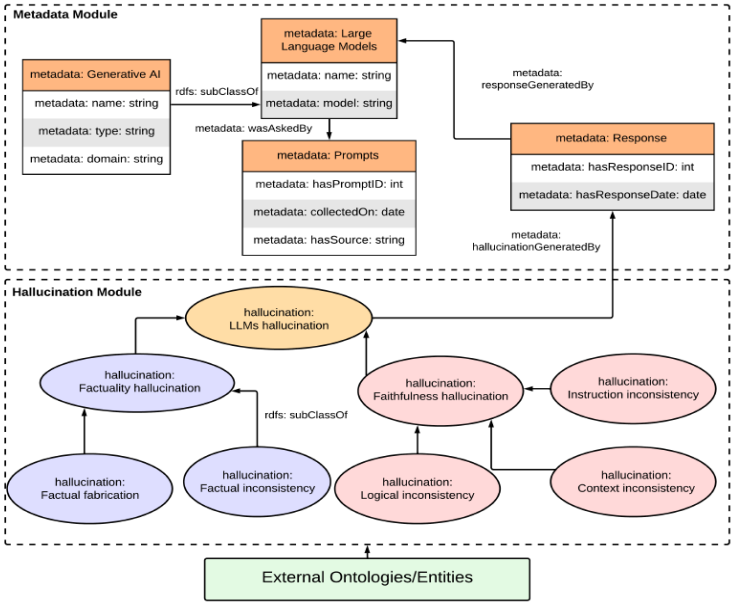

FIGURE 2.6: HALO ontology

Fig. 2.6
The LOT methodology has the following advantages:

- The LOT methodology is designed to be lightweight and agile, which allows for faster iterations and easier integration with software development practices and it is particularly well-suited for projects that need rapid, iterative development.
- It offers flexibility by combining various techniques (CQs, natural language statements ...) to define requirements and allows for different levels of formality depending on the project.
- Many open source ontologies have been developed successfully using the LOT methodology, e.g., the public transport ontology.

But, the LOT methodology has two disadvantages:

- LOT encourages the reuse of existing ontologies, but this can be challenging due to the potential inconsistencies and heterogeneity of reused ontologies. However, the methodology does not fully address how to resolve these issues.
- The emphasis of LOT on formal validation processes (such as the use of SPARQL queries) may not be sufficient for certain complex domains or non-technical stakeholders.

Ontology Engineering using Large Language Models

The methodologies analysed for ontology design involve users, domain experts, and ontology engineers, but do not include the use of advanced tools

based on artificial intelligence, such as LLMs. The study "From human experts to machines: An LLM supported approach to ontology and knowledge graph construction" [47] introduces a semi-automatic pipeline for the construction of knowledge graphs. There are six phases in the proposed pipeline. The process begins with the data collection phase, which does not involve the use of large language models (LLMs). Instead, domain experts generate a dataset and a set of publications relevant to the topic under study. In the next phase, known as competency question (CQ) generation, ChatGPT-3.5 is utilized to create competency questions. These questions are designed to describe the data collected in the previous phase at an abstract level. Domain experts provide inputs to the LLM to generate these questions, which are then evaluated by two domain experts to ensure their relevance and accuracy. Following this, the ontology creation phase is initiated. It builds on the competency questions generated earlier. This phase involves two steps: in the first step, a prompt is created that includes a competency question, the expected output, and a set of instructions. In the second step, the LLM is provided with a basic ontology structure containing the concepts and relationships derived from the competency questions. The CQ answering phase involves applying text processing techniques to extract answers to the competency questions. These answers are used to enrich the ontology and provide a deeper understanding of the dataset. During the knowledge graph (KG) construction phase, the competency questions, their answers, and the ontology generated by the LLM are combined and provided as input to the LLM. The prompt for this phase focuses on extracting entities, relationships, and concepts and mapping them onto the ontology, thereby creating a comprehensive knowledge graph. Finally, in the evaluation phase, the LLM evaluates the competency questions and the constructed knowledge graph by comparing them to a ground truth generated by human evaluators. The LLM assigns a score based on the alignment of the generated knowledge graph with the human-generated ground truth. The presented method enabled the creation of a fairly complex ontology consisting of 45 classes, 41 relations, and 365 axioms. However, it required significant computational resources, as an 80GB NVIDIA A100 GPU machine was used. Another approach for ontology learning using large language models is NeOn-GPT [33]. Starting from the NeOn methodology, the methodology is converted to a series of prompts to ChatGPT-3.5, these prompt are created using prompt engineering techniques and they are used first to generate ontology specification and competency questions. ChatGPT-3.5 is used also to generate a conceptual model of the ontology in the form of subject-relation-object triples and to generate implementation code. Once the code is generated, there is the syntax validation using the RDFLib python library [**rdflib**] and the consistency check in order to check logical inconsistencies. The final step is the pitfall resolution for checking circular axioms and missing disjointedness using OODPs API. The presented methodology is an evolution of the NeOn methodology, in which all tasks traditionally performed by the engineer are delegated to ChatGPT. This includes identifying requirements, writing competency questions, conceptualizing, and implementing

the ontology.ChatGPT can be used not only for the creation of ontologies but also for the creation of more complex knowledge graphs, specifically Hyper-Relational Knowledge Graphs, as discussed in the study "Construction of hyper-relational knowledge graph using pre-trained large language models" [20]. Hyper-relational knowledge graphs unlike the normal knowledge graphs incorporate additional qualifiers or attribute-value pairs [93], in the study starting from HyperRED dataset, a dataset about different topics , the hyper relational knowledge graph is built including in prompts the data from the dataset. Prompts are generated using Chain-of-Thoughts prompt engineering technique (CoT) and in order to evaluate the result generated, BERTScore is choose as a metric. The results obtained were not satisfactory, as the other algorithm(CubeRE)for extracting hyper-relational information was not surpassed. In general, large language models have shown promising potential in automating and enhancing the processes of ontology and knowledge graph construction replacing the ontology engineer in different tasks.

Ontology Design Patterns

In ontology engineering, Ontology Design Patterns (ODPs) [62] are a powerful approach for addressing recurring problems in the design of ontologies [40].

An OP is a modelling solution for solving a recurrent ontology design problem [36]. These patterns encapsulate best practices, enabling ontology engineers to draw from proven solutions when confronted with similar challenges and enabling them to optimize time and costs during the design.

In [31] there are five types of ODPs:

1. Structural ODPs
2. Reasoning ODPs
3. Correspondence ODPs
4. Presentation ODPs
5. Lexico-Syntactic ODPs
6. Content ODPs

We describe each type in detail below.

Structural ODPs are high-level modelling solutions that address architectural and logical challenges in ontology design. They include two main subtypes:

- Logical ODPs: these are combinations of logical constructs that resolve expressivity issues in ontology languages. They operate independently of specific content domains and are designed to bridge gaps where a representation language lacks built-in constructs. For example, Logical ODPs help model n-ary relations in OWL, which only supports binary relations.
- Architectural ODPs: these patterns influence the overall shape and structure of an ontology. They can be internal, focusing on consistent use of Logical ODPs within an ontology or external, involving meta-level constructs like modular networks connecting multiple ontologies.

Reasoning ODPs are applications of Logical ODPs specifically designed to achieve certain reasoning outcomes. Examples of Reasoning ODPs include classification, subsumption, inheritance, materialization, de-anonymization, and others. When applied to an ontology, Reasoning ODPs provide insights into the state of the ontology and enable a system to determine the type of reasoning required to perform tasks such as queries, evaluations, and more.

Correspondence ODPs are a type of ODP that focuses on addressing relationships and transformations between ontologies or between ontologies and non-ontological resources. They are divided into two main types:

- Re-engineering ODPs: provide guidelines for transforming ontological and non-ontological models, such as database schemas or thesauri, into ontologies.

- Mapping ODPs: These focus on creating semantic associations between elements of different ontologies without altering their structure. They help define equivalence, containment, and overlap relations as well as their negative counterparts.

Presentation ODPs focus on improving the usability and readability of ontologies from a user perspective. They serve as best practices to enhance reusability by making ontologies easier to evaluate and select. Key types include:

- Naming ODPs: provide conventions for naming ontology elements (classes, properties, files, namespaces). These ensure consistency and improve human readability and understanding.

- Annotation ODPs: define annotation properties or annotation property schemas that add metadata to ontology elements, enhancing their understandability and context.

For example, a Naming OP might recommend using the base URI of the publishing organization (e.g., http://www.example.org/ontologies/) to ensure clarity and consistency. Annotation ODPs could include descriptive metadata, such as the purpose or scope of a class. These patterns are essential for supporting effective reuse, adaptation, and user accessibility in diverse domains.

Content Patterns (CPs) are conceptual ODPs that address domain-specific modelling issues in ontologies. Unlike Logical ODPs, which are general and conceptualization-independent, CPs focus on solving content problems by proposing patterns for modelling domain-specific classes and properties. CPs are reusable through specialization, extension, or composition and impact only the ontology regions addressing particular modelling problems. Although language-independent in theory, CPs require implementation for practical reuse, often using OWL.

Lexico-Syntactic ODPs are linguistic structures or patterns composed of specific word types arranged in a particular order. These patterns allow for generalization and enable the extraction of conclusions about the meaning they convey. They are particularly useful for linking basic Logical and Content ODPs with natural language sentences, such as in educational or instructional contexts.

A more practical approach to ODPs is given by the Modular Ontology Design Library (MODL) [79]. MODL is a curated, well-documented, and structured collection of ODPs implemented in OWL designed to support modular ontology design. This library promotes the FAIR (Findable, Accessible, Interoperable, and Reusable) guiding principles for scientific data management and stewardship by addressing challenges like ontology reusability and interoperability. The library includes five categories of patterns: meta-patterns, organization of data, space-time movement, agents and roles, description, and details. MODL offers patterns for dynamic processes, complex roles, and detailed relationships like part-whole hierarchies. Its diversity ensures modularity and expressiveness for both general and domain-specific needs.

2.3 Large Language Models

Introduction and Classification

LLMs, as seen before, are capable of supporting ontology engineering. More broadly, this revolutionary AI technology is applied to various tasks across diverse domains, mainly because of their remarkable performance and versatility. A LLM is a large-scale, pre-trained statistical LM based on neural networks; it is capable of natural language generation, which can be used to frame a large variety of Natural Language Processing (NLP) tasks [99]. LLMs are trained on a vast amount of training data in order to predict the next word in a given sequence of words of any length, language and even including mathematical formulas and code [99]. LLMs are able to execute different heterogeneous tasks thanks to an underlying complex neural-network, with transformer architecture being the most advanced. Introduced in the 2017 by the seminal paper "Attention Is All You Need" [88], the transformer architecture is composed by two parts:

1. Encoder: a multilayer component where each layer consists of a multi-head self-attention mechanism and a position-wise fully connected feed-forward network.

2. Decoder: a multi-layer component that processes the encoder's output. Each layer is based on a modified self-attention mechanism that prevents future positions from influencing past positions, ensuring autoregressive property.

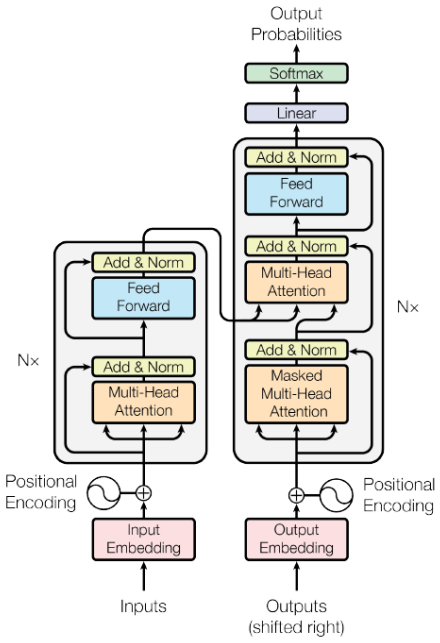

FIGURE 2.7: Transformer architecture

Fig. 2.7 depicts the transformer architecture with with the encoder (on the left) and the decoder (on the right). Two different variants of the transformer architecture have been adopted:

- Encoder-Decoder: processes inputs through the encoder and passes the intermediate representation to the decoder to generate the output [5].
- Decoder-only: does not have an encoder and it uses a decoder in order to generate tokens, each token depends on the previous sequence [21].
- Encoder-only: does not have a decoder and it processes the entire sequence simultaneously generating contextualized representations for each token[28].

The simplicity and versatility of the decoder-only architecture allows for reaching extremely large scale (billions of parameters and hundreds of gigabytes of training data). Such an unprecedented scale endowed LLMs with the so-called emerging abilities. They made LLMs able to solve a large variety of tasks with remarkable performance with no need for additional re-training or fine-tuning, but simply needing to be prompted. Indeed they are also called general purpose LLMs or Foundation Models. Such emerging abilities include:

- Instruction following: the ability of an LLM to execute commands and respond coherently to user instructions.
- In-context learning: the capability to learn and generalize from examples provided within the same interaction session, without updating the model's weights.
- step-by-step problem solving: the ability to break down complex problems into simpler steps, following a structured process to reach a solution.

Such abilities constitute the basis for the definition of effective prompts. Despite being less general than GPT, also BERT and T5 are often considered Foundation Models as they have represented crucial milestones in the development of LLMs. Moreover, for classification (summarization) tasks, BERT (T5) can still outperform GPT and other very large decoder-only LLMs.

State-of-the-art Large Language Models

General purpose LLMs are capable of performing a wide variety of tasks. In this section, we will analyse and discuss the following state-of-the-art general purpose LLMs:

- GPT
- BERT
- PaLM

- LLaMA
- T5
- BLOOM
- Mistral

The Generative Pre-trained Transformer (GPT), released by Open AI, is actually the most advanced and powerful LLM available to users. In two months, it gained over than 100 million of users, making it the fastest-growing consumer software application in history [6]. GPT is a decoder-only LLM.

Since its first version, GPT-1 released in 2018, the model has represented a step forward in AI. GPT-1 has 117 million parameters and is trained on the BookCorpus dataset. The model has been improved in 2019 with the next version GPT-2, which has 1.5 billions of parameters and is trained on WebText dataset with 8 millions of documents [71]. In 2020, OpenAI has released GPT-3, which has 175 billions of parameters and is trained over one trillion of documents. The latest version of GPT is GPT-4; it is a multimodal model, i.e., it can take as input both images and text and generate a response consisting of both images and text. Despite the continuous evolution and the billions of parameters of the model, it is still subject to issues, including hallucinations [6]. In general, an hallucination is a response generated by an AI model that contains false or misleading information presented as a fact. An example of hallucination is the following:

```
Prompt: Who won the 2024 Formula 1 constructors world
   championship?
Response: The Redbull team won the Formula 1 constructors
   world championship in 2024
```

The response is wrong because McLaren team won in 2024 the Formula 1 constructors world championship in 2024. Another example is:

```
Solve the equation: 3x - 11 = 5
Response: x = 4
```

The response is wrong because the correct result is 5.33 given by $x = \frac{11+5}{3}$.

GPT is a proprietary LLM, so its parameters and training set are not publicly available. GPT has been made available to the public as a web application called ChatGPT and represented in Fig. 2.8.

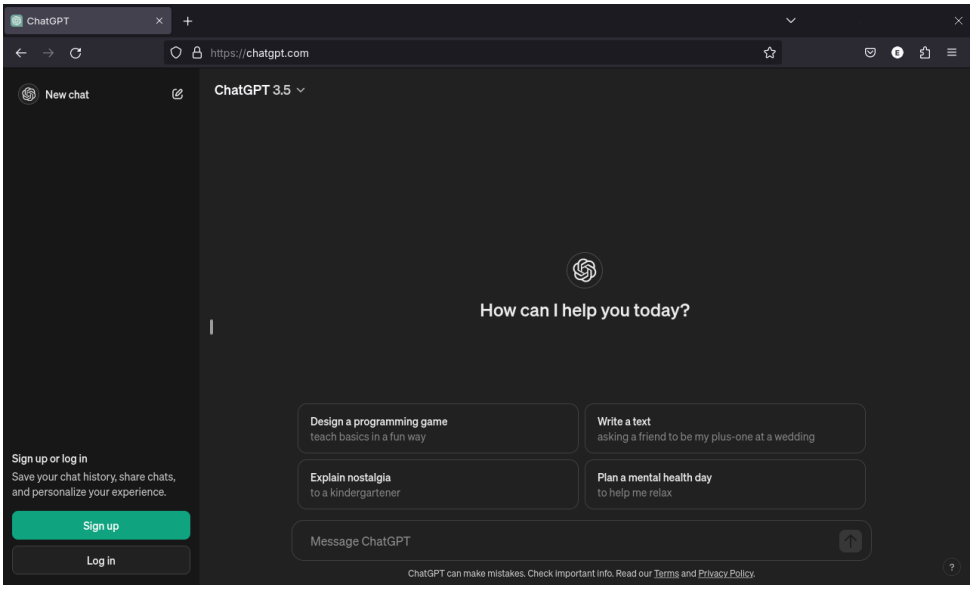

FIGURE 2.8: ChatGPT interface

Bidirectional Encoder Representations from Transformers (BERT) was introduced by Google AI in 2018 [46]. BERT is a encoder-only LLM. Compared to other models like GPT, BERT is a bidirectional model: it takes into account the context on the left and on the right of each input word. It has 340 milions of parameters. It is trained on the BookCorpus dataset [10] and on english Wikipedia. BERT is an open-source LLM.

Unlike GPT, it does not have a web interface, but the different versions of BERT are available for download on Huggingface [7].

In September 2023, Google AI released Pathways Language Model (PaLM). It is a decoder-only LLM. It has 540 billions of parameters and is trained with the Pathways system that allows parallel training [8, 11]. PaLM is trained using a combination of English and multilingual datasets (100 languages) that include high-quality web documents, books, Wikipedia, conversations, and GitHub code in order to learn to perform complex reasoning and code generation tasks[43]. PaLM is a proprietary model and it is available on the official website [8]. PaLM serves as the starting point for an even more powerful multimodal model: Google Gemini, which will be discussed later.

Large Language Model Meta AI (LLaMA) is a family of LLMs introduced in 2023 by another web giant: META. LLaMA models are decoder-only LLMs. The latest version LLaMA 3 includes over 400 billions of parameters [42]. LLaMA models, like PaLM, are trained using different sources, specifically: EnglishCommonCrawl dataset, C4 dataset, Github, Wikipedia, ArXiv and Stack Exchange. Thanks to the extensive datasets, LLaMA models are able to perform common sense reasoning, question-answering, mathematical reasoning, text comprehension and code generation. LLaMA models are open-source LLMs. LLaMA models can be downloaded on HuggingFace [9] or on

[7] https://huggingface.co/docs/transformers/model_doc/bert

[8] https://cloud.google.com/vertex-ai/generative-ai/docs/language-model-overview?hl=it#palm-api

[9] https://huggingface.co/meta-llama

the official META website [10].

Text-to-Text Transfer Transformer (T5) is an encoder-decoder LLM, introduced in 2020 by Google Research, pre-trained on a multi-task mixture of unsupervised and supervised tasks [72]. The biggest version has 11 bilions of parameters. The model has been trained on the CommonCrawl dataset that includes 750 GB of english text in addition to Wikipedia and RealNews-like dataset. T5 is an open-source LLM. All the versions of T5 are available for download on HuggingFace [11].

BigScience Large Open-science Open-access Multilingual Language Model (BLOOM) is an open-source decoder-only LLM released in 2022. In detail, the model has 175 billions of parameters and is trained on a dataset that includes 46 natural languages and 13 programming languages in form of 350 billions of unique tokens [49]. BLOOM currently is the biggest and the most powerful open-source LLM: it can be used for multilingual content generation, software development and research [29]. BLOOM is available on HuggingFace [12].

Mistral is a LLM released in September 2023 by the French startup MistralAI [44]. The biggest version has 123 bilions of parameters. Mistral has text and code generation capabilities. It is released in a free and in a premium version. It is available as a web interface[13] as represented in Fig. 2.9.

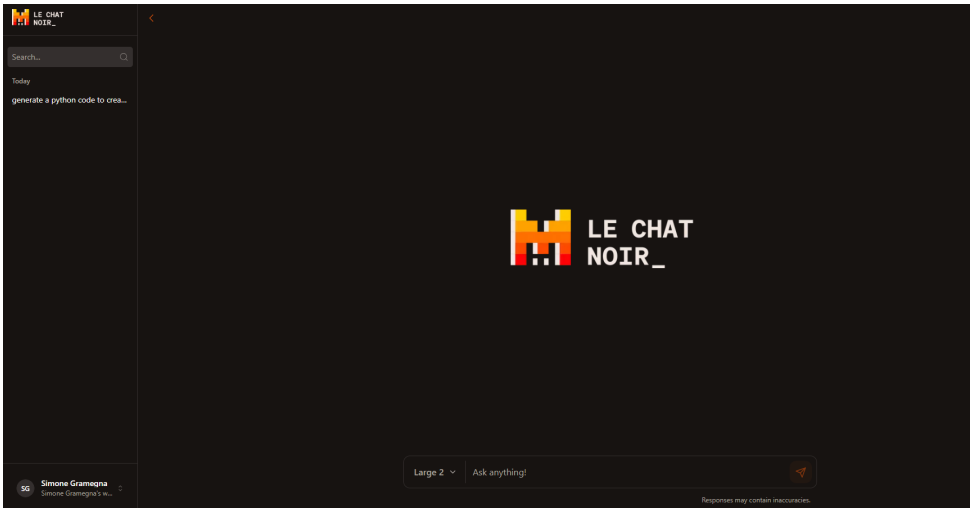

FIGURE 2.9: Mistral web interface

State-of-the-art Multimodal Large Language Models

An evolution of LLMs is represented by Multimodal Large Language Models MLLMs, which are capable of processing information in various forms: text, images, audio, video, and documents. They are based on a modified version

[10] https://www.llama.com/
[11] https://huggingface.co/docs/transformers/model_doc/t5
[12] https://huggingface.co/bigscience/bloom
[13] https://chat.mistral.ai/chat

of the transformer architecture that is able to learn embeddings (vector representations) also of images, audio, and video. Processing these embeddings through a multi-layered network generates outputs as text and/or media [89] as represented in Fig. 2.10.

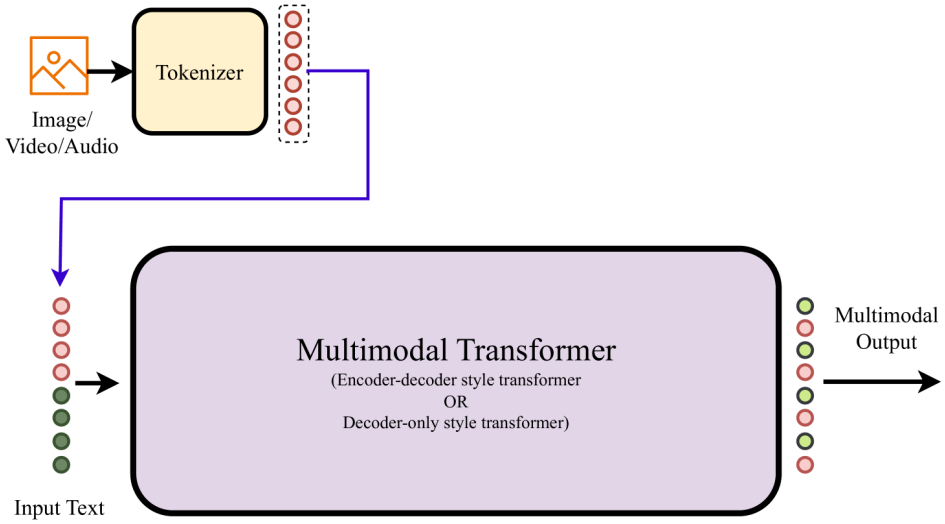

FIGURE 2.10: Example of multimodal transformer architecture

Several MLLMs have been introduced as evolutions of classic LLMs. We will cover the following state-of-the-art MLLMs:

- GPT-4
- Google Gemini
- GLaMM

GPT-4 is the latest version of the most popular LLM: GPT. Released in March 2023, it became available to premium users of the ChatGPT platform in three distinct versions: GPT-4, GPT-4 Turbo, and GPT-4o (omni). These versions represent an evolution of GPT-3, enhancing performance and efficiency [27]. The multimodal version of GPT-4, known as GPT-4o where the "o" stands for "omni," highlighting the "omniscience" that this model is capable of:

- Real-time voice communication: engage in real-time voice conversations, understanding spoken input and generating natural-sounding voice responses.
- Real-time vision: process and understand vision data like images and video and their semantics. The model can generate an image starting from text and applying styles or generating descriptions of an image.
- Data and Chart Reading: read and interpret data in forms of charts and graphs.

These features make GPT-4o a versatile and powerful tool applicable in many sectors, with particular emphasis on programming and data analysis. Details regarding the dataset, number of parameters, and training technique are not disclosed for competitive reasons.

The latest evolution of Google's LLMs is Gemini [41]. It has been introduced in 2023 and is the evolution of Bard: the chatbot based on PaLM, which is discussed before. Gemini is a proprietary model and it has proven to be superior to its rival GPT-4 on various benchmarks.

Grounding LLM (GLaMM) has been released in 2023 by Mohamed bin Zayed University of AI [73]. GLaMM is a MLLM that integrates natural language with object segmentation masks in images, combining scene-level, region-level, and pixel-level understanding. The model uses an auto-encoder to process images and an LLM to understand natural language. It is an open-source LLM: the source code and parameters are available on the official GitHub repository [14].

2.4 Prompting

Definition and Structure

At the core of interaction with LLMs are prompts, i.e., instructions written in natural language given as input to LLMs to generate responses [52]. We can see a prompt like a mathematical function that takes in input a text and gives an output $f(x) = y$ in which f is the template applied to the text of the problem. For example, if we want to translate a phrase in English:

```
Translate {X} in english
```

This is just a simple technique in which there is the manual creation of the template, but in literature there are more complex techniques that we are going to discuss. In more detail, there are four main components of a prompt:

- Instruction: (mandatory) a specific task or instruction you want the model to perform.
- Context: (optional) external information or additional context that can steer the model to better responses.
- Input data: (mandatory) the input or question that we are interested to find a response for.
- Output Indicator: (optional) the type or format of the output.

An example of output with all four components is the following:

```
Solve the following linear equation for x and show the
   steps clearly.
This is a basic algebra problem where you need to isolate x
    on one side of the equation.
```

[14] https://github.com/mbzuai-oryx/groundingLMM

```
Solve for x: 3x + 7 = 16
Provide the output in text format.
```

In this prompt the instruction is: "Solve the given linear equation for x.", the context is: "This is a basic algebra problem where you need to isolate x on one side of the equation.", the input data is: "Solve for x: 3x + 7 = 16" and the output indicator is: "Provide the output in text format." Context and output indicator are optional components but they are useful to improve the quality of the output.

Prompt Engineering

Prompt Engineering is a novel discipline concerned with prompt writing and verification. The first and simplest technique is the Zero-shot prompting in which no example is provided, and the LLM has no prior knowledge of the context, generating the output based solely on the given input prompt. An example is the following:

```
Classify the text into neutral, negative or positive.
Text: I think the vacation is okay.
Sentiment:
```

In Few-shot prompting, on the other hand, the input prompt provided to the LLM includes a few examples. Examples are useful to achieve a better understanding of the context, this is not possible with zero-shot prompting, as we can see in the following example:

```
Task: Translate the following sentences from italian to
   english.

Example "Oggi il tempo e' bello" output: "The weather is
   nice today"

input: "Mi piace leggere libri" // output: I love reading
   books
```

In Chain-of-Thought (CoT) Prompting, problem-solving is carried out step by step, breaking down a task into multiple stages to achieve more structured and accurate responses. An example is the following:

```
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis
   balls. Each can has 3 tennis balls. How many tennis
   balls does he have now?
A: Roger started with 5 balls. 2 cans of 3 tennis balls
   each is 6 tennis balls. 5 + 6 = 11. The answer is 11.
Q: The cafeteria had 23 apples. If they used 20 to make
   lunch and bought 6 more, how many apples do they have?

Model output: The cafeteria had 23 apples originally. They
   used 20 to make lunch. So they had 23 -  20 = 3.  They
   bought 6 more apples, so they have 3 + 6 = 9. The answer
    is 9.
```

The main issue with this technique is the manual intervention required, as the user must manually provide the reasoning that the LLM will follow. To address this problem, a variant called Automatic Chain-of-Thought (Auto-CoT) Prompting was introduced. Instead of manually providing the reasoning chain, the prompt includes the simple phrase, "Let's think step-by-step" for eliciting the model to engage in a CoT without adding examples of step-by-step problem solving in the prompt. Both CoT and Auto-CoT follow a single reasoning path, but we can generate multiple reasoning paths and choose the best answer; this is the basis of the Self-consistency prompting [90]. Techniques involving multiple prompts are useful in order to decide the best reasoning strategy but we can also decide not to use any complex reasoning, but a more simple strategy that can involve the ensembling, the augmentation, the composition or the decomposition of a single prompt [51]. In prompt ensembling, different prompts are combined on the same input in order to get a response. For instance:

```
PR1: China's capital is [X]
PR2: [X] is capital of China
PR3: The capital of China is [X]
```

To choose the response, we can apply several strategies, including majority voting and weighted voting. In prompt composition there is a composition of two or more subprompts in order to get the final prompt. In contrast, in prompt decomposition one main prompt is decomposed in two or more smaller subprompts. In LLMs, there is a significant issue that leads to incorrect responses, known as the problem of hallucinations. We refer to hallucination, in the context of LLMs, when there is the generation of texts or responses that are grammatically and syntactically correct but deviate from the given inputs or do not align with factual accuracy [96]. Prompt engineering techniques can help reduce this phenomenon, the main technique is Retrieval Augmented Generation (RAG). In this approach, starting from an input, a knowledge base of sources (web pages, articles, etc.) is created, and the text from these sources is incorporated into the subsequent input, providing context and generating more accurate and reliable responses. The improvement in performance and accuracy of the responses obtained from the LLM can also be achieved by incorporating the user's emotions into the prompt, known as Emotion Prompting. Incorporating emotions can be useful in tasks such as sentiment analysis of a sentence, but this is not the case for code generation, for which various prompt engineering techniques have been introduced. An evolution of the previously described Chain-of-Thoughts is Program of Thoughts (PoT) Prompting [18], where the large language model is given a prompt containing intermediate steps that include mathematical expressions or source code, as we can see in the following figure:

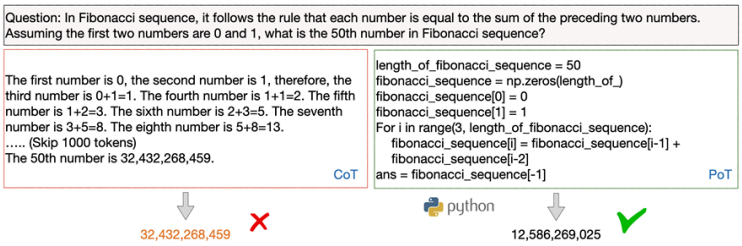

FIGURE 2.11: Program of Thoughts (PoT) Prompting

Another approach is the Scratchpad Prompting in which an arbitrary sequence of intermediate tokens is generated before the final answer is provided. The sequence is given in a tag called "scratch" [23, 61] in which the user writes intermediate steps useful to the LLM as shown below, were it is requested to solve a simple mathematical problem: The problem is: A store sells apples for 3 euros each. If I buy 5 apples and get a 10% discount, how much do I pay in total?

```
<scratch>
Perform the following calculations step by step. Write your
    notes in the format "Scratchpad:" for each step.

Problem: A store sells apples for 3 euros each. If I buy 5
   apples and get a 10% discount, how much do I pay in
   total?

Scratchpad:
1. The unit price of an apple is 3 euros.
2. The number of apples purchased is 5.
3. The total cost without discount is: 3 * 5 = 15 euros.
4. The 10% discount is: 15 * 0.10 = 1.5 euros.
5. The final price after the discount is: 15 - 1.5 = 13.5
   euros.
Final result: 13.5 euros.
</scratch>
```

In the example above, we begin the sequence with $< scratch >$ and we define the different steps that the model has to follow in order to solve the problem. The sequence ends with $< /scratch >$. The Rephrase and Respond (RaR) Prompting[22] technique automatically allows for the correction and improvement of the LLM's response through the prompt:

```
"{question}" Rephrase and expand the question, and respond.
```

This approach has four main advantages:

1. It lets the LLM self-improve the prompts while maintaining the context of the original query.
2. It allows better aligning the human's intended query with LLM's preferred style of question.
3. It expands the LLM's thought process and adding a step that will not naturally appear when using CoT.

4. It provides an approach for humans to interpret how LLMs understand the questions.

Another technique to obtain more accurate and comprehensive responses is the Take a Step Back Prompting[100], which consists of two steps. The first step involves inputting a specific question, and subsequently (step-back) inputting a more abstract, high-level question related to the task, as shown in the following figure:

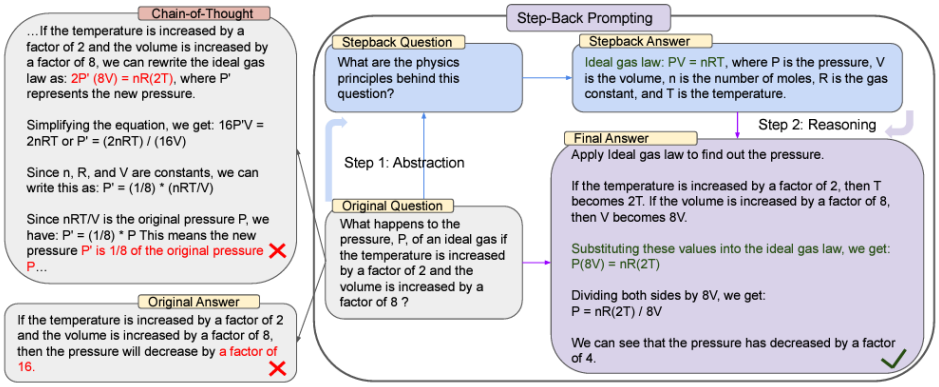

FIGURE 2.12: Take a Step Back Prompting

Prompt engineering in Multimodal Large Language Models

State-of-the-art prompt engineering techniques aim, when given as input to a LLM, to generate text. MLLMs allow for incorporating media such as image into prompts. Main tasks involving MLLMs are:

- Image generation
- Image classification
- Image editing

A preliminary approach to image classification is the use of zero-shot prompting and few-shot prompting [17]. In zero-shot prompting for image generation, no input examples are provided, but only the instruction to be executed, for example:

```
Classify the follwing image: [image.png]
```

In few-shot prompting, examples are provided to the model, for instance:

```
Given: [image1.png], [image2.png], [image3.png], classify
   the following image: [image.png]
```

Zero-shot prompting and few-shot prompting can be also used in image generation. However, this task is more complex and it can benefit from user feedback. An example is drawing a stick figure using letters of the alphabet [24], where the process starts from an initial prompt, such as:

```
Produce TikZ code that draws a person composed from letters
    in the alphabet. The arms and torso can be the letter Y
   , the face can be the letter O (add some facial features
   ) and the legs can be the legs of the letter H. Feel
   free to add other features.
```

The generated image is refined by adding details:

```
- The torso is a bit too long, the arms are too short
  and it looks like the right arm is carrying the face
     instead of the face being right above the torso. Could
      you correct this please?

- Please add a shirt and pants.
```

As we can see in the following example, where the latest version of ChatGPT, GPT4, has been applied:

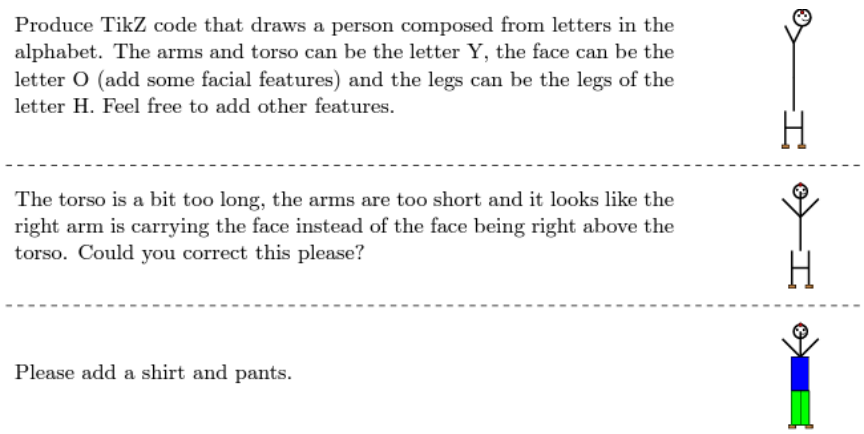

FIGURE 2.13: Drawing using GPT4

In image generation the characteristics of an image can be specified not only using user feedback but also using style modifiers. They are descriptors that consistently produce certain styles, for example: "tinted red", "made of glass", and combining those together they produce a more specific style [80]. For example, we want to generate, using DALL-E, a picture of a pyramid. The first picture is generated without using any modifier:

FIGURE 2.14: Pyramid picture with no modifiers

The second picture is generated using the prompt with three modifiers: A pyramid made of glass, rendered in Unity and tinted red getting the following result:

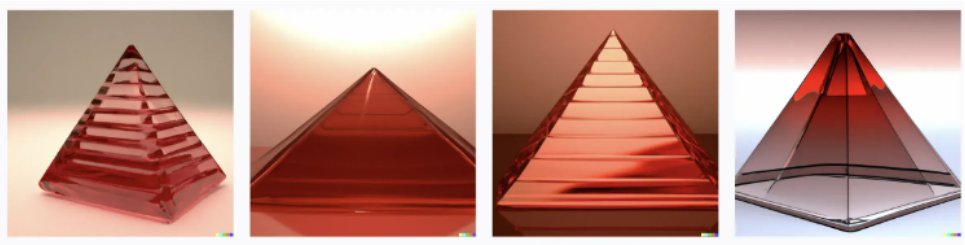

FIGURE 2.15: Pyramid picture with modifiers

In the prompt, in order to improve quality, we can add generic adjectives like: "amazing", "good", "beautiful" that are called quality boosters [65]. The Latin motto "repetita iuvant" that means that repeating words is useful can be applied in a certain sense in a prompt for image generation, the repetition technique gives more importance to some adjective instead of another it is possible to repeat more times the word, giving emphasis to it. For example, the generation of an image of a waterfall, but we want to emphasise the beauty of the picture we can use this following prompt:

```
A very very very very very beautiful painting of a mountain
    next to a waterfall
```

In this case the word "very" is repeated five times in order to give more importance to the adjective beautiful. Since repeating terms in a prompt can be weird and inaccurate, it is possible to use weighted terms technique[91]. In this technique a term has a numerical value (positive or negative) that corresponds to the weight, i.e., the importance of that term. For example, the generation of a picture of a mountain without trees, we use this prompt:

```
mountain | tree:-10.
```

in which the tree has negative weight and it doesn't have importance getting this result, as we can see in the figure 2.16:

FIGURE 2.16: Weighted terms image

Another approach in image generation is negative prompting, those allow users to exclude unwanted features and enhance the quality output.

Negative prompts are complementary to positive prompts which describe what the user wants, including the main subject and how it should look [55]. For example, we want to generate using a diffusion model a portrait of a man without moustache, so the prompt is:

```
- Positive prompt: Portrait photo of a man.
- Negative prompt: Moustache
```

Negative prompts can be applied on any image feature like: colors, lighting problems, unwanted elements, image quality and style. The Multimodal Chain-of-Thought prompting technique [98] is a variation of the Chain-of-Thought technique (CoT). This technique in Multi-CoT is enhanced by incorporating in the prompt text and visual inputs. The technique consists in two stages:

1. Rationale generation: the model processes the multimodal inputs in order to generate an intermediate reasoning chain (rationale). The rationale explains the logical steps required to arrive at the final answer, using both textual and visual content as context.

2. Answer inference: the generated rationale is combined with the original input and used by the model to infer the final answer.

as we can see in the figure 2.17 below:

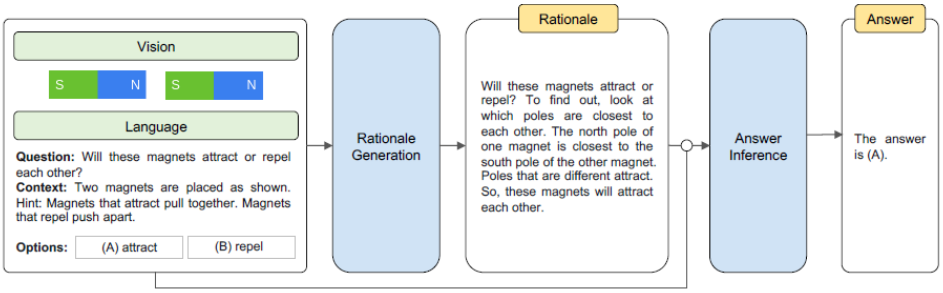

FIGURE 2.17: Multimodal Chain-of-Thought

Similar to the multimodal chain-of-thought, the Chain of Images (CoI) prompting technique [57] generates a series of images as intermediate representations in order to solve problems starting from a textual prompt.

Chapter 3

The Prompt Engineering Ontology: Requirements Specification

This chapter details the design phase of the Prompt Engineering Ontology PEO, based on LOT, a state-of-the-art ontology engineering methodology. Compared to other methodologies analysed, LOT is a recent methodology, introduced in 2022, and has been used in various projects such as the 'Ciudades Abiertas' project [19] for the construction of a set of ontologies used for sharing open data, and the BIMERR project [12], in which ontologies for sustainable construction were developed [14], among many others available on the LOT methodology website [1]. LOT has not only been successfully applied in industrial projects, but also in the development of various research ontologies, as seen in Chapter **??**, as it provides a straightforward and iterative method for designing and developing ontologies. Another reason for choosing LOT is its ease of learning, as it is inspired by the agile methodology in software development. The Linked Open Terms project not only provides very useful examples of ontologies to draw inspiration. LOT offers a Github repository [53] with all the necessary resources to be used in the specification of requirements. This last aspect is very important because the main drawback of the other methodologies analysed is the lack of concrete guidelines and the relevant tools to use, tools that, when mentioned, are often obsolete or inaccessible. This complicates the work of a developer who is approaching ontology engineering for the first time, as he needs to understand the fundamentals of the methodology but also figures out which development, validation, and testing software to use. The LOT methodology, thanks not only to the numerous available resources but also to the clear and precise description of the method and the recent tools to be used, resolves these issues and simplifies the developer's work. The LOT methodology has an inherently iterative nature and is oriented towards the publication of ontologies according to the FAIR principles [30], including specific recommendations, tips and potential tools that can be helpful to ontology developers. Moreover, the methodology extends the state-of-art methodologies like NeOn and Methontology with a modern approach.

The LOT methodology was preferred over recently introduced techniques that involve the use of LLMs. Although these techniques automate the process, they are computationally expensive and require numerous checks to

[1] https://lot.linkeddata.es/

verify the syntactic and semantic correctness of the produced artifact. Another downside is the lack of actual ontology projects implemented using LLMs, as these are very recent techniques that have not been tested on real projects but only on experimental cases. In the following sections, the application of the LOT methodology, methodology that has been discussed in Chapter **??**, to the design and implementation of PEO will be described, following the workflow outlined and described in the background chapter.

3.1 Use Case Specification

The LOT methodology includes six mandatory phases (plus one optional) in the ontology requirements specification. At the end of each phase, a document is produced containing the analysed aspect of the specifications, as shown in the following figure 3.1:

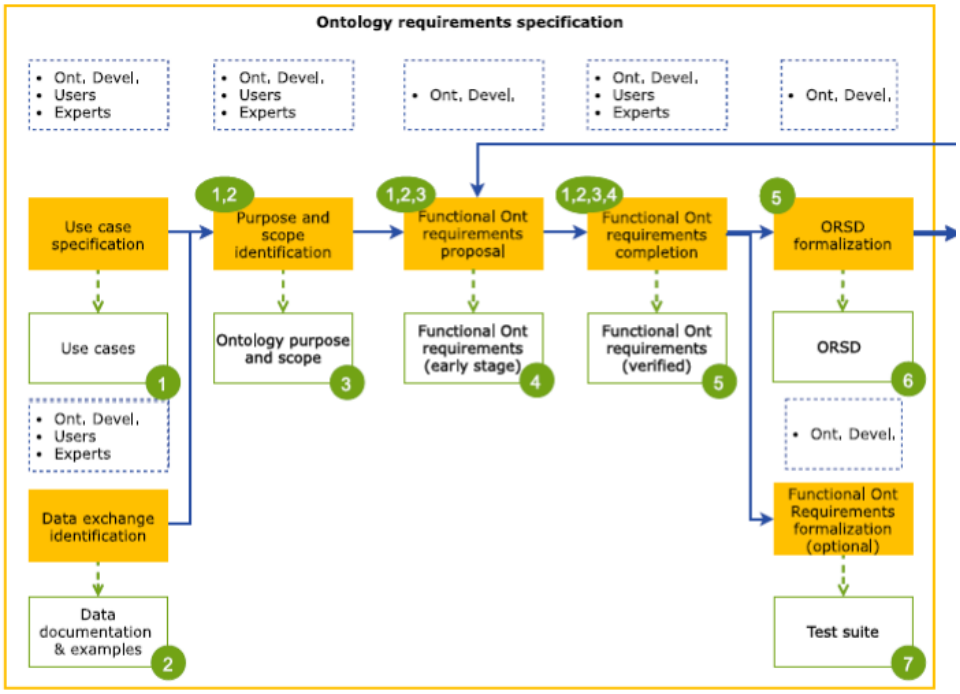

FIGURE 3.1: Ontology requirements specification workflow

In detail the phases are the following:

1. Use case specification
2. Data exchange identification
3. Purpose and scope identification
4. Functional ontology requirements proposal
5. Functional ontology requirements completion

6. ORSD formalization
7. Functional ontology requirements formalization (optional)

The first step in the design of PEO is the use case specification, this phase involves domain experts, ontology developers and users and it has the goal to imagine and specify how the ontology can be used in real life by real users. Taking into account the domain, the possible use and the possible users we decided to specify ten use cases. Each use case has a name, a description, a list of actors and a flow.

Use case 1

- Name: Optimizing LLM Responses
- Description: Researchers use PEO to design optimized prompts that improve LLM response quality.
- Actors: Researcher, PEO, LLM.
- Flow: The researcher uses the ontology to identify appropriate prompts for different types of tasks creating a set of selected prompts. Selected prompts are given as input to the LLM and responses are evaluated according to specific metrics on consistency, completeness and quality decided by the researcher.

Use case 2

- Name: Bias analysis
- Description: Researchers use PEO to generate responses and detect bias in the considered LLMs.
- Actors: Researcher, PEO, LLMs
- Flow: The researcher uses the ontology to generate prompts on sensitive topics (e.g., gender, ethnicity) and prompts are tested on one or more LLMs. Responses are collected using bias and fairness metrics and results are collected in order to improve prompts and considered models.

Use case 3

- Name: Code generation
- Description: Developers use prompt engineering techniques applied to a chosen LLM to generate source code in a specific programming language for a specific task.
- Actors: Developers, PEO, LLM

- Flow: The developer is working on an Android application written in Java and needs code to control the actions on a button. Using the ontology, he chooses the most appropriate prompt engineering technique and applies it to the creation of the prompt to a large language model of his choice, resulting in Java code as output. The code is tested and integrated into the application.

Use case 4

- Name: Prompt engineering lesson
- Description: The teacher uses PEO to teach prompt engineering techniques exploring different techniques and prompt described.
- Actors: Teacher, students, PEO.
- Flow: The teacher opens the ontology and shows with proper explanation different prompt engineering techniques represented in the ontology.

Use case 5

- Name: LLMs lesson
- Description: The teacher uses PEO to teach the different LLMs available.
- Actors: Teacher, students, PEO.
- Flow: The teacher opens the ontology and shows with proper explanation different LLMs represented in the ontology.

Use case 6

- Name: Social media content creation
- Description: The content creator uses PEO to generate prompts that optimize the creation of articles, social media posts and other textual content.
- Actors: Content creator, PEO, LLM
- Flow: The content creator opens the ontology and chooses the appropriate technique in order to generate text for a post on social media using a specific large language model. The content creator adapts the response according to his target.

Use case 7

- Name: Image generation
- Description: The content creator uses PEO to create a prompt to be given as input to a specific large language model capable of generating an image.
- Actors: Content creator, PEO, LLM.
- Flow: The content creator wants to create an AI-generated image for a video and he uses the ontology to choose the best large language models able to generate image, he chooses the prompt engineering technique to create the prompt in order to generate image. He watches the output and he continues to use the ontology to generate prompts in order to refine the image.

Use case 8

- Name: Prompt engineering experiments
- Description: Students use PEO to explore and create effective prompts to improve language model responses.
- Actors: Students, PEO, LLM.
- Flow: Students explore the ontology in order to understand different prompting techniques applying them to a chosen large language model.

Use case 9

- Name: Large language models learning
- Description: Students use PEO to learn different types of large language models.
- Actors: Students, PEO
- Flow: Students explore different large language models represented in the ontology and the relations among them, learning all the features of each large language model.

Use case 10

- Name: Explanation computer science topics
- Description: Student wants to generate a prompt using PEO in order to explain a computer science topic.
- Actors: Student, PEO, LLM
- Flow: The student uses the ontology to choose the most appropriate prompt engineering technique in order to generate using large language models. The student reads the obtained LLM response and can create a new prompt using another technique represented in the ontology.

3.2 Data Exchange Identification

The goal of the data exchange identification activity is to provide the necessary documentation about the domain to be modelled. Essentially, this phase involves gathering heterogeneous sources of information, such as scientific papers, websites, datasets, and similar existing ontologies. Since LOT does not provide a clear indication on how to represent this documentation, we simply noted the URLs of the gathered resources. The following list of papers was chosen as a source for PEO:

- A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications [78]
- Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing [51]
- A Survey of Large Language Models [99]
- Investigating Prompt Engineering in Diffusion Models [95]
- A Survey on Large Language Models: Applications, Challenges, Limitations, and Practical Usage [39]
- Large Language Models: A Survey [58]

The list of website chosen is the following:

- Prompting Guide
- Awesome-LLM
- List of LLMs
- Learn prompting

We decided to select these research papers and websites to build a well-structured and comprehensive ontology for prompt engineering and large language models. The academic papers provide in-depth analyses of prompting techniques, their applications, and the evolution of LLMs, offering a strong theoretical foundation. Meanwhile, the chosen websites serve as practical resources, curating up-to-date information on LLM architectures, prompting strategies and real-world use cases.

3.3 Purpose and Scope Identification

From use cases and the domain documentation provided in the data exchange identification task, in this phase there is the specification of the purpose: what is the objective of PEO and the specification of the scope of the ontology: what it is going to be represented in the ontology. The objective of PEO is to formalize knowledge about the creation and various types of

prompts for the different LLMs available by making it accessible to both experienced and less experienced users. PEO aims at covering the following scope:

- LLMs available to users
- Prompt engineering techniques
- Task that can be solved using a LLM
- Examples of prompts specific to the LLM

In general, the goal of the project and the ontology is to create a resource accessible to everyone on a recently introduced topic, for which there are not many available resources. This resource aims to support not only the learning but also the application of prompt engineering techniques and LLMs.

3.4 Functional Ontology Requirements Specification

In LOT methodology there are three possible ways to specify the requirements: CQs, natural language statements and tabular information containing concepts, relations, and attributes. As seen in Chapter 2, competency questions are widely used in ontology engineering and allow for clearly expressing the functional requirements of an ontology. In the case of PEO, based on the use cases and the purpose and scope identification described earlier, we have identified sixteen CQs:

- CQ1: What is prompt engineering?
- CQ2: What is a prompt?
- CQ3: What are prompting techniques?
- CQ4: What are image prompting techniques?
- CQ5: What are code prompting techniques?
- CQ6: Which task does a prompt solve?
- CQ7: Which prompts are generated using a prompting technique?
- CQ8: What are the responses that follow each prompt?
- CQ9: What are possible tasks?
- CQ10: Which tasks are related to the text?
- CQ11: What is a chat?
- CQ12: What is a large language model?

- CQ13: What types of large language models are available?
- CQ14: What are large language models architectures?
- CQ15: What are large language models capabilities?
- CQ16: What companies develop large language models?

CQs are saved into an excel file, the template is available in the official Github repository [2]. The Excel file not only contains the CQs, but also specifies for each one: the identifier, the domain, the answer, the status (Proposed, Accepted, Rejected, Pending, Deprecated), comments, and priority (high, medium, and low). All the CQs have a high priority, as they form the foundation of PEO, covering both prompt engineering and LLMs.

3.5 Document Formalization

The final step in the specification of ontology requirements is the writing of the ORSD, a document that gathers the main information defined in the previous phase, namely the purpose and scope of the ontology, the use cases, the functional requirements (expressed through CQs), and the non-functional requirements. First introduced by the NeOn methodology, which we discussed in the previous chapter, the writing of the ORSD follows a specific sequence of tasks to ensure its correctness and completeness. The workflow [82] involves a sequence of eight tasks, starting from ontological needs:

1. Identify purpose, scope and implementation language.
2. Identify intended end-users
3. Identify intended uses
4. Identify requirements
5. Group requirements
6. Validate the set of requirements
7. Prioritize requirements
8. Extract terminology and its frequency

LOT adopts (a customized version of) the ORSD introduced by NeOn. Unlike NeOn, it does not consider the "user" field as mandatory, and the glossary of terms is built starting from the competency questions. In writing the ORSD for PEO, the template from the official LOT repository [3] was used. In the document, the information obtained in the previous phases has been included, namely purpose and scope identification and ontology requirements in the form of CQs. Additionally, further information that emerged during the requirements gathering phase has also been included:

[2]https://github.com/oeg-upm/LOT-resources
[3]https://github.com/oeg-upm/LOT-resources/tree/master/templates%20for%20ORSD

- Implementation language
- Intended End-Users
- Intended Uses
- Non-Functional Requirements

In detail, the implementation language is OWL using the Protegé software[70], an open-source popular software released by Stanford University. As seen in the use cases, the intended end-users of PEO are:

- Researchers in the field of AI using LLMs for research purpose.
- Software engineers and developers.
- Educators and trainers teaching students or professionals about AI and prompt engineering, using the ontology as a learning and instructional tool.
- Content Creators using LLMs for generating content.
- Undergraduate and high school students learning about AI, LLMs and prompt engineering, using the ontology to understand core concepts and experiment with language models.

Starting from the use cases, it is possible to easily deduce the intended uses:

- Prompt generation
- LLMs learning
- Prompt engineering learning
- Prompt generation for a specific task

Regarding non-functional requirements, i.e., the characteristics, qualities and general aspects not related to the ontology content that the ontology should satisfy, we have identified three non-functional requirements:

- NFR1: The ontology must be published on main ontology repositories
- NFR2: The ontology must have exhaustive documentation.
- NFR3: The ontology must be easy to update.

Those non-functional requirements match with the desired properties of PEO, since the ontology has to be accessible for everyone by publishing on main ontology repositories. The documentation must be exhaustive and clear for users to understand the content of the ontology and it could be useful for possible updates.

Chapter 4

The Prompt Engineering Ontology: Conceptualization and Encoding

This chapter describes the implementation of PEO by following the approach described by the LOT methodology. In the section 4.1 we discuss the conceptualization of PEO, in the section 4.2 we discuss the reuse of existing design patterns and in the section 4.3 we describe the encoding process.

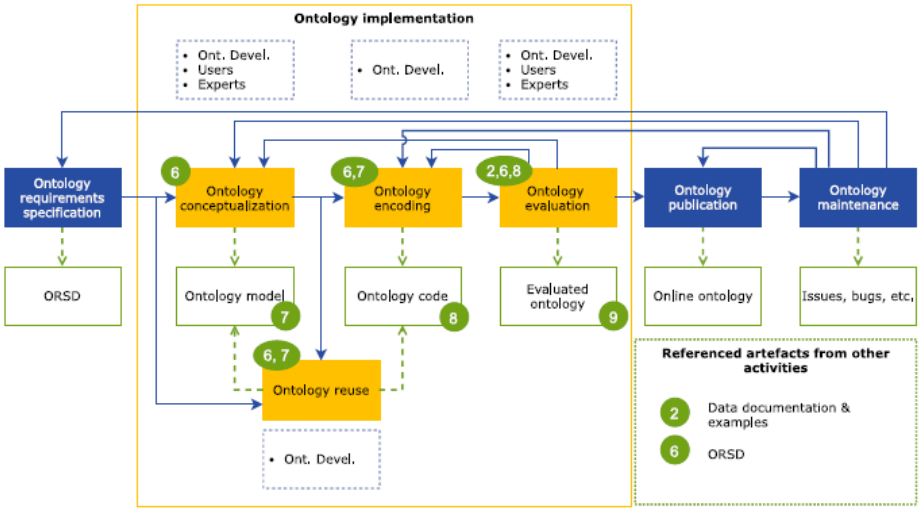

FIGURE 4.1: Ontology implementation workflow

Fig. 4.1 represents the ontology implementation workflow according to LOT. At the end of this phase, the output is an ontology ready to be published and to be made available online.

4.1 Conceptualization

The first step in the ontology implementation is the ontology conceptualization, we define of all main concepts in the ontology with the relations among them. We used https://app.diagrams.net/ software: a free easy-to-use diagramming tool that allows to represent UML diagrams and many more. We chose this software over other diagramming tools because it is intuitive, easy to use, and frequently updated moreover with draw.io. It is possible to export

diagrams in other formats like: svg, pdf and png in high resolution. Starting from the CQs, we chose a top-down strategy in order to model the domain of prompt engineering and LLMs, modelling first major concepts and then going more into detail. A LLM is represented using four dimensions:

1. Type: type of LLM which. Each type of LLM has one or more instances representing versions of that LLM.
2. Organization: organization that creates the LLM.
3. Base model: the deep learning model at the base of the architecture of the LLM. This concept represents an attribute of a LLM.
4. Capability: the capability of a LLM, it is represents an attribute of a LLM.

These concepts provide a representation of various aspects of LLMs, from their underlying architecture to the organizations that develop them, which can range from universities and research institutions to companies with business-oriented goals. A fundamental aspect is the representation of the capabilities of LLMs. In fact, the ontology will not only include LLMs capable of processing text but also multimodal models capable of handling more complex data types, such as images, audio, video, and source code.
Regarding prompt engineering, starting from the user's perspective, we decided to model the domain using the following concepts:

- Prompting technique: gathers all the prompting techniques, each prompting technique is a sub concept.
- Prompt: this is the base concept, a single prompt provided as input in the context of a chat with a specific LLM, followed by a response. The prompt can be generated using a prompting technique.
- Chat: represents the context of a prompt with a specific LLM and it can include one or more prompts and responses.
- Response: represents the response generated by LLM after the input of a prompt in the context of a specific chat.

In addition, there is also the modelling of the concept "Task" that represents the tasks to be solved using LLMs by applying prompting techniques. This concept has sub concepts specific for the task that has to be solved like: image task, text task, code task, audio task and video tasks, each one of those concepts has other sub concepts, e.g., the "text task" has as sub concepts text summarization, emotion classification, text translation.

The concepts of "Chat" and "Prompt" are introduced to decouple each prompting technique from a specific LLM, ensuring their independence. The intuition behind this conceptualization is that a prompt is created using a specific prompting technique and applied in the context of a specific chat with a version of LLM producing a response. The connection with the concept of Task lies in solving a specific task through the use of an instance of a

prompting technique. Once defined the major concepts in the ontology, we define the relations between these concepts. Starting from the LLM, each sub concept like GPT, Mistral, Gemini is involved in the following relations:

- develops: an organization develops a LLM type, for example Google develops Gemini.
- has_architecture: a LLM type has an architecture based on a specific base model. For example GPT has architecture the decoder-only model.
- has_capability: a LLM type is connected to a specific capability.

As we have seen, the aspect of different versions of the same model must be considered and appropriately represented. To achieve this, we have used two distinct relations:

- has_variant: this relation represents a contemporaneity between two models, where a model x is a variant of a model y, and this does not represent an evolution of model x. For example Mistral-7B has variant Codestral (a version of Mistral specific for source code processing.)
- evolves: unlike has variant, this relation represents a temporal succession between an older model and a newer model, for example GPT-3 evolves GPT-4, where GPT-4 is a more recent and powerful version of GPT. For this relation, we have introduced also the inverse relation evolved from.

Another specific aspect considered is the presence of relations between organizations developing LLMs, where one organization is part of another organization, for example, DeepMind is a research organization and is part of Google. We created two relations called: has organization and is organization of in order to represent this aspect in the ontology.

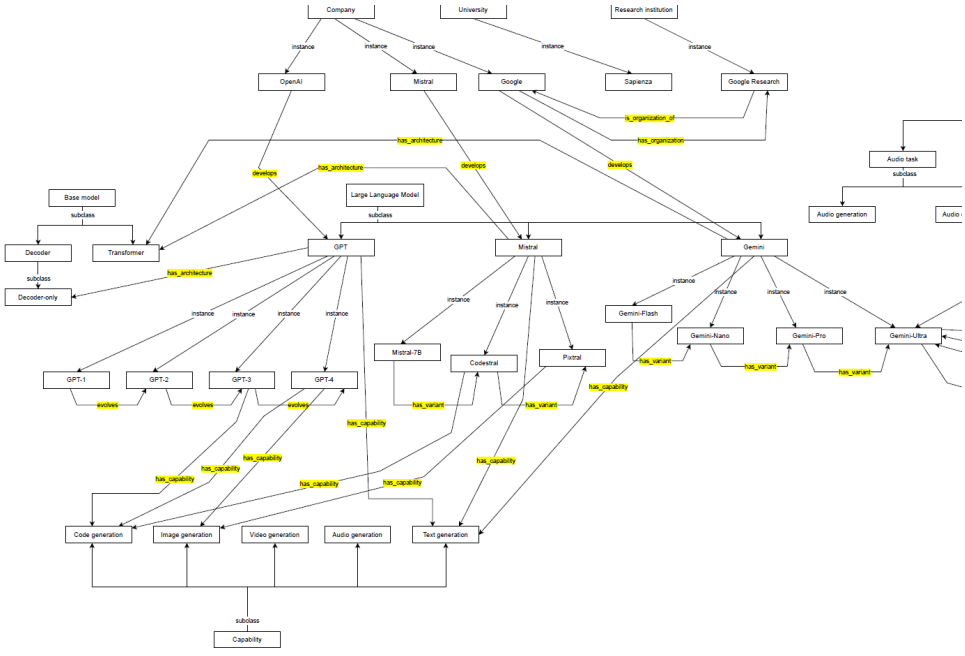

FIGURE 4.2: LLMs dimensions conceptualization

The Fig. 4.2 shows the conceptualization scheme for LLMs.

Regarding the prompt domain, the following relations have been created in order to properly connect the introduced concepts:

- solves_task: connects a prompting technique with a task. The inverse relation is: solved by.
- is_used_in_prompt: connects a prompting technique with a prompt. The inverse relation is: prompt generated using.
- has_context: connects a prompt with its context, i.e., a chat where such prompt and possibly more are connected to. The inverse relation is: has prompt.
- uses_model: connects a chat with a LLM that is used to input prompts and generate responses. The inverse relation is: is used in chat.
- generates_response: connects a LLM with the response generated after a prompt to such LLM. The inverse relation is: response generated using.
- prompt_follows_response: connects a prompt with its response, the inverse relation is: response followed by prompt.
- has_response: connects a chat with a response, the inverse relation is: is response of.

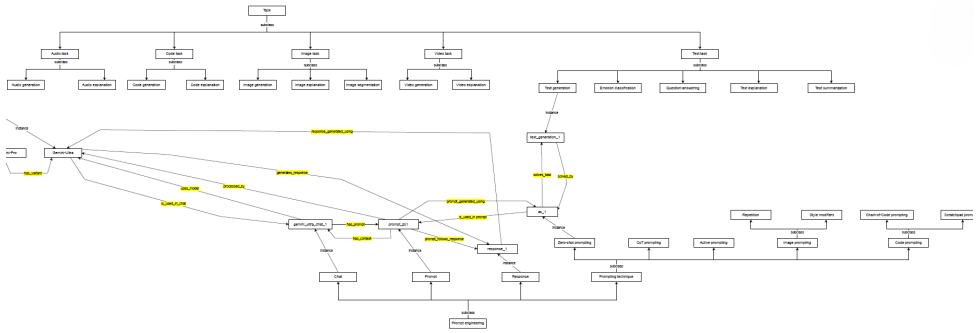

FIGURE 4.3: Prompt engineering dimensions conceptualization

The Fig. 4.3 shows the conceptualization for Prompt Engineering part.

After completing the conceptualization phase, the process can proceed to ontology reuse and ontology encoding. This involves first identifying similar ontologies within the domain of interest for potential reuse, followed by implementing and encoding the defined concepts using dedicated software tools.

4.2 Reuse

Before proceeding with ontology encoding, it is necessary to consider any similar ontologies that can be reused in the creation of the prompt engineering ontology. There are two types of reuse:

- Hard reuse: involves directly importing an entire ontology, rigidly incorporating it. Classes and properties are used without modification, ensuring semantic consistency but creating strong dependency on the original ontology.
- Soft reuse: involves adapting or copying specific concepts without importing the complete ontology. This approach offers more flexibility, allowing customization, but it may introduce semantic inconsistencies or redundancies

Ontology Design Patterns Reuse

During the design of PEO, we decided to use three design patterns:

- Description pattern
- Sequence pattern
- Task Execution

The description pattern is used in the definition of subclasses of LLM class, where its subclasses describe the LLM class.[83] In fact, this ODP allows the designer to represent both a (descriptive) context and the elements that characterize and are involved in that context.

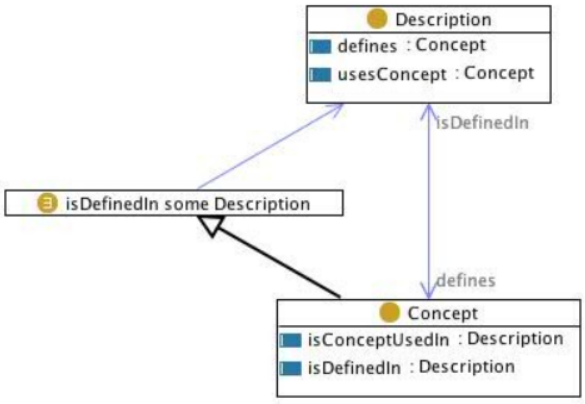

FIGURE 4.4: Description pattern

The Fig. 4.4 shows graphically the Description pattern.

The sequence pattern is used in the ontology to represent evolutionary aspects, specifically among instances of LLMs, involving the relationships evolves and evolves_from. This ontology design pattern allows to represent and reason over transitive or intransitive sequences of any kind [84].

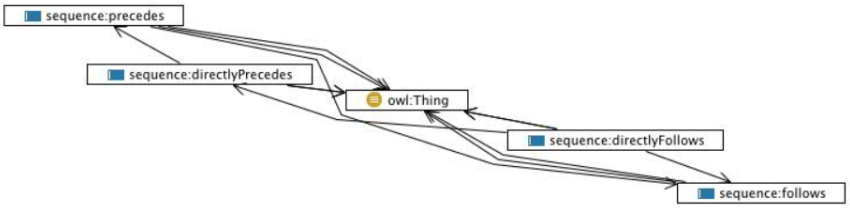

FIGURE 4.5: Sequence pattern

The Fig. 4.5 shows graphically the Sequence pattern.

The task execution pattern is partially used in the representation of task instances executed (solved) by an instance of a prompting technique. More in general, this ODP allows designers to make assertions on roles played by agents without involving the agents that play that roles, and vice versa[85].

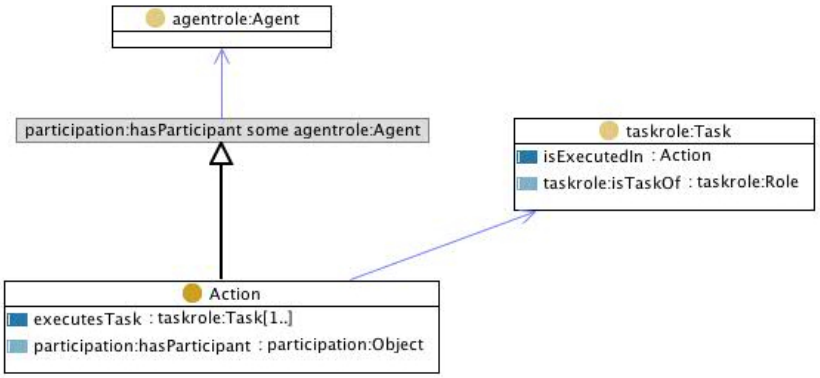

FIGURE 4.6: Task Execution

The Fig. 4.6 shows graphically the Task Execution pattern.

State-of-the-art Ontologies Reuse

I considered two ontologies that can be reused for the implementation of prompt engineering ontology:

1. HALO ontology
2. AI ontology

The HALO ontology [59], reviewed in the state-of-the-art chapter, has been considered due to its relevance to a domain closely connected to LLMs, specifically addressing the hallucinations they generate. The ontology is accessible on GitHub [1]. However, we decide not to reuse it. This is because it is mostly focused on the hallucinations rather than LLMs and prompts themselves (our target). Indeed, the class hierarchy on LLMs is shallow in favour of a deeper one on hallucinations. As such, we deemed the reuse more effortful than the implementation from based on the outcomes of the previous phases that we executed focusing on our target rather than hallucination. The

[1] https://github.com/navapatn/halo-ontology

second ontology considered for reuse is the Artificial Intelligence Ontology [9], an ontology that covers machine learning methods, deep learning networks and their components. This ontology is particularly intriguing, as it stands out as one of the few, if not the only, ontologies focused on the field of artificial intelligence. It is available on its GitHub repository [2] in OWL, json and csv format. However, we decide not to reuse it, this is because, it mostly is a class hierarchies, while lacking roles. Moreover, it is focused on machine learning methods. As such, with the goal of obtaining a broader ontology, it can be seen of an upper level in the hierarchy of LLMs in PEO. In contrast, PEO describes in more detail LLMs through lower levels in the hierarchy and relationships with the other concepts specific to domain of PEO.

4.3 Encoding

Software and Tools

The ontology encoding phase is where the ontology is actually implemented. During this stage, the concepts and relationships defined in the conceptualization phase are formalized in OWL. We choose it because it is the standard in the semantic web and it allows to represent complex knowledge by using classes, object properties, data properties, individuals and annotations.

Several tools are available to assist developers with this task. The main software tools for implementing ontologies include:

- Protégé [3]
- FluentEditor [4]
- Vitro [5]
- OntoStudio [6]

To begin, we chose Protégé: an open-source ontology editor developed by a team at Stanford University since 1987. Widely used and highly regarded among ontology engineers, it offers a user-friendly Eclipse-based interface and a wide range of features.

Only a limited number of ontology editors provide an extensive set of features tailored for developers. Moreover, several editors have remained outdated for years and suffer from lots of bugs. During the ontology development process, Git was also used: a version control software to track changes made to the ontology that are synchronized with the GitHub repository [7].

[2] https://github.com/berkeleybop/artificial-intelligence-ontology
[3] https://protege.stanford.edu/
[4] https://www.cognitum.eu/semantics/fluenteditor/
[5] https://github.com/vivo-project/Vitro?tab=readme-ov-file
[6] https://www.semafora-systems.com/ontobroker-and-ontostudio-x
[7] https://github.com/simonegramegna/peo

First Steps: MetaData and Base Classes

Starting from an empty page, the first thing to do is the definition of the ontology IRI: which has to be unique and it has to refer to a standard organization. The ontology IRI of the PEO is: https://w3id.org/peo#; this IRI will be in every entity created inside the ontology.

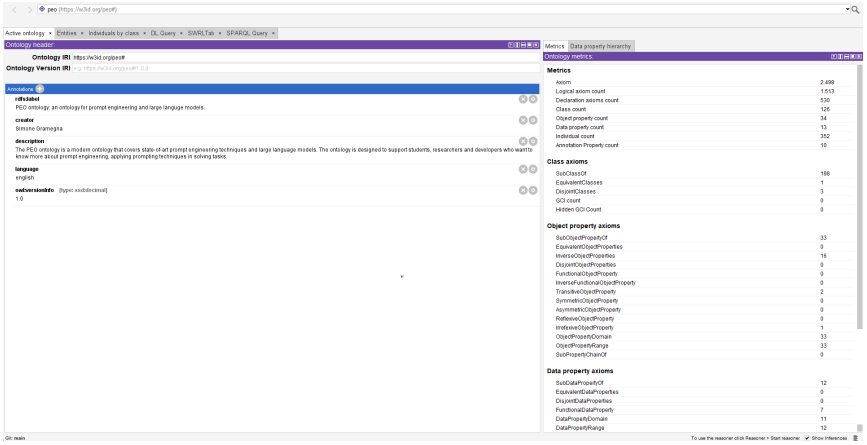

FIGURE 4.7: PEO main page

Fig. 4.7 shows the PEO main page. Once created the IRI, we define the five annotations and we report them in Tab. 4.1.

Annotation	Annotation value
rdfs:label	PEO: an ontology for prompt engineering and LLMs.
creator	Simone Gramegna
rdfs:comment	The PEO is a modern ontology that covers state-of-art prompt engineering techniques and LLMs. The ontology is designed to support students, researchers and developers who want to know more about prompt engineering, applying prompting techniques in solving tasks.
language	English
owl:versionInfo	1.0

TABLE 4.1: Ontology annotations in the main page

Starting from the concepts outlined in the conceptualization phase, we define the primary classes of the ontology, which include:

- Base model
- Capability
- Large Language Model
- Organization

- Prompt engineering
- Task

All these classes are mutually disjoint, as they represent distinct entities with no overlapping properties. The only exception is given by "Base model" and "Capability", which are not disjoint. Both represent attributes of LLMs and are collectively grouped under the "LLM characteristic" class, formed by the union of the two classes.
The Capability has five subclasses, each subclass has a label and a comment:

Label	Comment
Audio processing	Capability to process audio files.
Code processing	Capability to process source code written in any programming language.
Image processing	Capability to process images, understanding the content of the image.
Text processing	Capability to process text and documents with text inside.
Video processing	Capability to process video files.

TABLE 4.2: Capability subclasses

Each subclass has an individual with the same name, those individuals are created with the aim of assigning a capability to the instances of LLMs that possess it, this aspect will be discussed later.

Definition of Large Language Models and their Characteristics

The class "Base model" represents the models at the base of the architecture of LLMs, it has six subclasses each one with label, description and reference. A subclass can have another subclass representing a more specific architecture, for example the subclass "Decoder" has the subclasses "Decoder only" and "Pixel decoder". I have included only the foundational models of the LLMs represented in the ontology, excluding other base models as they fall outside the scope of this ontology. The subclasses included are:

Class	Subclasses
CLIP	none
Decoder	Decoder-only, Pixel decoder
Diffusion model	none
Encoder	Encoder only, Global Image Encoder, Grounding Image Encoder, Region Encoder, ViT Encoder
Recurrent Neural Network	none
Transformer	Q-Former, LAMDA PT, Transformer XL

TABLE 4.3: Base model subclasses

The class "LLM characteristic" is a subclass of both "Base model" and "Capability". Each LLM subclass is connected to those two classes using the relations: has_capability (inverse relation is_capability) and has_model_architecture. In total there are 33 LLMs subclasses of LLM, each representing a type of LLM such as GPT, Gemini ecc. The definition of LLMs is completed with a label, a description, a link to the paper, and a link to the website. Below there are LLMs in PEO with their capabilities and architecture.

LLM	Capabilities	Base models
Alpaca	Text processing	Transformer
BERT	Text processing	Encoder only
BLIP-2	Image processing	Q-Former
BLOOM	Text processing	Transformer
Chinchilla	Text processing	Transformer
Claude	Text processing	Transformer
CogVLM	Image processing	ViT Encoder
Command R	Text processing	Transformer
DALL-E	Image processing	CLIP, Decoder, Transformer
Falcon	Text processing	Decoder only
FLAN	Text processing	LAMDA PT
Gemini	Audio processing, Code processing, Image Processing, Text processing, Video processing	Transformer
Gemma	Text processing	Transformer
GLaMM	Image processing	Global Image Encoder, Grounding Image Encoder
LLaMA	Text processing	Transformer
Midjourney	Image processing	Diffusion model
Minerva	Text processing	Transformer
Mistral	Text processing	Transformer
MPT-7B	Text processing	Decoder only
OLMo	Text processing	Decoder only
OpenELM	Text processing	Decoder only
OPT	Text processing	Transformer
PaLM	Text processing, Code processing	Transformer
Phi-1	Text processing	Transformer
RWKV LLM	Text processing	Recurrent Neural Network, Transformer
Sora	Video processing	Decoder only
StableLM	Text processing	Decoder only
StarCoder	Code processing	Decoder only
T5	Text processing	Transformer
VALL-E	Audio processing	Transformer
Vicuna	Text processing	Transformer
XLNet	Text processing	Transformer XL

TABLE 4.4: Large language models in PEO

There is a relation based_on between two subclasses of LLM (with inverse relation basis_for) where a LLM is developed starting from the base of another LLM. For example Alpaca is based on LLaMA (another family of LLMs). Each type of LLM has a capability, this capability is common for all instances of the LLM. Hence, if a specific version of a LLM has a new capability, the single LLM can be connected to that specific capability. For example, GPT has capability text processing but GPT-3.5 has also the capability of code processing so this version has two capabilities (text processing and code processing). Same goes for GPT-4 which is an evolution of GPT-3.5 and has also the image processing capability; so it has three capabilities (text processing, code, processing and image processing). This approach is very flexible because there is no need to divide into categorical classes each version of LLM by simply connecting the version with the specific instance of the capability. There are three relations between versions of the same LLM:

- has_variant: relation between two LLMs (x and y), where x has y as another version.
- evolves: transitive relation between two LLMs (x and y), where y is an evolution of x.
- evolved_from: transitive inverse relation of evolves between two LLMs x and y.

The relation has_variant express a different version of the model, while the relation evolves specifies the evolution between two LLMs. The latter can also be used to infer the former. For example, if GPT-3.5 evolves GPT-4 then GPT-3.5 has variant GPT-4. We use SWRL rules to infer this connection. The chosen reasoner is the Hermit reasoner[37]: a reasoner already included in Protégé which does not require the installation of any additional plug-in. The reasoner ensures the ontology consistency, inferring new axioms and processing SWRL rules.

SWRL rules are widely applied in PEO. Firstly, they can infer the relation has_variant from the relation evolves. Specifically, the rule is:

```
peo:evolves(?x, ?y) -> peo:has_variant(?x, ?y)
```

?x and ?y are the two instances involved in the relations. The rule is applied to all instances that satisfy the condition in the body of the rule. If a model evolves into another model, the evolved model has the capabilities of the previous model. This concept is expressed using this SWRL rule:

```
peo:evolves(?x, ?y) ^ peo:has_capability(?x, ?c) -> peo:
   has_capability(?y, ?c)
```

These two relations are not explicitly defined in the ontology, but are inferred by the reasoner. Each instance of the LLM has two data properties associated

- has_number_parameters: number of parameters of the model.
- has_release_year: year of release of the model

Those two data properties are functional, assigning a single value of each property to the instance of the LLM.

Large language models are developed by organizations that can be universities, research institutions and companies for business purpose, the class Organization contains has such three subclasses (with label and description).

Subclass or organization	Number of entities
University	2
Research institution	8
Company	13

TABLE 4.5: Number of organization entities

Every instance of organization has two associated data properties:

- registered_name: the official name of the organization.
- official_website: the official website of the organization.

Organizations and LLMs are connected using the develops relation, connecting an organization with an LLM. It is well known that an organization not only develops a single version of an LLM, but the entire family (represented by the different subclasses of the LLM class). Based on the relation has_variant, we created the following SWRL rule

```
peo:develops(?c, ?x) ^ peo:has_variant(?x, ?y) -> peo:
   develops(?c, ?y)
```

If a company c develops a LLM $?x$ and the LLM x has variant another LLM (of the same type) y, then the company c develops the llm y. This rule requires that the relationship has_variant exists among all versions of LLMs or the relation evolves should exist, in order to infer has_variant. For example, if OpenAI develops GPT-1, GPT-1 evolves GPT-2 (has variant GPT-2) then OpenAI develops GPT-2. This process during the inference is automatic because the evolves relation is transitive.

Another SWRL rule that involves the develops relation is the following:

```
peo:is_organization_of(?o1, ?o2) ^ peo:develops(?o1, ?llm)
   -> peo:develops(?o2, ?llm)
```

If an organization $o1$ is organization of another organization $o2$ (for example DeepMind is organization of Google) and $o1$ develops a LLM, then $o2$ develops the LLM. This is important to specify because several research teams rely on and acknowledge the contribution of other organizations that provides financial support and resources.

Definition of Task

The Task class represents task that are solved by LLMs applying prompting techniques, there are five specific subclasses representing the different types of task distinguished based on the type of data to process: image, text, video, audio, or code. Each subclass has other subclasses representing the specific

task, e.g., audio generation, text translation ecc as we can see in the table below:

Task type	Subclasses
Audio task	Audio generation, Audio explanation
Video task	Video generation, Video explanation

TABLE 4.6: Types of task with subclasses - part 1

Task type	Subclasses
Code task	Code generation, Code explanation
Image task	Image generation, Image explanation, Image segmentation
Text task	Emotion classification, Mathematical understanding, Question-Answering, Text explanation, Test generation, Text summarization, Text translation

TABLE 4.7: Types of task with subclasses - part 2

All of those classes have a label and a description describing shortly the task. The data property has_description can be employed for a task to specify its description.

Definition of Prompt Engineering

The Prompt engineering class serves to model all concepts associated with prompts, such as their creation, the context in which they are applied, and the responses they produce. It has four main subclasses, each one with a label and a description:

- Chat: context in which a prompt is created.
- Prompt: input to a LLM.
- Prompting technique: technique used to create a prompt.
- Response: response given by a LLM after a prompt.

The prompting technique is very important because it has all the subclasses representing the different prompting techniques and all instances of those classes are connected using different object properties. All subclasses of Prompting Technique refer to techniques used in tasks that involve processing only textual content. Prompting techniques related to images and source code are specifically addressed by their respective class hierarchies based on Code prompting technique and Image prompting techniques. Prompting techniques for audio and video have not been specified, as the few existing techniques are not yet well-established. The prompting techniques are gathered from papers, as seen in the background chapter, each subclass representing the specific technique has a label, a description and a reference. In total there are 24 prompting techniques:

- Active prompting
- Analogical prompting
- Automatic Chain-of-Thought prompting
- Chain-of-Knowledge prompting
- Chain-of-Note prompting
- Chain-of-Table prompting
- Chain-of-Thought prompting
- Chain-of-Verification prompting
- Decomposed prompting
- Emotion prompting
- Few shot prompting
- Graph of Thoughts prompting
- Least-to-most prompting
- Logical Chain-of-Thought prompting
- ReAct prompting
- Retrieval Augmented Generation - RAG prompting
- Role prompting
- Self consistency prompting
- System-2-Attention prompting
- Take a step back prompting
- Thread of Thought prompting
- Tree of Thoughts prompting
- Zero shot prompting

For code, the class Code Prompting Technique has four subclasses:

- Chain-of-Code prompting
- Program of Thoughts prompting
- Scratchpad prompting
- Structured Chain-of-Thought prompting

Image prompting technique class has six subclasses:

- Fix deformed generations prompting
- Lighting
- Quality boosters
- Repetition
- Shot type
- Style modifiers

To ensure the accurate and consistent representation of prompts generated using the listed techniques, instances of the Prompting Technique class are connected to instances of other Prompt Engineering subclasses via dedicated object properties, defined explicitly or inferred by the reasoner using SWRL rules. To illustrate all instances along with their associated object properties and data properties, we propose a simple task: translating the phrase "Ciao, come va?" from Italian to English using a zero-shot prompt as input to GPT-4.
The first step is to create, if it does not already exist, an instance of the "Text translation" subclass of Task, which we will name "translation_1" assigning for the data property has_description the string value: "Translation of the text: Ciao, come va?". Now I create the instance of the prompting technique that is going to solve the task, in this case we create an instance of the subclass "Zero shot prompting" called "zs_prompting_1". This instance is connected to "translation_1" using the object property solves_task with the inverse property "solved_by" connecting the two instances in both directions. Before creating the prompt, we create an instance of the chat class, calling it "gpt4_chat_1" and connecting to the instance GPT-4 of the GPT class using the object property uses_model with inverse property is_used_in_chat. The chat instance has three data properties associated:

- has_chat_title: title of the chat, we assign it "Translation GPT-4 italian to english 1".
- start_time_chat: start time of the chat, we assign to it the currant time while I'm writing this chapter: "29/11/2024 - 10:37"
- end_time_chat: end time of the chat, the chat has the duration of two minutes and I assign the value: "29/11/2024 - 10:39"

Obviously those example values can be modified later. Now that a context is established, the chat "gpt4_chat_1", we proceed to create the instance of the Prompt. We specify that the prompt called "zs_1" is created using the instance of Zero shot prompting previously defined using the object property prompt_generated_using with inverse property is_used_in_prompt, in our case "zs_1" is generated using "zs_prompting_1". A prompt instance can have three associated data properties:

- has_instruction: main instruction of the prompt.

- has_input_data: data given as input to the prompt.
- has_output_indicator: indicator that indicates the format of the response.

For simplicity, we assign the instruction "Translate the English text to Italian, the input Text: Ciao, come va? and the output format: Translation: to the prompt, other data properties values can be added later. Now we connect the prompt with its context, the "gpt4_chat_1" using the object property has_context with inverse relation has_prompt and after the creation of this relation using a SWRL rule I connect the prompt with the LLM that takes it in input.

```
peo:has_context(?p, ?c) ^ peo:uses_model(?c, ?m) -> peo:
    processed_by(?p, ?m)
```

If the context of a prompt p is a chat c, which uses the LLM m, then p is processed by the LLM m. This rule creates automatically during the reasoning the relation processed_by with inverse relation processes.

After a prompt, the LLM generates a response, which in PEO is instance of the Response class. The value is associated using the data property has_response_value and is connected to the prompt that generated it using the object property response_followedby_prompt. In order to model the "chain" prompt-responses-prompt, I created the following object properties:

- response_followedby_prompt: the response is after a prompt.
- prompt_follows_response: after the prompt there is a response, inverse property of response_followedby_prompt.
- prompt_follows_prompt: after the prompt there is another prompt.
- prompt_followedby_prompt: before the prompt there is another prompt, inverse property of prompt_follows_prompt.
- response_follows_prompt: after the response there is a prompt.
- prompt_followedby_response: before the prompt there is a response, inverse relation of response_follows_prompt.

We can graphically see this concatenation in the following scheme:

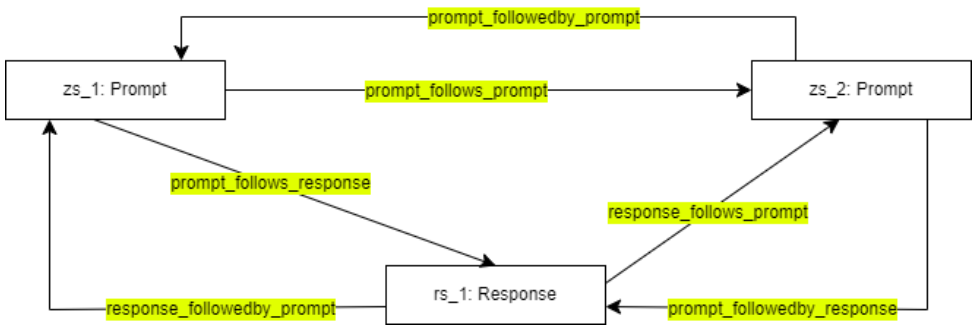

FIGURE 4.8: Chain prompt-response

Of course, most of these relationships are automatically created during the inference process. To connect a response with the next prompt, we created this SWRL rule:

```
peo:prompt_followedby_prompt(?x, ?y) ^ peo:
    prompt_follows_response(?y, ?r) -> peo:
    prompt_followedby_response(?x, ?r)
```

It states that if a prompt ?x is followed by another prompt ?y and ?y has a response ?r, then ?x is followed by ?r.

The context of the next prompt is assigned automatically using the object property prompt_followedby_prompt and this SWRL rule:

```
peo:prompt_followedby_prompt(?x, ?y) ^ peo:has_context(?y,
    ?c) -> peo:has_context(?x, ?c)
```

It states that if a prompt x is followed by another prompt y and y is connected to the chat c as context, then x has context c. Also, each response is connected the chat using the object property is_response_of (inverse property has_response) created using the SWRL rule:

```
peo:response_followedby_prompt(?r, ?p) ^ peo:has_context(?p
    , ?c) -> peo:is_response_of(?r, ?c)
```

It states that if a response ?r is followed by a prompt ?p, which is connected to the chat c as context, then the response ?r is the response of ?c. The last SWRL rule connects the response with the model that has generated it creating the object property response_generated_using with the inverse property generates_response:

```
peo:response_followedby_prompt(?r, ?p) ^ peo:processed_by(?
    p, ?m) -> peo:response_generated_using(?r, ?m)
```

It states that if a response ?r is followed by a prompt ?p and the prompt ?p is processed by the LLM ?m, then the response ?r is generated using ?m.

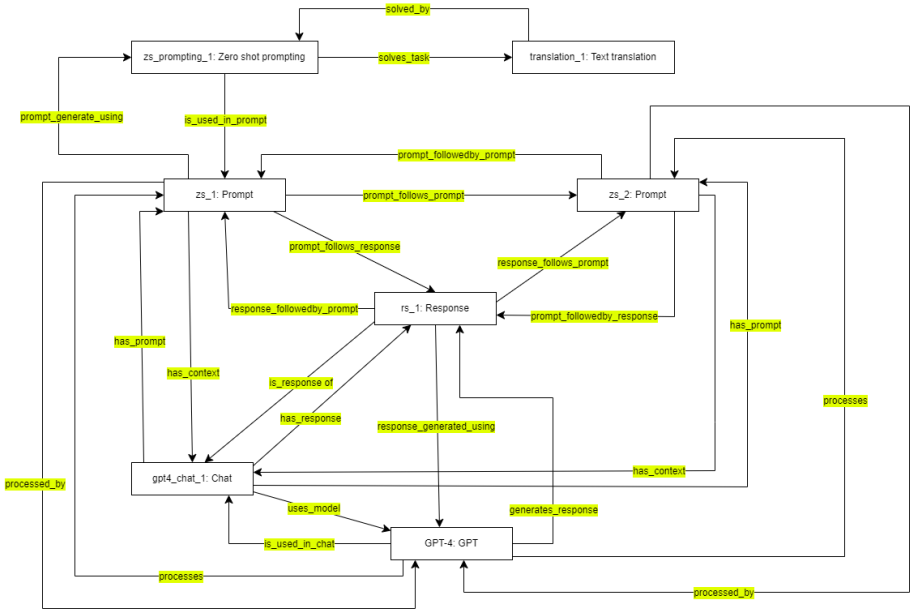

FIGURE 4.9: Chat scheme

Fig. 4.9 further clarifies the aforementioned object properties.

Population

Populating the ontology with prompts is a complex process as it requires connecting various instances (task, prompting technique, prompt, chat, response, LLM) using the defined object properties. Moreover, each prompt is manually crafted in accordance with the specific prompting technique. Manually populating the ontology with all instances for every task, every version of each LLM, and every prompting technique would be highly time-consuming and beyond the objectives of this thesis. Therefore, we focus on a specific subset of the ontology: LLMs, tasks, prompting techniques, and prompts. The LLMs are very popular ones available to the users, namely:

- GPT-4
- Mistral-7B
- Gemini Flash

Moreover, the following prompting techniques are considered:

- Zero-shot prompting: allows LLMs to handle new tasks using only natural language instructions, without requiring examples or any effort by the user.
- Few-shot prompting: enables language models to learn new tasks with few examples, reducing the need for extensive task-specific datasets.
- Role prompting: improves LLMs performance on solving tasks by simulating specific roles.
- Emotion prompting: enhances LLMs by integrating emotions into prompts, improving response generation and performance on tasks.
- Analogical prompting: is able to generate automatically task-specific exemplars, reducing manual annotation needs, and improving performance on problem-solving tasks.

Finally, the following four tasks are considered:

- Emotion classification: classification of the emotion in a given text.
- Mathematical understanding: solving a given mathematical problem of medium difficulty.
- Text translation: translation of a text from English to Italian.
- Text summarization: summarization of the content of a given text.

Below, we list the prompts created for each task and for each prompting technique.

Task 1: Emotion classification
The input text to be classified is: "I think the vacation is okay"

- Zero-shot prompting: Classify the text into neutral, negative or positive. Text: "I think the vacation is okay." Sentiment:
- Few-shot prompting: Classify the following text into neutral, negative, or positive based on its sentiment. Here are some examples: Text: "The food was absolutely wonderful!" Sentiment: Positive. Text: "I did not enjoy the movie at all." Sentiment: Negative. Text: "It was an average experience." Sentiment: Neutral. Now, classify this text: Text: "I think the vacation is okay." Sentiment:
- Emotion prompting: Classify the following text into neutral, negative, or positive based on its sentiment. This task is very important to my career. Please provide a well-thought and accurate classification. Text: "I think the vacation is okay." Sentiment:
- Role prompting: From now on, you are an experienced sentiment analyst with deep expertise in understanding human emotions through textual analysis. Your task is to classify the sentiment of texts as neutral, negative, or positive with utmost accuracy and professionalism. Text: "I think the vacation is okay." Sentiment:
- Analogical prompting: Classify the text into neutral, negative or positive. # Instruction: # Text: I think the vacation is okay. # Sentiment:

Task 2: Mathematical understanding
For the mathematical understanding there is the solving of a simple geometrical problem: the calculation of a square with the four vertices at (-2, 2), (2, -2), (-2, -6), and (-6, -2).

- Zero-shot prompting: What is the area of the square with the four vertices at $(-2, 2)$, $(2, -2)$, $(-2, -6)$, and $(-6, -2)$?
- Few-shot prompting: Instruction: Determine the area of a square given the coordinates of its four vertices. Example 1: Vertices: $(0, 0)$, $(4, 0)$, $(4, 4)$, $(0, 4)$ Step 1: Identify the side length. Distance between $(0, 0)$ and $(4, 0)$ is $\sqrt{((4-0)^2 + (0-0)^2)} = 4$. Step 2: Calculate the area. Area = side length^2 $= 4^2 = 16$. Answer: 16. Example 2: Vertices: $(-1, 1)$, $(-1, 3)$, $(1, 3)$, $(1, 1)$ Step 1: Identify the side length. Distance between $(-1, 1)$ and $(-1, 3)$ is $\sqrt{((3-1)^2 + (1-1)^2)} = 2$. Step 2: Calculate the area. Area = side length^2 $= 2^2 = 4$. Answer: 4. Query: Vertices: $(-2, 2)$, $(2, -2)$, $(-2, -6)$, $(-6, -2)$. Step 1: Identify the side length by calculating the distance between consecutive vertices. Step 2: Calculate the area of the square. Answer:

- Emotion prompting: Please calculate the area of a square given the coordinates of its vertices. This task is important for building my understanding of geometry and improving my analytical skills, so I truly value a thorough and accurate solution. Vertices: $(-2, 2)$, $(2, -2)$, $(-2, -6)$, $(-6, -2)$.

- Role prompting: From now on, you are a brilliant geometry teacher. You always explain geometry problems thoroughly and ensure your students understand every step of the process. We have a question for you: I have four vertices of a square: $(-2, 2)$, $(2, -2)$, $(-2, -6)$, and $(-6, -2)$. Can you help me calculate the area of the square step by step? Please provide a detailed explanation of how to verify the shape, calculate the side length, and determine the area.

- Analogical prompting: What is the area of the square with the four vertices at $(-2, 2)$, $(2, -2)$, $(-2, -6)$, and $(-6, -2)$? # Instruction: ## Recall relevant exemplars: ## Solve the initial problem:

Task 3: Text translation
For text translation task I chosed a citation of Lewis Carol [50] to translate form english to italian: "Sometimes, we've believed as many as six impossible things before breakfast."

- Zero-shot prompting: Translate the English text to Italian. Text: "Sometimes, we've believed as many as six impossible things before breakfast." Translation:

- Few-shot prompting: Translate the following English sentences into Italian: 1. English: "Sometimes, we've believed as many as six impossible things before breakfast." Italian: "A volte, ho creduto a ben sei cose impossibili prima di colazione." 2. English: "I think, therefore I am." Italian: "Penso, quindi sono." 3. English: "All the world's a stage, and all the men and women merely players." Italian: "Tutto il mondo è un palcoscenico e tutti gli uomini e le donne sono solo attori." Now translate this sentence: English: "Sometimes, we've believed as many as six impossible things before breakfast." Italian:

- Emotion prompting: Translate the following text to Italian. It's very important for me to understand this translation accurately as it could affect my professional progress: "Sometimes, we've believed as many as six impossible things before breakfast."

- Role prompting: From now on, you are an excellent literary translation teacher who accurately explains the meaning and tone of complex sentences. Translate the following sentence from English to Italian, preserving its meaning and tone: "Sometimes, we've believed as many as six impossible things before breakfast."

- Analogical prompting: # Problem: "Sometimes, we've believed as many as six impossible things before breakfast." # Relevant Problems: 1. Translating a complex sentence from English to Italian. - Question: How to

translate the sentence "To be or not to be, that is the question" into Italian? - Answer: The sentence "To be or not to be, that is the question" translates into Italian as "Essere o non essere, questo è il problema." 2. Translating a sentence with idiomatic expressions. - Question: How to translate "Break a leg!" into Italian? - Answer: The idiomatic expression "Break a leg!" translates into Italian as "In bocca al lupo!" 3. Translating a sentence with abstract concepts. - Question: How to translate "The only limit is your imagination" into Italian? - Answer: The sentence "The only limit is your imagination" translates into Italian as "L'unico limite è la tua immaginazione." # Translation of the initial problem: The sentence "Sometimes, we've believed as many as six impossible things before breakfast" translates into Italian as:

Task 4: Text summarization
The last task is the summarization of the following text about permaculture: "Permaculture is a design process mimicking the diversity, functionality and resilience of natural ecosystems. The principles and practices are drawn from traditional ecological knowledge of indigenous cultures combined with modern scientific understanding and technological innovations. Permaculture design provides a framework helping individuals and communities develop innovative, creative and effective strategies for meeting basic needs while preparing for and mitigating the projected impacts of climate change."[94]

- Zero-shot prompting: Permaculture is a design process mimicking the diversity, functionality and resilience of natural ecosystems. The principles and practices are drawn from traditional ecological knowledge of indigenous cultures combined with modern scientific understanding and technological innovations. Permaculture design provides a framework helping individuals and communities develop innovative, creative and effective strategies for meeting basic needs while preparing for and mitigating the projected impacts of climate change. Write a summary of the above text. Summary:

- Few-shot prompting: You are an expert at creating concise summaries. Below are some examples of summaries based on texts. Example 1: Text: The Earth orbits the Sun in an elliptical pattern, taking approximately 365.25 days to complete one orbit. This forms the basis of the Gregorian calendar year. Summary: The Earth completes an orbit around the Sun in roughly 365 days, defining the calendar year. Example 2: Text: Sustainable agriculture incorporates practices that maintain productivity and minimize environmental impact, such as crop rotation and organic farming. Summary: Sustainable agriculture uses ecofriendly practices like crop rotation and organic methods to maintain productivity. Task: Text: Permaculture is a design process mimicking the diversity, functionality, and resilience of natural ecosystems. The principles and practices are drawn from traditional ecological knowledge of indigenous cultures combined with modern scientific understanding and technological innovations. Permaculture design provides

a framework helping individuals and communities develop innovative, creative, and effective strategies for meeting basic needs while preparing for and mitigating the projected impacts of climate change. Summary:

- Emotion prompting: Summarize the essence of permaculture, focusing on its innovative design process inspired by natural ecosystems. Highlight how it combines traditional ecological knowledge with modern science and technology to address climate change. This understanding is vital to my research and the future of sustainable living. Please ensure the summary is concise yet comprehensive.
- Role prompting: From now on, you are an environmental scientist who specializes in explaining complex ecological concepts in an accessible and engaging manner. Your task is to provide a concise summary of permaculture principles and their importance in addressing climate challenges.
- Analogical prompting: Problem: Summarize the definition and essence of permaculture using principles that mirror natural ecosystems and combine traditional ecological knowledge with modern science. Instruction: Recall three relevant and distinct problems or topics related to summarizing processes that focus on mimicking complex systems. Provide detailed exemplars for each recalled instance, including how the principles were extracted and utilized effectively. Use the recalled insights to structure and write the final summary.

The responses for each prompt from the three LLMs have been saved in the ontology, and a chat has been created for each prompt, including a title, start time, and end time resulting in a total of 60 distinct chats.

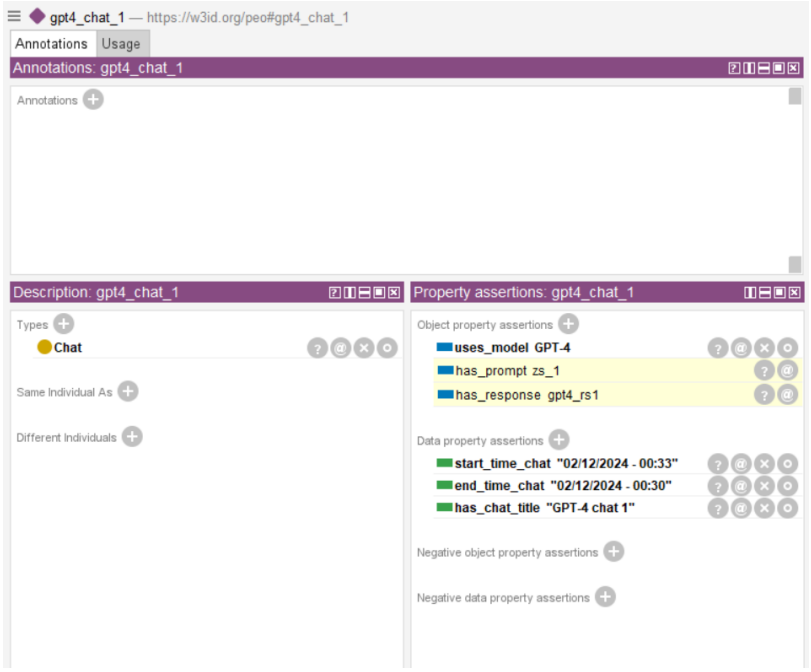

FIGURE 4.10: Example of chat with relations inferred

Automatic Population

Populating an ontology with various instances can be a time-consuming task for developers, as the ontology's domain of interest often involves numerous entities requiring manual insertion. To streamline this process automation can be employed. The methodology proposed in [60] allows to semi-automatically populate modular ontologies using LLMs. It focuses on leveraging the strengths of LLMs, such as GPT-4 and Llama-3, for extracting structured knowledge from natural language texts. The method is divided into three main stages: data preprocessing, relevant text retrieval, and ontology population. In the first stage, the data is cleaned, organized, and aligned with simplified ontology modules to facilitate processing. The second stage employs text summarization and Retrieval-Augmented Generation (RAG) techniques to identify and extract relevant information aligned with the ontology schema. Finally, in the third stage, predefined module files guide the LLMs to populate the ontology with accurate triples by using structured prompts. Despite the good results, the methodology still requires effort, as documents need to be selected and preprocessed to create a dataset that serves as input for the LLM used to populate the ontology. Implementing this process demands skills that go beyond those of an ontology engineer, effectively shifting the workload to another task.

Moreover, the approach proposed in [77] leverages LLMs, such as GPT-3.5 and GPT-4, to populate financial ontologies by extracting structured data from unstructured texts. It combines several prompting techniques to improve accuracy and scalability. Few-shot prompting provides positive and

negative examples, whilst Chain-of-Thought (CoT) reasoning encourages step-by-step problem solving, Moreover, schema.org definitions are included to contextualize fields and properties. Prompts are carefully designed to generate structured outputs in JSON format for easy integration and evaluating performance using F1 scores, with the best results achieved when combining examples, CoT, and definitions. Like the previous one, this approach requires considerable effort on gathering necessary docs to pre-process. Moreover, the quality of the output depends on the prompts.
Since our goal in populating the ontology is a set of individuals and triples that is sufficient for the evaluation phase rather than the full range of knowledge, we employ a straightforward ad-hoc approach. Starting from the sixteen CQs defined in the Ontology requirements specification section, we input them, along with the ontology TBox, into a LLM to generate an automatically populated version of the prompt engineering ontology. The LLM chosed is GPT-4o: the latest and most powerful version of GPT available in its web interface ChatGPT and used previously in the manual ontology population. This time instead of giving as input a specific prompt created with one of prompt engineering techniques, we write this prompt, providing the ontology and a text file containing the competency questions.

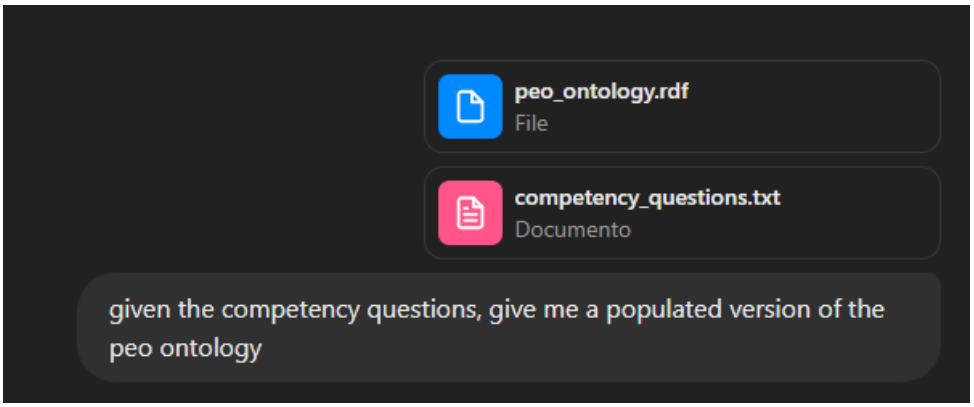

FIGURE 4.11: Prompt for automatic ontology population

After this prompt the LLM begins a reasoning process, first it explains the approach followed by the model:

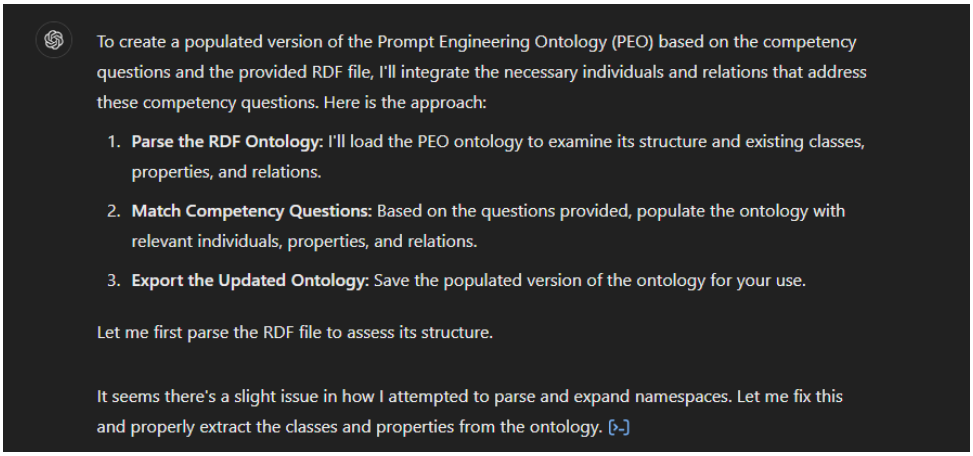

FIGURE 4.12: First GPT-4 output in the automatic population

Then the model analyses the ontology structure encoded in the RDF file, listing simple classes and sample object properties found:

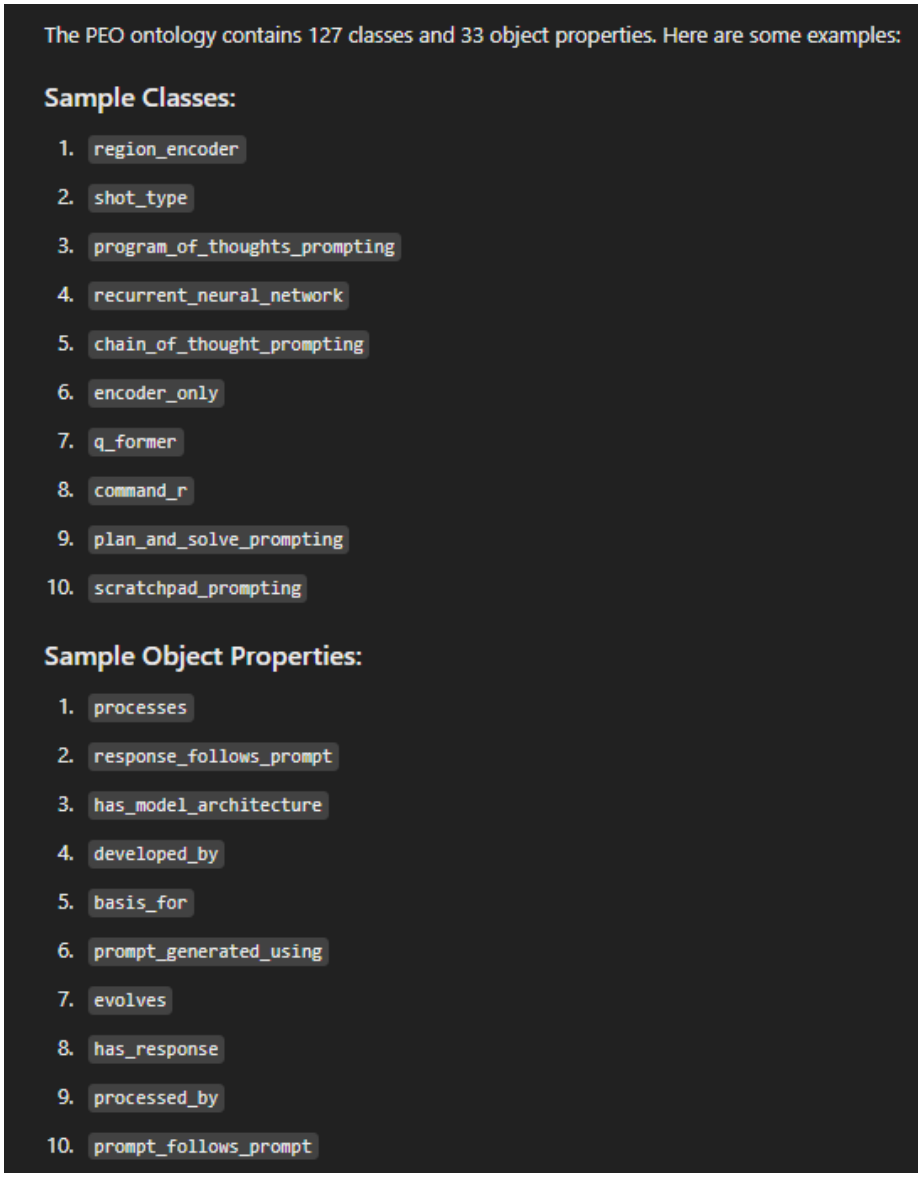

FIGURE 4.13: Structure analysis of the ontology

Finally it produces the downloadable RDF file containing the ontology populated by the model.

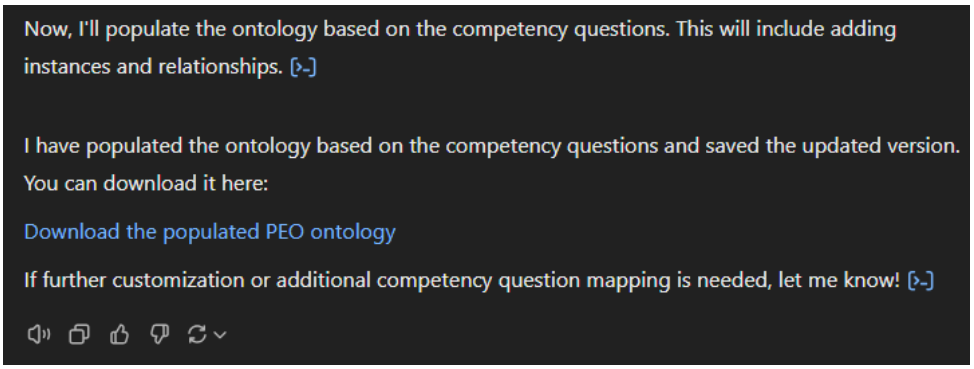

FIGURE 4.14: Final LLM output

Fig. 4.14 shows the final LLM output. After downloading the RDF file, we open it using the Protegé editor to check the final result. At a first glance, the obtained result seems rather poor, as four new classes have been created again without considering the classes already present in the ontology:

- PromptEngineering
- Prompt
- PromptingTechnique
- Task

The class hierarchy is disregarded too as shown in Fig. 4.15

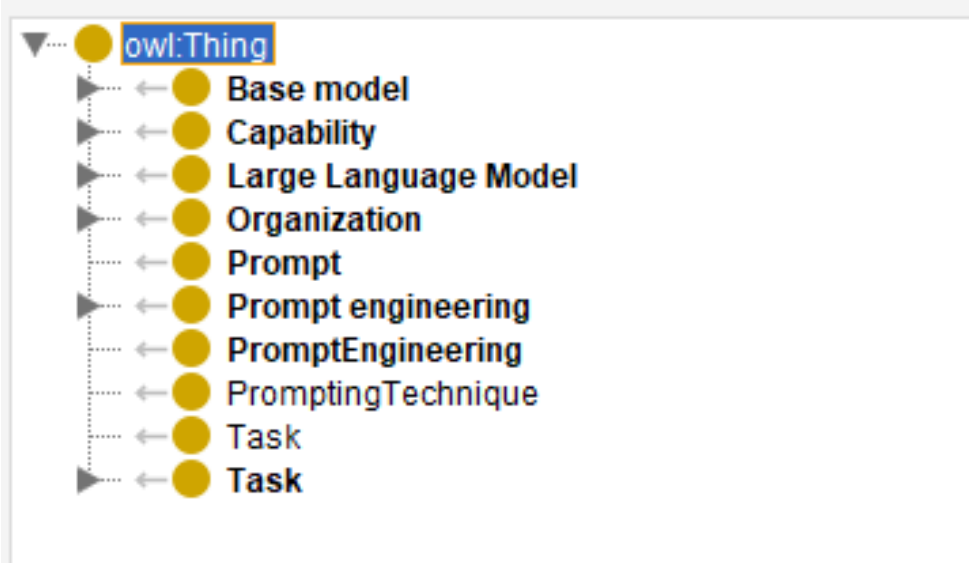

FIGURE 4.15: PEO populated automatically

Just two classes have a definition: Prompt and PromptEngineering with no individuals created while the PromptingTechnique class has no defintion and three individuals created:

- ChainOfThoughtPrompting
- FewShotPrompting
- ZeroShotPrompting

There is no new instance of chat and all the mechanism defined to link a prompting technique with a chat is completely ignored. No new object properties or data properties have been created by GPT-4o, just an annotation property called hasDefinition. Any other useful information is not created, the three entities are not linked with any object property and they do not have any data property. Additional prompts would clearly be needed as input for the LLM to improve the result, which is currently poor and adds no useful information compared to the original, manually populated version of the prompt engineering ontology.

Chapter 5

Experimental Evaluation, Publication, and Maintenance

This Chapter reports the evaluation process of PEO. Sect. 5.1 discusses the process of the ontology consistency check, sect. 5.2 measures ontology metrics , sect. 5.3 detects pitfall in PEO and sect. 5.4 checks the execution of translated competency questions. In the final part of the chapter, sect. 5.5 discuss the publication and sect. 5.6 discusses results obtained during the experimental evaluation.

5.1 Ontology Consistency Check

The ontology consistency check is a reasoning task available in OWL, and consequently Protege, that assesses the consistency of the ABox with respect to the TBox. Hence, we execute the reasoner available in Protege that performs such check as well as other reasoning services including the rule-based inference. We specifically adopt the HermiT reasoner. Moreover, we consider both the manually populated and the automatic populate ontology. On the original version of the PEO, the reasoner does not give any inconsistency and all the axioms are inferred correctly.

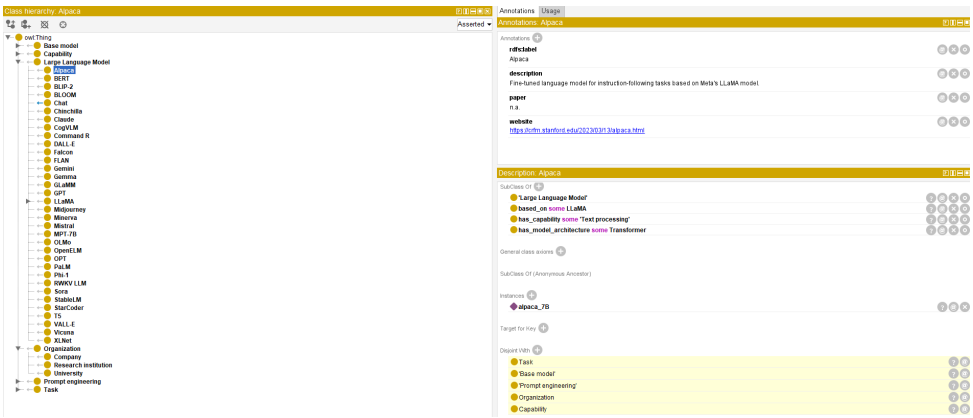

FIGURE 5.1: PEO with HermiT reasoner

Fig. 5.1 shows the execution of HermiT reasoner on PEO. The reasoner infers correctly that the class "Alpaca" is disjoint with the classes "Task", "Base model", "Prompt engineering", "Organization" and "Capability" because the

superclass "Large language model" is declared disjoint with that list of classes. The ten SWRL rules declared work properly and infer the correct object properties for individuals:

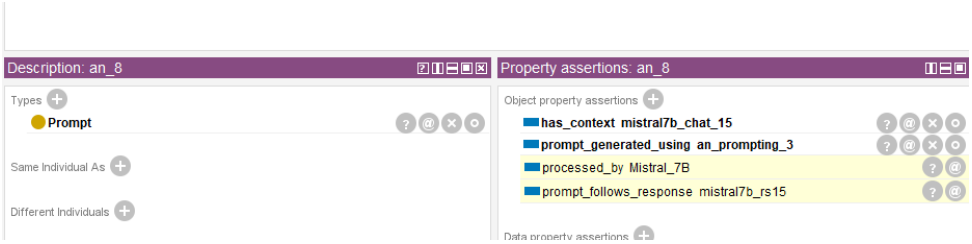

FIGURE 5.2: Inference on prompt individual

The consistency check of the other version of the PEO, the one populated using GPT-4 has given the exact results, because as said in the previous section no additional significant information has been added to the ontology. The classes added by GPT-4 have no link with the other existing classes and thus they do not lead to new inferences.

5.2 OntoMetrics

Moreover, we evaluated PEO based on OntoMetrics: a web-based tool developed by the University of Rostock that validates and provides statistical analyses of ontologies [48]. Given the ontology in form of RDF file or code, it calculates automatically:

- Base metrics: these include simple counts of ontology elements such as classes, axioms, and objects, providing a quantitative overview of the ontology's components.
- Schema Metrics: these metrics evaluate the structure of the ontology's schema, considering aspects like attribute richness and inheritance richness.
- Knowledge base Metrics: these assess the ontology's knowledge base, focusing on the population of instances within the ontology.
- Class Metrics: these metrics analyse individual classes within the ontology, examining factors such as class connectivity and fullness.
- Graph Metrics: these evaluate the ontology's taxonomy as a graph, measuring properties like depth and breadth.

For PEO, I will calculate base metrics, schema metrics and graph metrics.

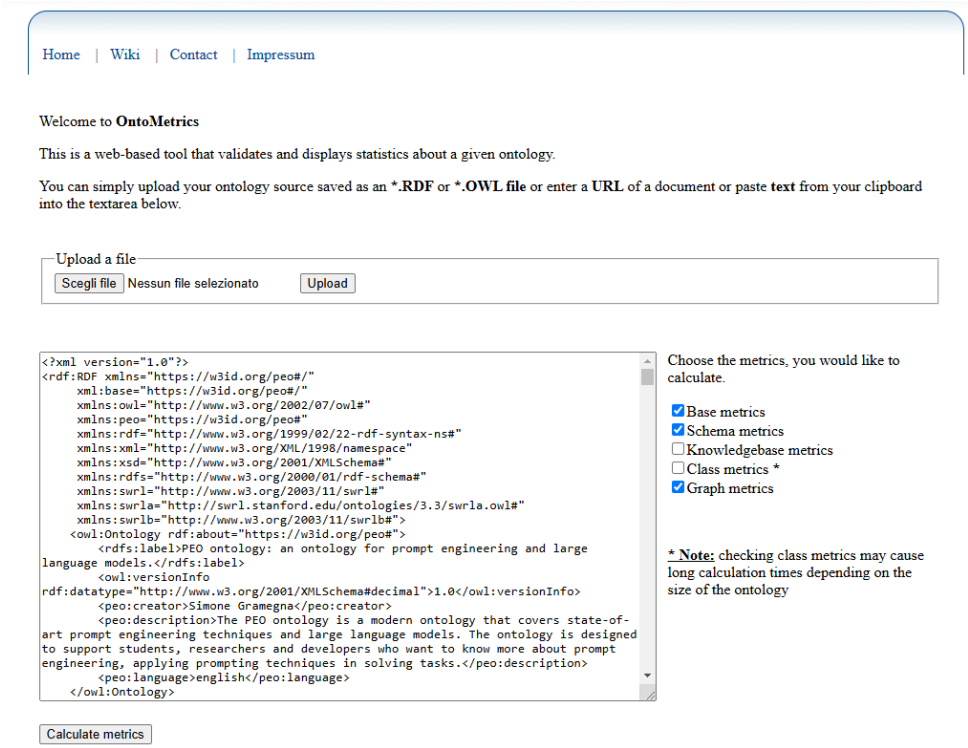

FIGURE 5.3: OntoMetrics interface

The values computed for base metrics are reported in Tab. 5.1.

Property	Value
Axioms	2684
Logical axioms count	1695
Class count	126
Total classes count	126
Object property count	34
Total object properties count	34
Data property count	13
Total data properties count	13
Properties count	47
Individual count	352
Total individuals count	352
DL expressivity	SRIF(D)

TABLE 5.1: PEO base metrics statistics

This table provides a high-level overview of the ontology's structure and complexity. With 2684 axioms, the ontology is rich in detail and represents a substantial body of knowledge. Among these, 1695 logical axioms indicate a strong focus on enabling inference and reasoning. The 126 classes show that the ontology has a robust framework for organizing concepts, while 34 object properties and 13 data properties highlight the ontology's emphasis on relationships over attribute-driven modelling. The inclusion of 352 individuals suggests that the ontology is well-populated, demonstrating practical applicability. The expressivity of $SRIF(D)$ indicates that the ontology supports features like inverse roles and data ranges, balancing computational

efficiency with expressive capability. This ontology is comprehensive, but may require optimization to handle its inherent complexity effectively.

Class Axiom Type	Count
SubClassOf axioms count	199
Equivalent classes axioms count	1
Disjoint classes axioms count	3
GCICount	0
HiddenGCICount	0

TABLE 5.2: PEO Class Axioms statistics

The table 5.2 contains the count of the different class axiom types. There are one hundred ninety-nine SubClassOf axioms, there is just one Equivalent class axiom and there are three Disjoint classes axioms. The Global Cardinality Restrictions count (GCICount) and Hidden Global Cardinality Restrictions count (HiddenGCICount) are both equal to zero. Those two indicators equal to zero mean that there are not explicit and implicit global cardinality restrictions.

Object Property Axiom Type	Count
SubObjectPropertyOf axioms count	33
Equivalent object properties axioms count	0
Inverse object properties axioms count	16
Disjoint object properties axioms count	0
Functional object properties axioms count	0
Inverse functional object properties axioms count	0
Transitive object property axioms count	2
Symmetric object property axioms count	0
Asymmetric object property axioms count	0
Reflexive object property axioms count	0
Irreflexive object property axioms count	1
Object property domain axioms count	33
Object property range axioms count	33
SubPropertyChainOf axioms count	0

TABLE 5.3: PEO Object Property Axioms Statistics

The table 5.3 contains object property axiom types count in PEO. There are thirty-three SubObjectPropertyOf axioms, while no Equivalent object properties or Disjoint object properties axioms are present. Additionally, the ontology includes 16 Inverse object properties axioms and two Transitive object property axioms. Furthermore, no Functional, Inverse functional, Symmetric, or Asymmetric object property axioms are defined. There is also no Reflexive object property axiom, whereas one Irreflexive object property axiom is present.Regarding domain and range definitions, the ontology contains thirty-three Object property domain axioms and 33 Object property range axioms. Lastly, there are no SubPropertyChainOf axioms.

Data Property Axiom Type	Count
SubDataPropertyOf axioms count	12
Equivalent data properties axioms count	0
Disjoint data properties axioms count	0
Functional data property axioms count	7
Data property domain axioms count	11
Data property range axioms count	12

TABLE 5.4: PEO Data Property Axioms Statistics

Table 5.4 presents the count of different data property axiom types in PEO. Specifically, there are twelve SubDataPropertyOf axioms, while no Equivalent data properties or Disjoint data properties axioms are present. Additionally, the ontology includes seven Functional data property axioms. Regarding domain and range definitions, the ontology contains eleven Data property domain axioms and twelve Data property range axioms.

Individual Axiom Type	Count
Class assertion axioms count	352
Object property assertion axioms count	390
Data property assertion axioms count	580
Negative object property assertion axioms count	0
Negative data property assertion axioms count	0
Same individuals axioms count	0
Different individuals axioms count	0

TABLE 5.5: PEO Individual Axioms Statistics

Table 5.5 presents the count of different individual axiom types in PEO. Specifically, there are three hundred fifty-two Class assertion axioms, three hundred ninety Object property assertion axioms, and five hundred eighty Data property assertion axioms. Additionally, there are no Negative object property assertion axioms, Negative data property assertion axioms, Same individuals axioms, or Different individuals axioms in the ontology.

Annotation Axiom Type	Count
Annotation axioms count	5
Annotation assertion axioms count	459
Annotation property domain axioms count	0
Annotation property range axioms count	0

TABLE 5.6: PEO Annotation Axioms Statistics

Table 5.6 presents the count of different annotation axiom types in PEO. Specifically, there are five Annotation axioms and four hundred fifty-nine Annotation assertion axioms. Additionally, there are no Annotation property domain axioms or Annotation property range axioms in the ontology.

Metric	Value
Attribute richness	0.103175
Inheritance richness	1.579365
Relationship richness	0.160338
Attribute class ratio	0.0
Equivalence ratio	0.007937
Axiom/class ratio	21.301587
Inverse relations ratio	0.390244
Class/relation ratio	0.531646

TABLE 5.7: PEO Schema Metrics

Table 5.7 presents the schema metrics for PEO. The Attribute richness is approximately zero point one, while the Inheritance richness is around one point six. The Relationship richness is about zero point one six. The Attribute class ratio is zero, and the Equivalence ratio is approximately zero point zero zero eight. The Axiom per class ratio is around twenty-one point three, whereas the Inverse relations ratio is about zero point three nine. Finally, the Class per relation ratio is approximately zero point five three.

Metric	Value
Absolute root cardinality	7
Absolute leaf cardinality	109
Absolute sibling cardinality	126
Absolute depth	318
Average depth	2.52381
Maximal depth	4
Absolute breadth	126
Average breadth	7.0
Maximal breadth	33
Ratio of leaf fan-outness	0.865079
Ratio of sibling fan-outness	1.0
Tangledness	0.269841
Total number of paths	126
Average number of paths	31.5

TABLE 5.8: PEO Graph Metrics

The table 5.8 The table 5.8 provides graph metrics of PEO. The absolute root cardinality (7) and absolute leaf cardinality (109) indicate a moderately deep hierarchy with good granularity at the leaf level. The maximal depth (4) and average depth (2.52381) suggest a shallow graph, which can make the ontology easier to understand but may oversimplify complex domains. The tangledness (0.269841) is moderate, indicating a graph structure that is connected but not overly complex. The high sibling fan-out ratio (1.0) shows that sibling classes are evenly distributed, while the ratio of leaf fan-outness (0.865079) implies a balanced spread of subclasses from intermediate nodes.

Despite there being no significant differences between the original version of the PEO and the version populated using GPT-4, we proceed to calculate metrics.

Values computed for base metrics are:

Property	Value
Axioms	2705
Logical axioms count	1710
Class count	128
Total classes count	128
Object property count	34
Total object properties count	34
Data property count	13
Total data properties count	13
Properties count	47
Individual count	359
Total individuals count	359
DL expressivity	SRIF(D)

TABLE 5.9: PEO updated base metrics statistics

Class Axiom Type	Count
SubClassOf axioms count	197
Equivalent classes axioms count	1
Disjoint classes axioms count	3
GCICount	0
HiddenGCICount	0

TABLE 5.10: PEO updated Class Axioms Statistics

Object Property Axiom Type	Count
SubObjectPropertyOf axioms count	33
Equivalent object properties axioms count	0
Inverse object properties axioms count	16
Disjoint object properties axioms count	0
Functional object properties axioms count	0
Inverse functional object properties axioms count	0
Transitive object property axioms count	8
Symmetric object property axioms count	1
Asymmetric object property axioms count	0
Reflexive object property axioms count	0
Irreflexive object property axioms count	1
Object property domain axioms count	33
Object property range axioms count	33
SubPropertyChainOf axioms count	0

TABLE 5.11: PEO updated Object Property Axioms Statistics

Data Property Axiom Type	Count
SubDataPropertyOf axioms count	12
Equivalent data properties axioms count	0
Disjoint data properties axioms count	0
Functional data property axioms count	7
Data property domain axioms count	12
Data property range axioms count	12

TABLE 5.12: PEO updated Data Property Axioms Statistics

Individual Axiom Type	Count
Class assertion axioms count	358
Object property assertion axioms count	393
Data property assertion axioms count	580
Negative object property assertion axioms count	0
Negative data property assertion axioms count	0
Same individuals axioms count	0
Different individuals axioms count	0

TABLE 5.13: PEO updated Individual Axioms Statistics

Annotation Axiom Type	Count
Annotation axioms count	5
Annotation assertion axioms count	456
Annotation property domain axioms count	0
Annotation property range axioms count	0

TABLE 5.14: PEO updated Annotation Axioms Statistics

The schema metrics for the update version of PEO are:

Metric	Value
Attribute richness	0.101563
Inheritance richness	1.539063
Relationship richness	0.161702
Attribute class ratio	0.0
Equivalence ratio	0.007813
Axiom/class ratio	21.132813
Inverse relations ratio	0.390244
Class/relation ratio	0.544681

TABLE 5.15: PEO Schema Metrics

The graph metrics for updated version of PEO are:

Metric	Value
Absolute root cardinality	11
Absolute leaf cardinality	111
Absolute sibling cardinality	128
Absolute depth	316
Average depth	2.46875
Maximal depth	4
Absolute breadth	128
Average breadth	7.111111
Maximal breadth	33
Ratio of leaf fan-outness	0.867188
Ratio of sibling fan-outness	1.0
Tangledness	0.265625
Total number of paths	128
Average number of paths	32.0

TABLE 5.16: PEO updated Graph Metrics

The new values calculated by OntoMetrics on the updated version of the ontology do not differ significantly from the previously calculated values, as the large language model GPT-4 has only added four new classes and three new instances. All results for PEO and its second version are on Github [1].

5.3 Static Validation

The methods discussed so far are widely used in ontology evaluation; however, they are not capable of capturing modelling errors in greater detail. These errors can compromise not only the quality and usability of the ontology but also lead to potential inconsistencies. Therefore, it is necessary to adopt a different approach from those previously examined, OntoMetrics and the HermiT reasoner, because they have the following issues:

- OntoMetrics: it provides a quantitative measure of ontology's structural characteristics without analysing qualitative aspects, so it is not able to detect semantic issues in the ontology.
- HermiT reasoner (reasoners in general): reasoners are designed to check the logical consistency of an ontology and perform inferencing identifying inconsistencies in axioms. But reasoners are not able to detect errors outside logical inconsistencies, such as missing domain or range definitions and they cannot check best-practice violation and usability issues.

These limitations highlight the need for a tool that can go beyond structural and logical evaluation, addressing both semantic and usability dimensions of ontology quality. OOPS! (Ontology Pitfall Scanner) is an online tool for ontology evaluation [66] that automates the evaluation process without requiring any effort by the developer.

[1] https://github.com/simonegramegna/peo/tree/main/evaluation

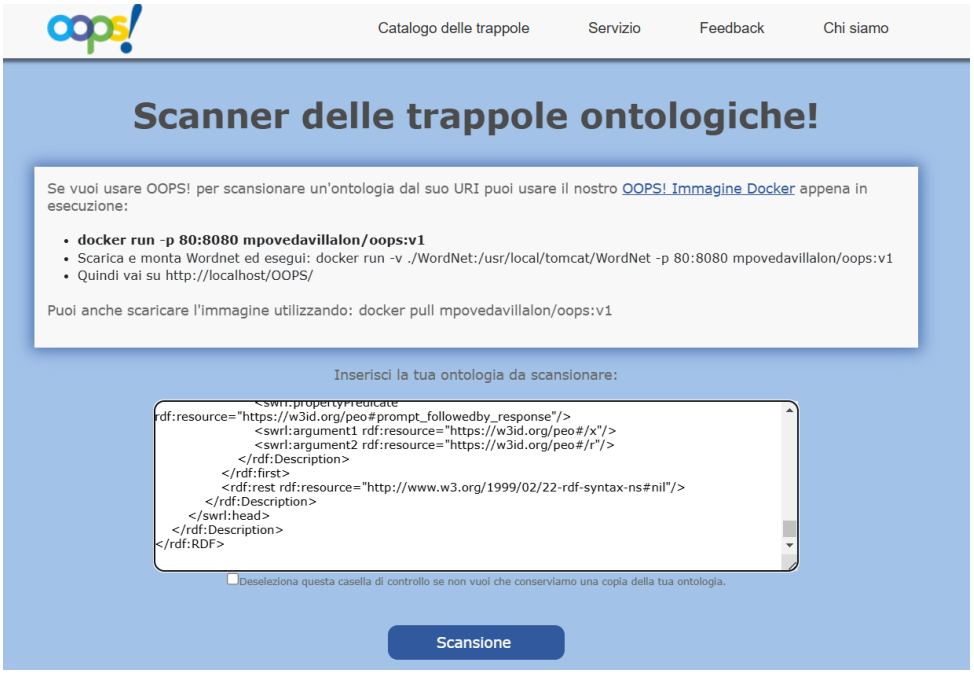

FIGURE 5.4: OOPS! web interface

Fig 5.4 shows the oops web interface. Given the input ontology, OOPS is able to detect 40 different pitfall that are classified into three categories: critical, important and minor by parsing the RDF code and generating a complete response using the OOPS! scanner. Pitfalls in the scanner include:

- Structural pitfalls: formal structure and syntax like cycles in hierarchy and unconnected ontology elements.
- Functional pitfalls: use and functionality of the ontology like missing domain or incorrectly defined inverse relationships.
- Usability and Profiling Pitfalls: clarity, maintainability, and human-readability like missing annotations, inconsistent naming and ambiguous terms.

I will use OOPS! scanner on both versions of the PEO detecting and explaining found pitfalls. For the first version of PEO there are just three pitfalls:

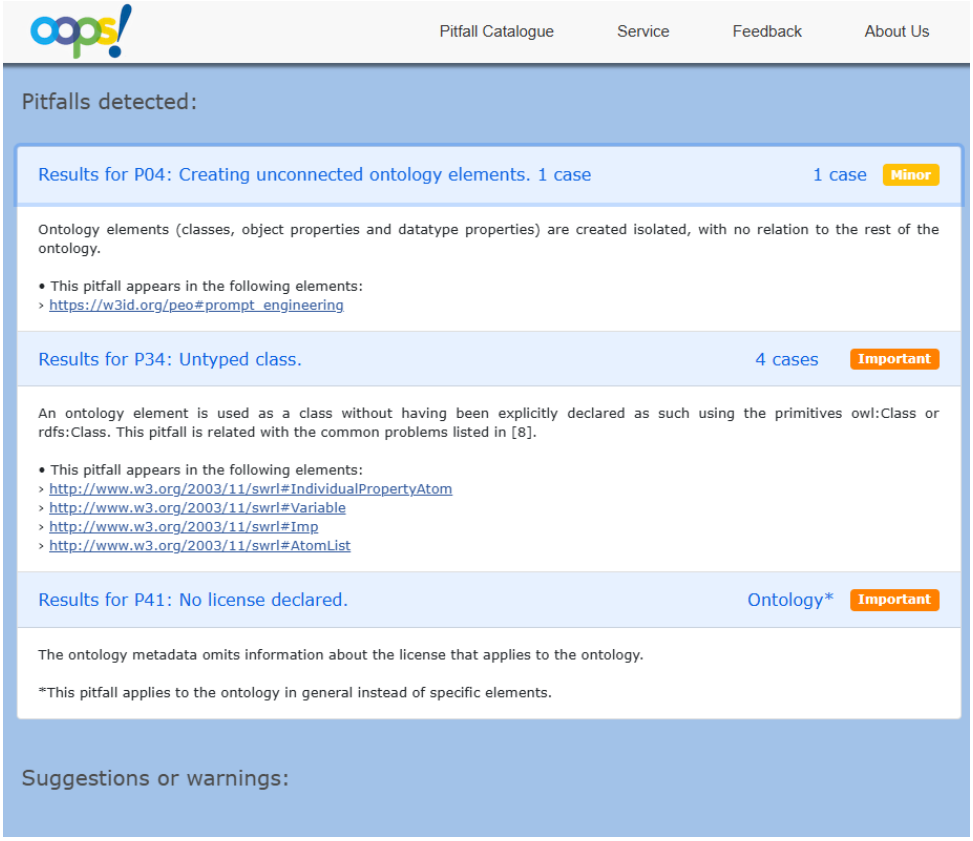

FIGURE 5.5: Detected pitfalls PEO

The explanation of detected pitfall is:

- P04 (Creating unconnected ontology elements): this pitfall involves ontology elements (classes, object properties and datatype properties) are created isolated, with no relation to the rest of the ontology.
- P34 (Untyped class): this pitfalls involves all ontology elements that are used as a class without having explicitly declared. This pitfall involves four elements in PEO:
  http://www.w3.org/2003/11/swrl#IndividualPropertyAtom,
  http://www.w3.org/2003/11/swrl#Variable,
  http://www.w3.org/2003/11/swrl#Imp and
  http://www.w3.org/2003/11/swrl#AtomList.
- P41 (No license declared): there is no license in the ontology metadata.

Overall, the report is satisfactory, as no critical pitfalls have been identified, nor any issues found in the main classes or object properties of the ontology and I will discuss in more detail in Results discussion section. The complete report is available here [2]. After detecting pitfalls in the original version of the PEO, I also examined the populated version generated using GPT-4 nonetheless, no significant differences were observed between the two versions. Compared to the three pitfalls identified in the previous version, OOPS detects five pitfalls:

[2]https://github.com/simonegramegna/peo/blob/main/evaluation/oops_report_peo.xml

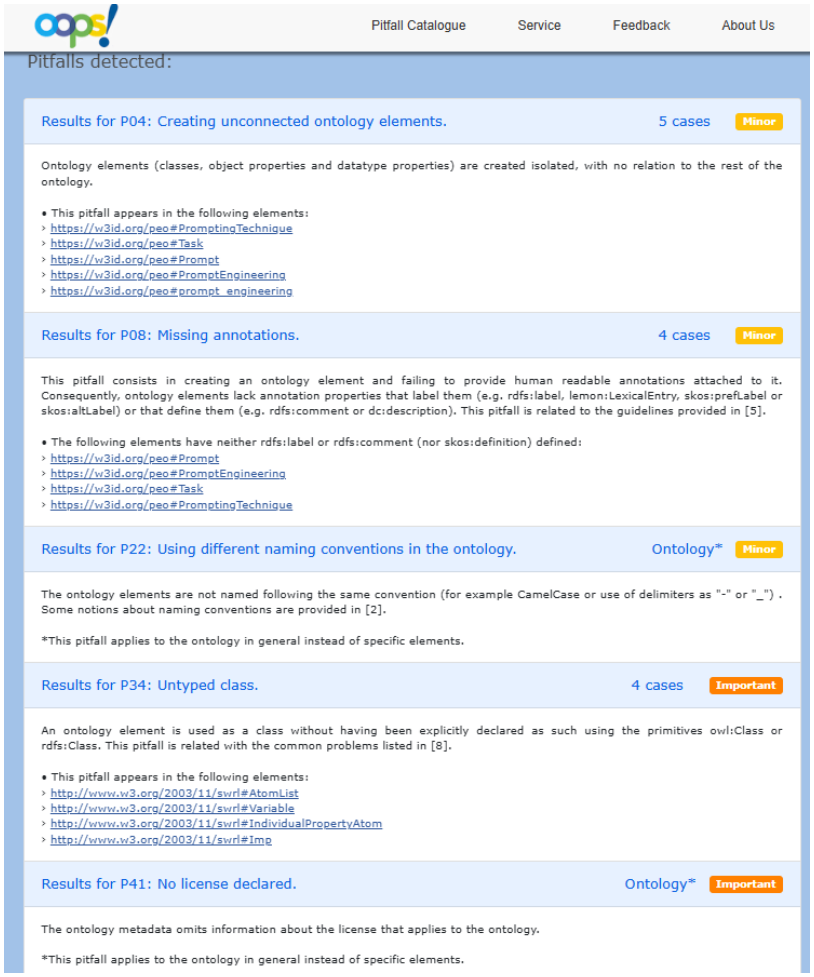

FIGURE 5.6: Detected pitfalls second version PEO

- P04 (Creating unconnected ontology elements): this pitfall, as said before, involves unconnected ontology elements. This time the pitfall involves not only the prompt_engineering class but also the new classes created by GPT-4: Task, PromptingTechnique, Prompt and PromptEngineering. Those classes have no relations with other classes.

- P08 (Missing annotations): this pitfall involves elements that lack annotation properties that label them (rdfs:label) or that define them (rdfs:comment). This pitall involves the four classes created by GPT-4 with no comment provided: Task, PromptingTechnique, Prompt and PromptEngineering.

- P22 (Using different naming conventions in the ontology): this pitfall involves ontology elements are not named following the same convention (for example CamelCase or use of delimiters as "-" or "_"). This pitfall is given by the new classes created by GPT-4 that do not follow the naming convention used in the ontology (with the "_" separator) by creating classes declared in camel case.

- P34 (Untyped class): this is the same pitfall detected before.

- P41 (No license declared): this is the same pitfall detected before.

In general, the detected pitfalls are caused by GPT-4's lack of understanding of the ontology's structure and I will discuss in more detail in the Results discussion section. The oops report is available here [3].

5.4 Competency Questions

The final step of ontology evaluation is converting the CQs defined in the Ontology requirements specification section into SPARQL queries in order to compare the expected result to the actual result of each CQ. SPARQL queries are generated manually and executed using Jupyter notebook [4]: an enviroment for running python code and the rdflib python library rdflib [5]: a library to access rdf files and run SPARQL queries inside python. The code is available in the Github repository [6]. The evaluation is mode for both versions of the ontology.
Firstly, the ontology is loaded with rdflib as represented in fig. 5.7

```
[73]: from rdflib import Graph
      import pandas as pd

      peo_path = "../peo_ontology.rdf"

      peo = Graph()
      peo.parse(peo_path, format="xml")
```

FIGURE 5.7: Jupyter notebook SPARQL

The pandas library is used to display query results in data-frames. For such purpose, we implemented three helper functions:

- execute_query: it takes as input the SPARQL query string and returns a list of results.
- results_to_df: it takes as input as input the list of results ad creates and returns a dataframe with results inside which columns' names are the label names.
- print_results: it iterates on the list of results and prints each element.

We can see the implementation below, in fig. 5.8:

[3] https://github.com/simonegramegna/peo/blob/main/evaluation/oops$_r eport_p eo_g pt4.xml$
[4] https://jupyter.org/
[5] https://rdflib.readthedocs.io/en/stable/
[6] https://github.com/simonegramegna/peo/tree/main/evaluation

[40]:
```python
def execute_query(ontology, sparql_query: str):
    results_list = []
    results = ontology.query(sparql_query)

    for row in results:
        results_list.append(row)

    return results_list


def results_to_df(results_list: list):
    column_names = results_list[0].labels

    data = []
    for row in results_list:
        row_dict = {column: str(row[column]) if row[column] else None for column in column_names}
        data.append(row_dict)

    df = pd.DataFrame(data)
    return df


def print_results(results: list):
    for r in results:
        print(r)
```

FIGURE 5.8: Support python functions

Once this is done, we can translate each of the sixteen CQs into SPARQL queries to be executed on PEO.
The first CQ CQ1: What is prompt engineering? is translated into:

```
SELECT DISTINCT ?property ?value
WHERE {
    <https://w3id.org/peo#prompt_engineering> ?property
        ?value .
}
```

[4]:

	property	value
0	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
1	http://www.w3.org/2000/01/rdf-schema#comment	Practice of crafting and optimizing input prom...
2	http://www.w3.org/2000/01/rdf-schema#label	Prompt engineering

FIGURE 5.9: CQ1 SPARQL query results

The output, in fig. 5.9, from the query matches the expected result, which is the definition of prompt engineering.

The second competency question CQ2: What is a prompt? is translated into:

```
SELECT DISTINCT ?property ?value
WHERE {
    <https://w3id.org/peo#prompt> ?property ?value .
}
```

[6]:

	property	value
0	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
1	http://www.w3.org/2000/01/rdf-schema#subClassOf	https://w3id.org/peo#prompt_engineering
2	http://www.w3.org/2000/01/rdf-schema#comment	Input to a large language model.
3	http://www.w3.org/2000/01/rdf-schema#label	Prompt

FIGURE 5.10: CQ2 SPARQL query results

The output, in fig. 5.10, also in this case matches the expected result, which is the definition of prompt.

The third competency question CQ3: What are prompting techniques? is translated into:

```
SELECT DISTINCT ?subclass ?label
WHERE {
    ?subclass rdfs:subClassOf <https://w3id.org/peo#
       prompting_technique> .
    OPTIONAL { ?subclass rdfs:label ?label . }
}
```

[8]:

	subclass	label
0	https://w3id.org/peo#active_prompting	Active prompting
1	https://w3id.org/peo#analogical_prompting	Analogical prompting
2	https://w3id.org/peo#automatic_cot_prompting	Automatic Chain-of-Thought prompting
3	https://w3id.org/peo#chain_of_knowledge_prompting	Chain-of-Knowledge prompting
4	https://w3id.org/peo#chain_of_note_prompting	Chain-of-Note prompting
5	https://w3id.org/peo#chain_of_table_prompting	Chain-of-Table prompting
6	https://w3id.org/peo#chain_of_thought_prompting	Chain-of-Thought prompting
7	https://w3id.org/peo#chain_of_verification_pro...	Chain-of-Verification prompting

FIGURE 5.11: CQ3 SPARQL query results

The complete table contains, in fig. 5.11, all the prompting techniques in the ontology.

The fourth competency question CQ4: What are image prompting techniques? is translated into:

```
SELECT DISTINCT ?subclass ?label
WHERE {
    ?subclass rdfs:subClassOf <https://w3id.org/peo#
       image_prompting> .
    OPTIONAL { ?subclass rdfs:label ?label . }
}
```

[10]:

	subclass	label
0	https://w3id.org/peo#fix_deformed_generations	Fix deformed generations
1	https://w3id.org/peo#lighting	Lighting
2	https://w3id.org/peo#quality_boosters	Quality boosters
3	https://w3id.org/peo#repetition	Repetition
4	https://w3id.org/peo#shot_type	Shot type
5	https://w3id.org/peo#style_modifiers	Style modifiers

FIGURE 5.12: CQ4 SPARQL query results

As expected. in fig 5.12, we get all the image prompting techniques in the ontology.

The fifth competency question CQ5: What are code prompting techniques? is translated into:

```
SELECT DISTINCT ?subclass ?label
WHERE {
    ?subclass rdfs:subClassOf <https://w3id.org/peo#
       code_prompting> .
    OPTIONAL { ?subclass rdfs:label ?label . }
}
```

[12]:

	subclass	label
0	https://w3id.org/peo#chain_of_code_prompting	Chain-of-Code prompting
1	https://w3id.org/peo#program_of_thoughts_promp...	Program of Thoughts prompting
2	https://w3id.org/peo#scratchpad_prompting	Scratchpad prompting
3	https://w3id.org/peo#structured_cot_prompting	Structured Chain-of-Thought prompting

FIGURE 5.13: CQ5 SPARQL query results

The table contains, in 5.13, all the code prompting techniques in the ontology.

The sixth competency question CQ6: Which task does a prompt solve? is translated into:

```
SELECT DISTINCT ?prompt ?task ?taskLabel
WHERE {
    ?prompt <https://w3id.org/peo#solves> ?task .
    OPTIONAL { ?task rdfs:label ?taskLabel . }
}
```

[14]:

	prompt	task
0	https://w3id.org/peo#an_prompting_1	https://w3id.org/peo#emotion_classification_1
1	https://w3id.org/peo#em_prompting_1	https://w3id.org/peo#emotion_classification_1
2	https://w3id.org/peo#fs_prompting_1	https://w3id.org/peo#emotion_classification_1
3	https://w3id.org/peo#ro_prompting_1	https://w3id.org/peo#emotion_classification_1
4	https://w3id.org/peo#zs_prompting_1	https://w3id.org/peo#emotion_classification_1
5	https://w3id.org/peo#an_prompting_2	https://w3id.org/peo#math_1
6	https://w3id.org/peo#em_prompting_2	https://w3id.org/peo#math_1
7	https://w3id.org/peo#fs_prompting_2	https://w3id.org/peo#math_1
8	https://w3id.org/peo#ro_prompting_2	https://w3id.org/peo#math_1
9	https://w3id.org/peo#zs_prompting_2	https://w3id.org/peo#math_1

FIGURE 5.14: CQ6 SPARQL query results

The table contains, in fig 5.14, all the prompts that solve tasks.

The seventh competency question CQ7: Which prompts are generated using a prompting technique? is translated into:

```
SELECT DISTINCT ?prompt ?technique ?techniqueLabel
WHERE {
    ?prompt <https://w3id.org/peo#prompt_generated_using> ?
       technique .
    OPTIONAL { ?technique rdfs:label ?techniqueLabel . }
}
```

[16]:

	prompt	technique
0	https://w3id.org/peo#an_1	https://w3id.org/peo#an_prompting_1
1	https://w3id.org/peo#an_2	https://w3id.org/peo#an_prompting_1
2	https://w3id.org/peo#an_3	https://w3id.org/peo#an_prompting_1
3	https://w3id.org/peo#an_10	https://w3id.org/peo#an_prompting_4

FIGURE 5.15: CQ7 SPARQL query results

The complete table, in fig 5.15, contains all the prompt instances generated using instances of prompting techniques.

The eighth competency question CQ8: What are the responses that follow each prompt? is translated into:

```
SELECT DISTINCT ?prompt ?response
WHERE {
    ?response <https://w3id.org/peo#
       response_followedby_prompt> ?prompt .
}
```

[18]:

	prompt	response
0	https://w3id.org/peo#zs_3	https://w3id.org/peo#gemini_rs1
1	https://w3id.org/peo#an_6	https://w3id.org/peo#gemini_rs10
2	https://w3id.org/peo#zs_9	https://w3id.org/peo#gemini_rs11
3	https://w3id.org/peo#fs_9	https://w3id.org/peo#gemini_rs12
4	https://w3id.org/peo#em_9	https://w3id.org/peo#gemini_rs13
5	https://w3id.org/peo#ro_9	https://w3id.org/peo#gemini_rs14

FIGURE 5.16: CQ8 SPARQL query results

The complete table contains, in fig. 5.16, all the responses of all prompt instances.

The ninth competency question CQ9: What are possible tasks? is translated into:

```
SELECT DISTINCT ?task ?label
WHERE {
    ?task rdf:type owl:Class .
    ?task rdfs:subClassOf* <https://w3id.org/peo#task> .
    OPTIONAL { ?task rdfs:label ?label . }
}
```

[20]:

	task	label
0	https://w3id.org/peo#task	Task
1	https://w3id.org/peo#audio_task	Audio task
2	https://w3id.org/peo#audio_explanation	Audio explanation
3	https://w3id.org/peo#audio_generation	Audio generation
4	https://w3id.org/peo#code_task	Code task
5	https://w3id.org/peo#code_explanation	Code explanation

FIGURE 5.17: CQ9 SPARQL query results

In the completed table, in 5.17, we get all the possible task represented in the ontology.

The tenth competency question CQ10: Which tasks are related to the text? is translated into:

```
SELECT DISTINCT ?task ?label
WHERE {
    ?task rdf:type owl:Class .
    ?task rdfs:subClassOf* <https://w3id.org/peo#text_task>
        .
    OPTIONAL { ?task rdfs:label ?label . }
}
```

[22]:

	task	label
0	https://w3id.org/peo#text_task	Text task
1	https://w3id.org/peo#emotion_classification	Emotion classification
2	https://w3id.org/peo#mathematical_understanding	Mathematical understanding
3	https://w3id.org/peo#question_answering	Question-answering
4	https://w3id.org/peo#text_explanation	Text explanation
5	https://w3id.org/peo#text_generation	Text generation
6	https://w3id.org/peo#text_summarization	Text summarization
7	https://w3id.org/peo#text_translation	Text translation

FIGURE 5.18: CQ10 SPARQL query results

The resulting table, in fig. 5.18, contains, all the task related to text.

The eleventh competency question CQ11: What is a chat? is translated into:

```
SELECT DISTINCT ?property ?value
WHERE {
    <https://w3id.org/peo#chat> ?property ?value .
}
```

[24]:

	property	value
0	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
1	http://www.w3.org/2000/01/rdf-schema#subClassOf	https://w3id.org/peo#prompt_engineering
2	http://www.w3.org/2000/01/rdf-schema#subClassOf	Ndadca356ac344befa09c619a5ec83a4a
3	http://www.w3.org/2000/01/rdf-schema#comment	Interactive dialogue session between a user an...
4	http://www.w3.org/2000/01/rdf-schema#label	Chat

FIGURE 5.19: CQ11 SPARQL query results

The result, in fig. 5.19, is the definition of Chat.

The twelfth competency question CQ12: What is a large language model? is translated into:

```
SELECT DISTINCT ?property ?value
WHERE {
    <https://w3id.org/peo#large_language_model> ?property ?
       value .
}
```

[26]:

	property	value
0	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
1	http://www.w3.org/2000/01/rdf-schema#comment	Type of computational model designed for natur...
2	http://www.w3.org/2000/01/rdf-schema#label	Large Language Model
3	https://w3id.org/peo#reference	https://en.wikipedia.org/wiki/Large_language_m...

FIGURE 5.20: CQ12 SPARQL query results

The result, in fig 5.20, is the definition of Large Language Model.

The thirteenth competency question CQ13: What types of large language models are available? is translated into:

```
SELECT DISTINCT ?type ?label
WHERE {
    ?type rdfs:subClassOf <https://w3id.org/peo#
       large_language_model> .
    OPTIONAL { ?type rdfs:label ?label . }
}
```

[28]:

	type	label
0	https://w3id.org/peo#alpaca	Alpaca
1	https://w3id.org/peo#bert	BERT
2	https://w3id.org/peo#blip-2	BLIP-2
3	https://w3id.org/peo#bloom	BLOOM
4	https://w3id.org/peo#chinchilla	Chinchilla
5	https://w3id.org/peo#claude	Claude
6	https://w3id.org/peo#cogvlm	CogVLM

FIGURE 5.21: CQ13 SPARQL query results

The resulting complete table, in fig. 5.21, is the collection of all large language models represented in the ontology.

The fourteenth competency question CQ14: What are large language models architectures? is translated into:

```
SELECT DISTINCT ?type ?label
WHERE {
    ?type rdfs:subClassOf <https://w3id.org/peo#base_model>
        .
    OPTIONAL { ?type rdfs:label ?label . }
}
```

[30]:

	type	label
0	https://w3id.org/peo#clip	CLIP
1	https://w3id.org/peo#decoder	Decoder
2	https://w3id.org/peo#diffusion_model	Diffusion model
3	https://w3id.org/peo#encoder	Encoder
4	https://w3id.org/peo#recurrent_neural_network	Recurrent Neural Network
5	https://w3id.org/peo#transformer	Transformer

FIGURE 5.22: CQ14 SPARQL query results

The result, in fig. 5.22, is the collection of main large language models architectures.

The fifteenth competency question CQ15: What are large language models capabilities? is translated into:

```
SELECT DISTINCT ?type ?label
WHERE {
    ?type rdfs:subClassOf <https://w3id.org/peo#capability>
        .
    OPTIONAL { ?type rdfs:label ?label . }
}
```

[32]:

	type	label
0	https://w3id.org/peo#audio_processing	Audio processing
1	https://w3id.org/peo#code_processing	Code processing
2	https://w3id.org/peo#image_processing	Image processing
3	https://w3id.org/peo#text_processing	Text processing
4	https://w3id.org/peo#video_processing	Video processing

FIGURE 5.23: CQ15 SPARQL query results

Results are represented, in fig. 5.23, by the collection of all the capabilities represented in PEO.

The sixteenth and last competency question CQ16: What companies develop large language models? is translated into:

```
SELECT DISTINCT ?company ?label
WHERE {
    ?company rdf:type <https://w3id.org/peo#company> .
    OPTIONAL { ?company rdfs:label ?label . }
}
```

[34]:

	company	label
0	https://w3id.org/peo#anthropic	None
1	https://w3id.org/peo#apple	None
2	https://w3id.org/peo#cohere	None
3	https://w3id.org/peo#databricks	None
4	https://w3id.org/peo#deepmind	None
5	https://w3id.org/peo#google	None
6	https://w3id.org/peo#meta	None

FIGURE 5.24: CQ16 SPARQL query results

The result, in fig. **??**, is the collection of companies that develop large language models.
We proceed also, to execute SPARQL queries on the second version of PEO, the one that has been populated automatically using GPT-4. We translated only competency questions that match changes introduced by the LLM. Competency questions chosen for translation are four: CQ1, CQ2, CQ3 and CQ9. The first competency question CQ1: What is prompt engineering? is translated into:

```
SELECT DISTINCT ?property ?value
WHERE {
    <https://w3id.org/peo#PromptEngineering> ?property ?
       value .
}
```

[76]:

	property	value
0	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
1	https://w3id.org/peo#hasDefinition	The practice of crafting effective prompts to ...

FIGURE 5.25: CQ1 SPARQL query results - PEO updated

Fig. 5.25 shows the new output of CQ1.
The second competency question CQ2: What is a prompt? is translated into:

```
SELECT DISTINCT ?property ?value
WHERE {
    <https://w3id.org/peo#Prompt> ?property ?value .
}
```

[78]:

	property	value
0	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
1	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#NamedIndividual
2	https://w3id.org/peo#prompt_generated_using	https://w3id.org/peo#ChainOfThoughtPrompting
3	https://w3id.org/peo#prompt_generated_using	https://w3id.org/peo#FewShotPrompting
4	https://w3id.org/peo#prompt_generated_using	https://w3id.org/peo#ZeroShotPrompting
5	https://w3id.org/peo#hasDefinition	An input or query provided to an AI model to g...

FIGURE 5.26: CQ2 SPARQL query results - PEO updated

Fig. 5.26 shows the new output of CQ2.

The third competency question CQ3: What are prompting techniques? is translated into:

```
SELECT DISTINCT ?instance
WHERE {
  ?instance <http://www.w3.org/1999/02/22-rdf-syntax-ns#
     type> <https://w3id.org/peo#PromptingTechnique> .
}
```

[81]:

	instance
0	https://w3id.org/peo#ChainOfThoughtPrompting
1	https://w3id.org/peo#FewShotPrompting
2	https://w3id.org/peo#ZeroShotPrompting

FIGURE 5.27: CQ3 SPARQL query results - PEO updated

Fig. 5.27 shows the new output of CQ3.

The ninth competency question CQ9: What are possible tasks? is translated into:

```
SELECT DISTINCT ?instance
WHERE {
  ?instance <http://www.w3.org/1999/02/22-rdf-syntax-ns#
     type> <https://w3id.org/peo#Task> .
}
```

[84]:

	instance
0	https://w3id.org/peo#CodePrompting
1	https://w3id.org/peo#ImagePrompting
2	https://w3id.org/peo#TextPrompting

FIGURE 5.28: CQ9 SPARQL query results - PEO updated

Fig. 5.28 shows the new output of CQ9.

5.5 Publication

The last step after the Ontology implementation in the LOT methodology is the Ontology publication phase, which scope is to provide an online ontology accessible both as human-readable documentation and a machine-readable documentation from its URI. This phase is divided into three sub-activities:

1. Propose release candidate
2. Ontology documentation
3. Online publication

as we can see in the figure below:

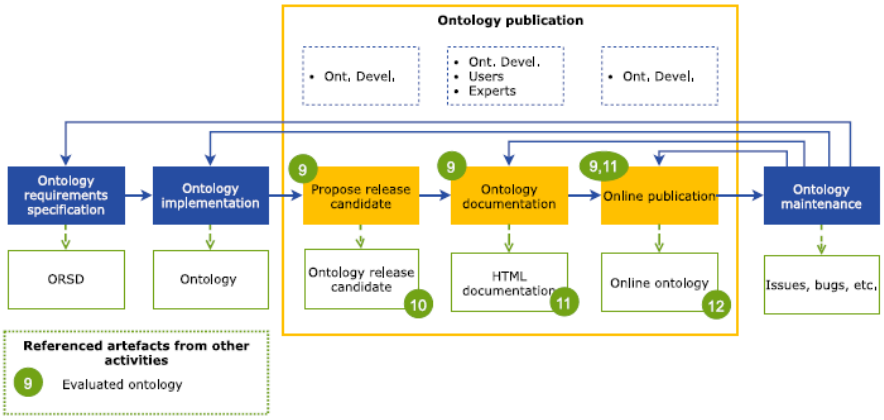

FIGURE 5.29: Ontology publication workflow

Propose Release Candidate

After all the implementation and evaluation, in this step there is the decision about which version of the ontology is going to be published. It is a quite easy choice because, as said in pervious sections, the version populated automatically did not add any useful information from the original version. Moreover it added, according the OOPS! report, more pitfalls (five vs three).

Ontology Documentation

The ontology documentation is generated using Protegé and the OWLdoc plug-in, which automatically generate HTML documentation starting from ontology code. The output of OWLdoc is the following:

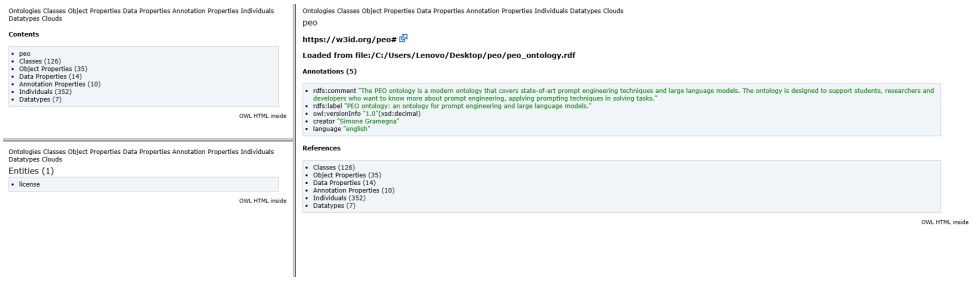

FIGURE 5.30: OWLdoc output

The OWLdoc is available in the GitHub repository here [7] in the /docs folder and it can be consulted online here [8] The documentation is made accessible for everyone using Vercel: a cloud platform for hosting web applications and static sites. This free service using an integrated Github action, takes as input the documentation in the repository and publishes online automatically without requiring any effort by the developer. The official documentation website is: https://peoontology.vercel.app/.

Online Publication

Once the ontology documentation is ready, the ontology can be published online on major vocabularies repository, by accessing the ontology using its URI. I will consider two most known repositories for ontology publishing: w3id.org and BioPortal.

W3id.org a permanent identifier service that provides stable, persistent, and HTTP-resolvable URIs for web resource. The creation of a new identifier for publishing a new ontology is made using Github and the official W3id.org Github repository. The procedure followed is straightforward, first we fork the W3id.org Github repository on my Github, creating a "copy" of the repository.

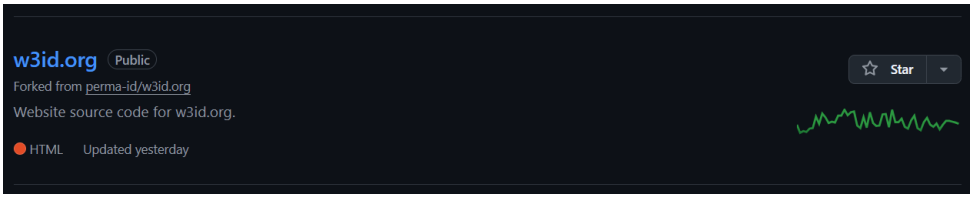

FIGURE 5.31: Forked W3id.org repository

Then we create a new branch and we create a new directory with the intended permanent identifier name, in my case we create a directory called "peo". Inside this directory we create two files:

- README.md: contains more identifier info and contact info, for human to read.

[7] https://github.com/simonegramegna/peo/tree/main/docs
[8] https://peoontology.vercel.app/

- .htaccess: contains redirection rules, for computer to read and perform.

In the case of PEO, the README.md contains all my contact information with the link to the PEO Github repository while the .htaccess contains the following redirections rules:

```
# Activates Rewrite Engine
RewriteEngine On

# Content negotiation for RDF/XML
RewriteCond %{HTTP_ACCEPT} application/rdf\+xml
RewriteRule ^$ https://raw.githubusercontent.com/
   simonegramegna/peo/refs/heads/main/peo_ontology.rdf [R
   =303,L]

# Content negotiation for Turtle
RewriteCond %{HTTP_ACCEPT} text/turtle
RewriteRule ^$ https://raw.githubusercontent.com/
   simonegramegna/peo/refs/heads/main/peo_ontology.ttl [R
   =303,L]

# Default: serves HTML for browser or client not RDF-aware
RewriteRule ^$ https://peoontology.vercel.app/ [R=303,L]

# Blocks directory indexing
Options -Indexes
```

Once the two files are completed, we submitted a pull request to the main branch that is approved by one of repository administrators. As soon the merge of the created branch, it is possible to access to the ontology using its URI, in the case of PEO the URI is: https://w3id.org/peo bringing the user with an HTTP request to the online ontology documentation.

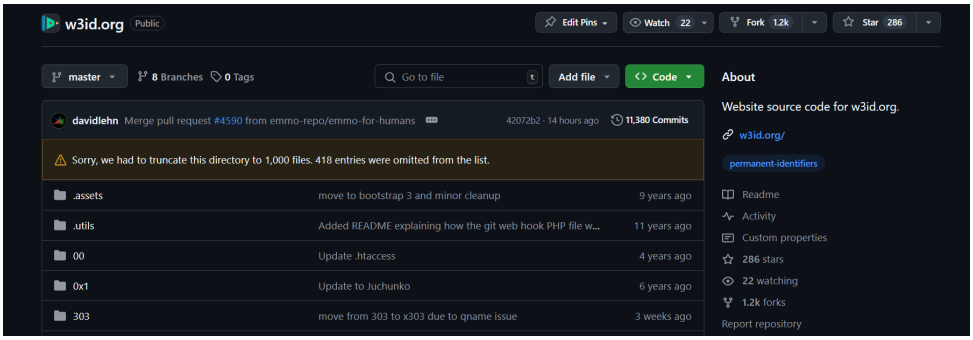

FIGURE 5.32: W3id.org repository main directory

Fig. 5.32 shows W3id.org repository main directory

We decided to publish PEO also on BioPortal[9]. First we create an account on BioPortal and then we specify all the informations about the ontology: the version, the type (RDF) and the release date.

[9]https://bioportal.bioontology.org/

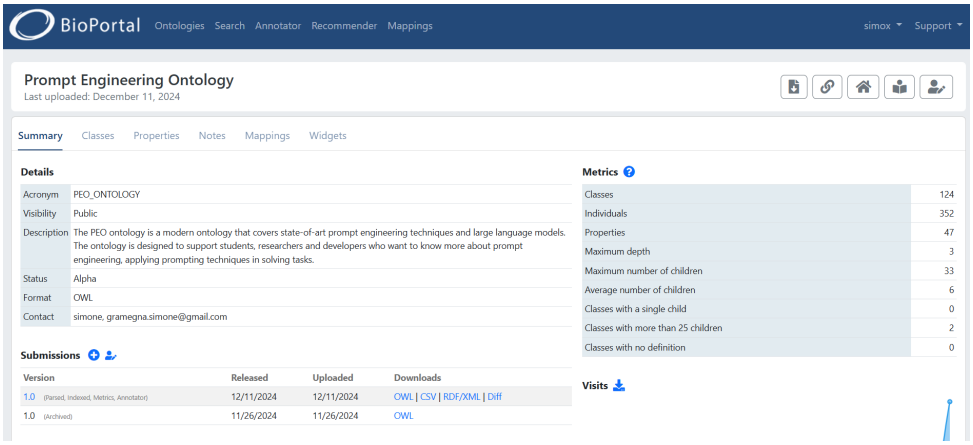

FIGURE 5.33: BioPortal main page

The ontology is published here [10] and it is possible using the web interface view the whole ontology: classes, object properties, data properties and individuals. This is a very useful feature because the user has no need to download any additional software, moreover BioPortal computes automatically base ontology metrics.

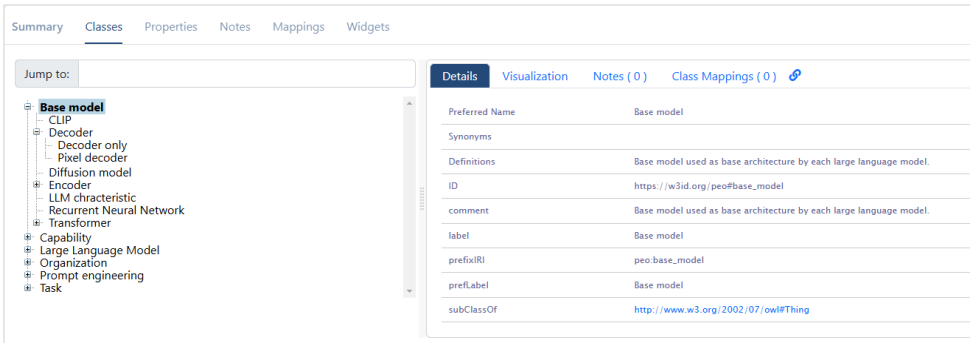

FIGURE 5.34: BioPortal ontology interface

[10] https://bioportal.bioontology.org/ontologies/PEO$_{O}NTOLOGY?p = summary$

5.6 Results Discussion

The design and implementation phases led the creation of an ontology that is the very first to comprehensively and exhaustively represent prompt engineering techniques and large language models. As discussed in previous chapters, resources on these topics are often quite limited and fragmented. So PEO becomes an important reference in these fields providing users with a useful and effective. PEO also supports developers and researchers in choosing the best prompt engineering techniques and LLM for their scopes. Additionally, it assists content creators and copywriters in selecting the most suitable approach and model for automatic content creation.

The results obtained during the evaluation process of PEO reveal important insights into its design, structure, and functionality. Through a combination of tools and methods, such as HermiT reasoner, OntoMetrics, and OOPS! validation, the ontology was rigorously assessed, highlighting both its strengths and areas for improvement.

There are three main strengths of the PEO found during the evaluation:

1. Logical Consistency and Reasoning: the HermiT reasoner confirmed that the ontology's logical structure is sound, with no inconsistencies detected in both the manually populated version and the updated version generated by GPT-4. Moreover, additional triples and axioms are inferred.

2. Expressiveness and Structural Metrics: OntoMetrics demonstrated the ontology's comprehensiveness, with 2684 axioms and $SRIF(D)$ expressivity, supporting advanced reasoning while maintaining computational efficiency. The hierarchical organization (199 SubClassOf axioms) facilitates inheritance. Moreover, the inclusion of 352 individuals indicates practical applicability.

3. Semantic Coverage: competency questions translated into SPARQL queries yielded the expected results, confirming that the ontology effectively represents key concepts, such as prompting techniques, tasks, and the relationships between prompts and their generated responses. This shows that the ontology is well-aligned with its design goals.

The evaluation found also the following limitations:

- Pitfalls in Structure: OOPS! detected structural issues, such as unconnected elements and missing annotations. These issues were more in the GPT-4-populated version, where newly introduced classes (e.g., Task, PromptingTechnique) lacked connections to the broader ontology. This indicates that automated population requires further refinement.

- Simplistic Relationships: metrics such as low relationship richness (0.160338) suggest that the ontology is more focused on defining concepts than on establishing interconnections. While this simplifies reasoning, it may limit the representation of complex semantic relationships.

- Underutilized Properties: object and data property axioms reveal limited use of features such as functional, symmetric, or inverse properties. Incorporating these could enhance the ontology's ability to model constraints and unique relationships.

- Inconsistencies in Naming Conventions: the naming conventions of new classes in the other version of PEO introduced by GPT-4 deviated from the original ontology's standards, potentially affecting usability and integration.

In general, the major issues are present in the version of the PEO populated using GPT-4, as, as previously observed, it has not proven capable of correctly populating the original version of the ontology with new instances. Regarding the original version of the PEO, there are solvable pitfalls, which I will discuss in the next section.

Considering the limitations outlined in the previous section, caused by the pitfalls in the main version of the PEO, we proceed to address and correct the identified pitfalls:

- P04 - Creating unconnected ontology elements: this pitfall involves the prompt_engineering class which has no relation with any other class in the ontology.

- P34 - Untyped class: this pitfall involves elements in SWRL rules.

- P41 - No license declared: this pitfall is due to the absence of the license in the ontology.

The resolution of these pitfalls is essential to enhance the quality of the published ontology.

The pitfall P04, as said before, involves the prompt_engineering class which has no relation with any other class in the ontology, this because the prompt_engineering is a sort of "container" of classes related to prompt. In order to connect prompt_engineering we created the object property applied_to (with inverse relation supports) which connects prompt_engineering with the large_language_model class.

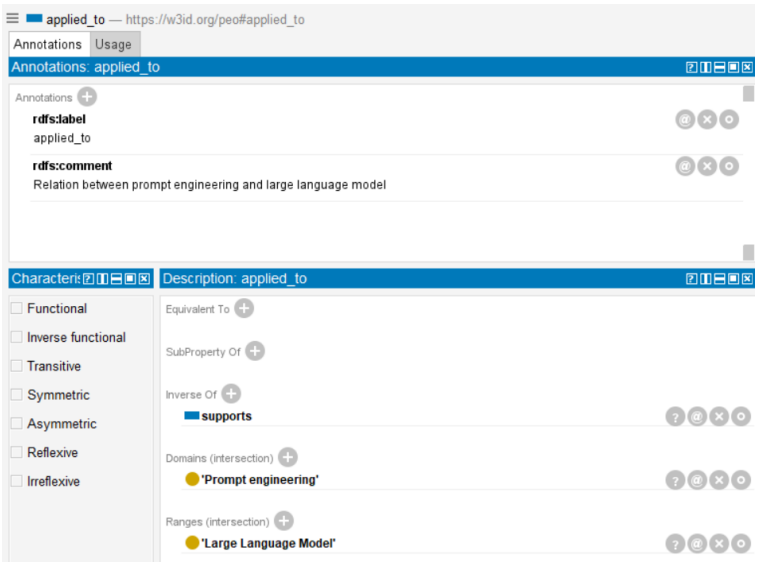

FIGURE 5.35: Object property between Prompt Engineering class and Large Language Model class

The pitfall P34 stems from constructs automatically created by Protegé when a SWRL rule is defined. In fact, SWRL is not part of the OWL languages so the types of the variables are not recognized. To address this issue, we removed the SWRL rules and replaced them with property chains defined in OWL. In this way, it is possible not only to reduce complexity, as SWRL rules involve a non-negligible computational cost, but also to make the ontology more portable, given that not all reasoners support SWRL rules.
In this case, we replaced the SWRL rule:

```
peo:develops(?c, ?x) ^ peo:has_variant(?x, ?y) -> peo:
    develops(?c, ?y)
```

with the following object property chain:

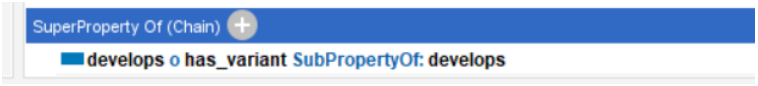

FIGURE 5.36: Object property chaining for develops

The same approach is applied to the other object properties. A particular case is the SWRL rule S2:

```
peo:evolves(?x, ?y) -> peo:has_variant(?x, ?y)
```

In which, instead of using the property chain, we define it directly as an object property evolves as sub property of has_variant as we can see in the figure below:

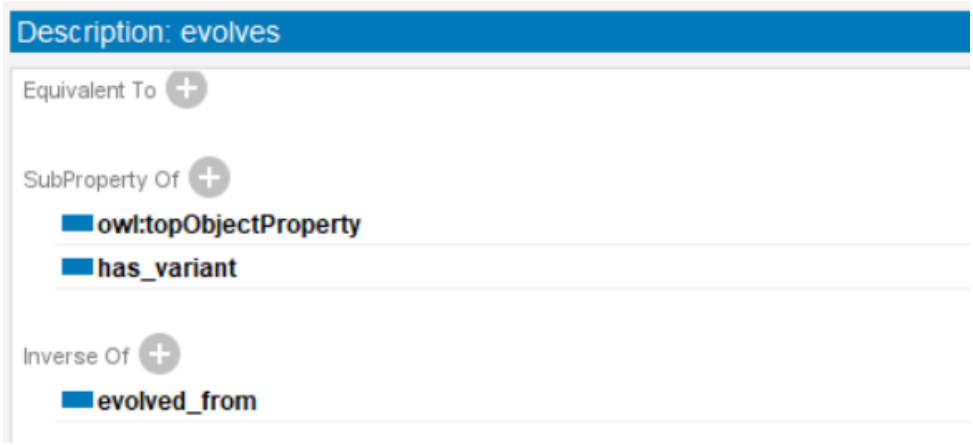

FIGURE 5.37: evolves object property

When the HermiT reasoner is activated, the inferences derived from the property chains are equivalent to those produced by the SWRL rules.

The pitfall P41, as said before, is due to the absence of the license in the ontology. In order to solve this pitfall, I added the annotation property dcterms:license as in [97] with the link of the license (GNU General Public License v3) as value. Fig. 5.38 as we can see in the figure below:

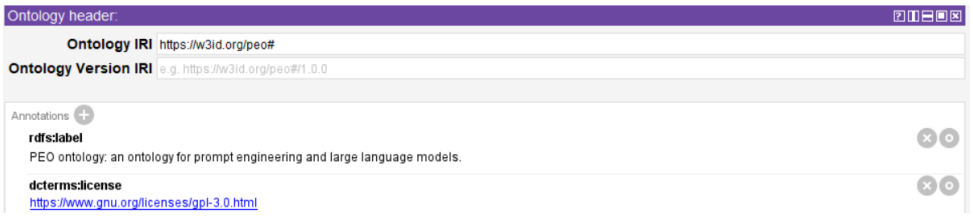

FIGURE 5.38: License annotation in PEO

To solve this pitfall, instead of using software license like GNU General Public License, I used the Creative Commons license available at this link: https://creativecommons.org/licenses/by/4.0/.

During the revision phase, it has been noted that class, object properties and data properties name do not follow the Semantic Web naming conventions that establishes Camel Case for class names, with the first letter in upper case while for object properties and data properties the Camel Case is used but the first letter is in lower case. For example the object property evolved_from with this modification, it becomes evolvedFrom and the large_language_model class becomes LargeLanguageModel.

We stated two RQs in Chapter:

1. RQ 1: Does PEO provide comprehensive and consistent knowledge on LLMs and prompt engineering?

2. RQ 2: Can we use PEO to infer additional knowledge?

At the end of the experimental evaluation, it is possible to give an answer to the RQs. The answer to RQ 1, is: yes. Based on the results obtained during the testing and evaluation phase, we can say that PEO provides comprehensive and consistent knowledge on LLMs and prompt engineering. The execution of SPARQL queries confirms that PEO is able to provide correct informations on concepts like prompt, prompting technique, large language models and task. From the analysis of the results obtained with OntoMetrics, PEO has a good number of classes (126) and individuals (352), with individuals interconnected through 34 object properties. The tangledness equal to 0.26 and the relationships richness equal to 0.16 demonstrates that PEO is not complex and it has moderate relations between concepts.

Also the answer to RQ 2 is: yes. The inference mechanism and property chaining defined in the PEO makes possible to get informations about all the characteristics of each large language model represented and the connections with prompts generated using prompt engineering techniques. Minor issues highlighted by pitfalls have been resolved once identified, ensuring the quality of the final result.

Despite these minor issues, the PEO is well-designed, comprehensive, well-documented, and has very few problems. It significantly outperforms state-of-the-art ontologies related to large language models and prompt engineering.

Chapter 6

Conclusions and Future Works

In this concluding chapter, I present some final thoughts and propose potential directions for future research concerning the development of PEO.

6.1 Conclusions

Inspired by the curiosity to explore an emerging field like prompt engineering and driven by the need to create a comprehensive resource that could be useful to those who share the same interest, we created PEO. During the design and encoding phases, we applied more modern techniques of ontology engineering. This allowed us to make PEO an ontology capable of fully describing all concepts related to prompt engineering and LLMs, considering the interconnections between them. Special attention was given to the experimentation and evaluation phase to ensure the quality of the ontology as well as its consistency with the specifications. The publication of all artifacts online, according to the FAIR principles, makes PEO visible and editable by any user under the Creative Commons license. Users can further advance the project through various future developments.

6.2 Future Works

The future developments of the PEO ontology are numerous and can proceed in different yet complementary directions. Among these, we consider as interesting the following future developments:

- Automatic population using LLMs: experiment with other LLMs and applying more advanced techniques for ontology population.
- Integration with external ontologies: integrate with the ontologies on LLMs
- Improvement of the inference part: further enhance the expressivity of the TBox to gain even more inference capabilities.
- PEO-based AI agent: the development of a PEO-based AI agent can dynamically select and refine prompts based on user intent, task type, and LLM capabilities, optimizing interactions with language models.

- PEO ontology web application: the development of a web application using HTML, CSS, and JavaScript to interact with the PEO ontology is a potential future advancement that could enhance its accessibility and support its wider adoption.

The future developments that will be carried out will surely enhance PEO, a unique project never experimented that was casually born during a Teams call with my advisor on a warm day in mid-May 2024.

La mattina mi alzo e penso a vincere, vado a letto la sera e penso a vincere e quando sono in monoposto, non posso che pensare a vincere.

Charles Leclerc

Bibliography

[1] Accessed: 24-09-2024. URL: https://www.w3.org/TR/owl2-overview/.

[2] Accessed: 25-09-2024. URL: http://neon-project.org/nw/book-chapters/Chapter-01.pdf.

[3] Accessed: 25-09-2024. URL: http://neon-project.org/nw/book-chapters/Chapter-03-1.pdf.

[4] Accessed: 25-09-2024. URL: https://www.w3.org/TR/swbp-vocab-pub/.

[5] *A Perfect guide to Understand Encoder Decoders in Depth with Visuals*. Accessed: 23-10-2024. URL: https://medium.com/@ahmadsabry678/a-perfect-guide-to-understand-encoder-decoders-in-depth-with-visuals-30805c23659b.

[6] Josh Achiam et al. "Gpt-4 technical report". In: *arXiv preprint arXiv:2303.08774* (2023).

[7] Xavier Amatriain. "Prompt design and engineering: Introduction and advanced methods". In: *arXiv preprint arXiv:2401.14423* (2024).

[8] Rohan Anil et al. "Palm 2 technical report". In: *arXiv preprint arXiv:2305.10403* (2023).

[9] *Artificial Intelligence Ontology*. Accessed: 26-11-2024. URL: https://berkeleybop.github.io/artificial-intelligence-ontology/.

[10] Jack Bandy and Nicholas Vincent. "Addressing" documentation debt" in machine learning research: A retrospective datasheet for bookcorpus". In: *arXiv preprint arXiv:2105.05241* (2021).

[11] Paul Barham et al. "Pathways: Asynchronous distributed dataflow for ml". In: *Proceedings of Machine Learning and Systems* 4 (2022), pp. 430–449.

[12] *BIMERR project*. Accessed: 14-10-2024. URL: https://bimerr.iot.linkeddata.es/.

[13] Stefano Borgo et al. "DOLCE: A descriptive ontology for linguistic and cognitive engineering". In: *Applied ontology* 17.1 (2022), pp. 45–69.

[14] Nefeli Bountouni et al. "The BIMERR Interoperability Framework: Towards BIM Enabled Interoperability in the Construction Sector". In: *ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction*. Vol. 38. IAARC Publications. 2021, pp. 94–101.

[15] Ștefan Busnatu et al. "Clinical applications of artificial intelligence—an updated overview". In: *Journal of clinical medicine* 11.8 (2022), p. 2265.

[16] Giovanni Canfora, Daniela Di Fatta, and G Pilatoi. "Ontologie e Linguaggi Ontologici per il Web Semantico". In: *Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni* (2004).

[17] Banghao Chen et al. "Unleashing the potential of prompt engineering in Large Language Models: a comprehensive review". In: *arXiv preprint arXiv:2310.14735* (2023).

[18] Wenhu Chen et al. "Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks". In: *arXiv preprint arXiv:2211.12588* (2022).

[19] *Ciudad abiertas*. Accessed: 14-10-2024. URL: https://ciudadesabiertas.es/vocabularios/#Descripci%C3%B3n.

[20] Preetha Datta et al. "Construction of Hyper-Relational Knowledge Graphs Using Pre-Trained Large Language Models". In: *arXiv preprint arXiv:2403.11786* (2024).

[21] *Decoder-Based Large Language Models: A Complete Guide*. Accessed: 23-10-2024. URL: https://www.unite.ai/decoder-based-large-language-models-a-complete-guide/.

[22] Yihe Deng et al. "Rephrase and respond: Let large language models ask better questions for themselves". In: *arXiv preprint arXiv:2311.04205* (2023).

[23] *Detailed Scratchpad Prompting: LLMs As Interpreters*. Accessed: 30-09-2024. URL: https://okarthikb.github.io/site/blog/detailed-prompting.html.

[24] *Draw a Person Using Alphabet Letters*. Accessed: 15-10-2024. URL: https://www.promptingguide.ai/prompts/image-generation/alphabet-person.

[25] Michael Erdmann and Walter Waterfeld. "Overview of the neon toolkit". In: *Ontology Engineering in a Networked World*. Springer, 2011, pp. 281–301.

[26] Paola Espinoza-Arias, María Poveda-Villalón, and Oscar Corcho. "Using LOT methodology to develop a noise pollution ontology: a Spanish use case". In: *Journal of Ambient Intelligence and Humanized Computing* 11.11 (2020), pp. 4557–4568.

[27] *Everything You Need to Know About GPT-4, GPT-4 Turbo, and GPT-4o*. Accessed: 28-10-2024. URL: https://medium.com/@tarekalkhatibb/everything-you-need-to-know-about-gpt-4-gpt-4-turbo-and-gpt-4o-5b607399c58e.

[28] Ethan Ewer et al. "ENTP: Encoder-only Next Token Prediction". In: *arXiv preprint arXiv:2410.01600* (2024).

[29] *Exploring BLOOM: A Comprehensive Guide to the Multilingual Large Language Model*. Accessed: 24-10-2024. URL: https://www.datacamp.com/blog/exploring-bloom-guide-to-multilingual-llm.

[30] *FAIR principles and Semantics on the Web: where is the meeting point?* Accessed: 14-10-2024. URL: https://joinup.ec.europa.eu/collection/oeg-upm/news/fair-ontologies.

[31] R de A Falbo et al. "Ontology patterns: clarifying concepts and terminology". In: *Proceedings of the 4th Workshop on Ontology and Semantic Web Patterns*. Vol. 1188. 2013.

[32] Gilles Falquet et al. "An introduction to ontologies and ontology engineering". In: *Ontologies in Urban development projects* (2011), pp. 9–38.

[33] Nadeen Fathallah et al. "NeOn-GPT: A Large Language Model-Powered Pipeline for Ontology Learning". In: *The Extended Semantic Web Conference*. 2024.

[34] Mariano FernAndez, AsunciOn Gomez-Perez, and NMETHONTOLOGY Juristo. "From ontological art towards ontological engineering". In: *Proceedings of the Spring Symposium Series on Ontological Engineering (AAAI'97), AAAI Press*. 1997.

[35] Otilia Fodor, Roxana Iosif, and Sorin Vintila. "An Ontological Approach for Tourism Domain". In: *Informatica Economica* 17.3 (2013), pp. 69–82. URL: https://www.researchgate.net/publication/274077474_An_Ontological_Approach_for_Tourism_Domain.

[36] Aldo Gangemi and Valentina Presutti. "Ontology design patterns". In: *Handbook on ontologies*. Springer, 2009, pp. 221–243.

[37] Birte Glimm et al. "HermiT: an OWL 2 reasoner". In: *Journal of automated reasoning* 53 (2014), pp. 245–269.

[38] Asunción Gómez-Pérez, Oscar Corcho, and Mariano Fernández-López. "OntoWeb: A Semantic Web Community Portal". In: *IEEE Intelligent Systems* 17.2 (2002), pp. 38–45. DOI: 10.1109/5254.988457. URL: https://ieeexplore.ieee.org/document/988457.

[39] Muhammad Usman Hadi et al. "A survey on large language models: Applications, challenges, limitations, and practical usage". In: *Authorea Preprints* (2023).

[40] Pascal Hitzler, Aldo Gangemi, and Krzysztof Janowicz. *Ontology engineering with ontology design patterns: foundations and applications*. Vol. 25. IOS Press, 2016.

[41] *Introducing Gemini: our largest and most capable AI model*. Accessed: 28-10-2024. URL: https://blog.google/technology/ai/google-gemini-ai/.

[42] *Introducing Meta Llama 3: The most capable openly available LLM to date*. Accessed: 24-10-2024. URL: https://ai.meta.com/blog/meta-llama-3/.

[43] *Introducing PaLM 2*. Accessed: 24-10-2024. URL: https://cloud.google.com/blog/topics/healthcare-life-sciences/sharing-google-med-palm-2-medical-large-language-model.

[44] Albert Q Jiang et al. "Mistral 7B". In: *arXiv preprint arXiv:2310.06825* (2023).

[45] Andreas Kaplan and Michael Haenlein. "Siri, Siri, in my hand: Who's the fairest in the land? On the interpretations, illustrations, and implications of artificial intelligence". In: *Business horizons* 62.1 (2019), pp. 15–25.

[46] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *Proceedings of naacL-HLT*. Vol. 1. Minneapolis, Minnesota. 2019, p. 2.

[47] Vamsi Krishna Kommineni, Birgitta König-Ries, and Sheeba Samuel. "From human experts to machines: An LLM supported approach to ontology and knowledge graph construction". In: *arXiv preprint arXiv:2403.08345* (2024).

[48] Birger Lantow. "OntoMetrics: Putting Metrics into Use for Ontology Evaluation." In: *KEOD*. 2016, pp. 186–191.

[49] Teven Le Scao et al. "Bloom: A 176b-parameter open-access multilingual language model". In: (2023).

[50] *Lewis Carol quotes*. Accessed: 28-11-2024. URL: https://www.goodreads.com/quotes/9467-alice-laughed-there-s-no-use-trying-she-said-one-can-t.

[51] Pengfei Liu et al. "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing". In: *ACM Computing Surveys* 55.9 (2023), pp. 1–35.

[52] *LLM Prompt Engineering for Beginners: What It Is and How to Get Started*. Accessed: 27-09-2024. URL: https://medium.com/thedeephub/llm-prompt-engineering-for-beginners-what-it-is-and-how-to-get-started-0c1b483d5d4f.

[53] *LOT-resources*. Accessed: 14-10-2024. URL: https://github.com/oeg-upm/LOT-resources.

[54] Yuri Malheiros, Fred Freitas, and Rio Tinto-PB-Brazil. "A Method to Develop Description Logic Ontologies Iteratively Based on Competency Questions: an Implementation." In: *ONTOBRAS* 1041 (2013), pp. 142–153.

[55] *Master 10 Top Negative Prompts: Examples and Develop Guide*. Accessed: 15-10-2024. URL: https://medium.com/@marketing_novita.ai/master-10-top-negative-prompts-examples-and-develop-guide-3a22791aee30#:~:text=Negative%20prompts%20are%20powerful%20tools,%2C%20content%20creators%2C%20and%20developers..

[56] Drew McDermott. "A Temporal Logic for Reasoning about Processes and Plans". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 1982, pp. 251–256. URL: https://dl.acm.org/doi/10.5555/1623411.1623476.

[57] Fanxu Meng et al. "Chain of Images for Intuitively Reasoning". In: *arXiv preprint arXiv:2311.09241* (2023).

[58] Shervin Minaee et al. "Large language models: A survey". In: *arXiv preprint arXiv:2402.06196* (2024).

[59] Navapat Nananukul and Mayank Kejriwal. "HALO: an ontology for representing and categorizing hallucinations in large language models". In: *Disruptive Technologies in Information Sciences VIII*. Vol. 13058. SPIE. 2024, pp. 86–100.

[60] Sanaz Saki Norouzi et al. "Ontology Population using LLMs". In: *arXiv preprint arXiv:2411.01612* (2024).

[61] Maxwell Nye et al. "Show your work: Scratchpads for intermediate computation with language models". In: *arXiv preprint arXiv:2112.00114* (2021).

[62] *Ontology design patterns*. Accessed: 07-02-2024. URL: http://ontologydesignpatterns.org/wiki/Main_Page.

[63] *Ontology Development 101: A Guide to Creating Your First Ontology*. Accessed: 23-09-2024. URL: https://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html.

[64] *Ontology Language*. Accessed: 13-02-2024. URL: https://en.wikipedia.org/wiki/Ontology_language.

[65] Jonas Oppenlaender. "A taxonomy of prompt modifiers for text-to-image generation". In: *Behaviour & Information Technology* (2023), pp. 1–14.

[66] María Poveda-Villalón, Asunción Gómez-Pérez, and Mari Carmen Suárez-Figueroa. "Oops!(ontology pitfall scanner!): An on-line tool for ontology evaluation". In: *International Journal on Semantic Web and Information Systems (IJSWIS)* 10.2 (2014), pp. 7–34.

[67] María Poveda-Villalón et al. "LOT: An industrial oriented ontology engineering framework". In: *Engineering Applications of Artificial Intelligence* 111 (2022), p. 104755.

[68] Valentina Presutti et al. "eXtreme design with content ontology design patterns". In: *Proc. Workshop on Ontology Patterns*. 2009, pp. 83–97.

[69] *Prompt Engineering: una guida all'interrogazione efficace degli LLM*. Accessed: 20-09-2024. URL: https://www.diariodiunanalista.it/posts/guida-prompt-engineering/.

[70] *Protege software*. Accessed: 17-10-2024. URL: https://protege.stanford.edu/.

[71] Alec Radford et al. "Language models are unsupervised multitask learners". In: *OpenAI blog* 1.8 (2019), p. 9.

[72] Colin Raffel et al. "Exploring the limits of transfer learning with a unified text-to-text transformer". In: *Journal of machine learning research* 21.140 (2020), pp. 1–67.

[73] Hanoona Rasheed et al. "Glamm: Pixel grounding large multimodal model". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 13009–13018.

[74] *Real Life Examples of Artificial Intelligence*. Accessed: 20-09-2024. URL: https://www.ironhack.com/gb/blog/real-life-examples-of-artificial-intelligence.

[75] Mike de Roode et al. "SAREF4INMA: A SAREF extension for the industry and manufacturing domain". In: *Semantic Web* 11.6 (2020), pp. 911–926.

[76] Edna Ruckhaus et al. "Applying the LOT methodology to a public bus transport ontology aligned with transmodel: Challenges and results". In: *Semantic Web* 14.4 (2023), pp. 639–657.

[77] Chanatip Saetia et al. "Financial Product Ontology Population with Large Language Models". In: *Proceedings of TextGraphs-17: Graph-based Methods for Natural Language Processing*. 2024, pp. 53–60.

[78] Pranab Sahoo et al. "A systematic survey of prompt engineering in large language models: Techniques and applications". In: *arXiv preprint arXiv:2402.07927* (2024).

[79] Cogan Shimizu, Quinn Hirt, and Pascal Hitzler. "MODL: a modular ontology design library". In: *arXiv preprint arXiv:1904.05405* (2019).

[80] *Style Modifiers*. Accessed: 15-10-2024. URL: https://learnprompting.org/docs/image_prompting/style_modifiers.

[81] Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez, and Mariano Fernández-López. "The NeOn methodology for ontology engineering". In: *Ontology engineering in a networked world*. Springer, 2011, pp. 9–34.

[82] Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez, and Boris Villazón-Terrazas. "How to write and use the ontology requirements specification document". In: *On the Move to Meaningful Internet Systems: OTM 2009: Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009, Vilamoura, Portugal, November 1-6, 2009, Proceedings, Part II*. Springer. 2009, pp. 966–982.

[83] *Submissions:Description*. Accessed: 21-12-2024. URL: http://ontologydesignpatterns.org/wiki/Submissions:Description.

[84] *Submissions:Sequence*. Accessed: 21-12-2024. URL: http://ontologydesignpatterns.org/wiki/Submissions:Sequence.

[85] *Submissions:TaskExecution*. Accessed: 21-12-2024. URL: http://ontologydesignpatterr org/wiki/Submissions:TaskExecution.

[86] *SWRL*. Accessed: 12-02-2024. URL: https://en.wikipedia.org/wiki/Semantic_Web_Rule_Language.

[87] *The History and Role of Voice Assistants in Smart Homes*. Accessed: 20-09-2024. URL: https://tcaflisch.medium.com/the-history-and-role-of-voice-assistants-in-smart-homes-3cf3cd43467a.

[88] A Vaswani. "Attention is all you need". In: *Advances in Neural Information Processing Systems* (2017).

[89] Shakti N Wadekar et al. "The Evolution of Multimodal Model Architectures". In: *arXiv preprint arXiv:2405.17927* (2024).

[90] Xuezhi Wang et al. "Self-consistency improves chain of thought reasoning in language models". In: *arXiv preprint arXiv:2203.11171* (2022).

[91] *Weighted Terms*. Accessed: 17-10-2024. URL: https://learnprompting.org/docs/image_prompting/weighted_terms.

[92] *What is ChatGPT? The world's most popular AI chatbot explained*. Accessed: 20-09-2024. URL: https://www.zdnet.com/article/what-is-chatgpt-the-worlds-most-popular-ai-chatbot-explained/.

[93] *What is hyper-relational knowledge graph?* Accessed: 26-09-2024. URL: https://typeset.io/questions/what-is-hyper-relational-knowledge-graph-u1ofepnvz4.

[94] *What is Permaculture?* Accessed: 28-11-2024. URL: https://extension.usu.edu/permaculture/what-is-permaculture.

[95] Sam Witteveen and Martin Andrews. "Investigating prompt engineering in diffusion models". In: *arXiv preprint arXiv:2211.15462* (2022).

[96] Hongbin Ye et al. "Cognitive mirage: A review of hallucinations in large language models". In: *arXiv preprint arXiv:2309.06794* (2023).

[97] Norlia M Yusof and Shahrul Azman M Noah. "Malaysian food composition ontology evaluation". In: *International Journal of Machine Learning and Computing* 9.5 (2019), pp. 700–705.

[98] Zhuosheng Zhang et al. "Multimodal chain-of-thought reasoning in language models". In: *arXiv preprint arXiv:2302.00923* (2023).

[99] Wayne Xin Zhao et al. "A survey of large language models". In: *arXiv preprint arXiv:2303.18223* (2023).

[100] Huaixiu Steven Zheng et al. "Take a step back: Evoking reasoning via abstraction in large language models". In: *arXiv preprint arXiv:2310.06117* (2023).

Sì nu artìst!

Ruvo di Puglia, 23 febbraio 2025