Pitfalls in AI-Generated Ontologies: Strategies for Detection and Mitigation

Pasquale Lisena[1,*], Julien Plu[2], Oscar Moreno Escobar[2], Edouard Trouillez[2] and Raphaël Troncy[1]

[1]EURECOM, Sophia Antipolis, France
[2]LettrIA, Paris, France

Abstract

Detecting pitfalls in ontologies aim to identify modeling patterns that can lead to inconsistencies, redundancy, or poor semantic design. Ontology design patterns have been proposed in the community and a number of tools and libraries rely on them to detect potential issues occurring during ontology modeling, but they were largely developed before the widespread use of large language models (LLMs) in ontology engineering. As a result, they do not capture a new class of artifacts that frequently appear in LLM-generated ontologies. In this paper, we introduce a complementary set of ontology pitfalls targeting patterns that are not necessary produced by human ontology engineers but commonly arise from LLM-assisted ontology generation. These include semantic redundancies, hierarchy conflicts, and property modeling artifacts. We implement detection strategies combining structural analysis, lexical similarity, and LLM-based evaluation. We evaluate the proposed approach on two ontologies generated by the Ontology Toolkit ontology generation system, showing how the new pitfalls help identify modeling issues specific to LLM-generated ontologies. Our code and data are released at https://github.com/D2KLab/Ontology-Pitfalls-Detector.

Keywords

Large Language Model, Ontology Engineering, Generative AI, LLM Evaluation, Ontology Evaluation

1. Introduction

Ontologies play a central role in knowledge graphs, providing structured vocabularies that support data integration, reasoning, and interoperability. However, designing high-quality ontologies is a complex task that requires careful modeling decisions. Initiatives such as LOV [1] provide curated collections of ontologies to promote ontology re-use. The Ontology Design Pattern (ODP) community offers reusable modeling solutions through the *Ontology Design Pattern Repository*[1]. In the biomedical domain, the *OBO Foundry* [2] provides a coordinated suite of interoperable ontologies that follow shared principles for openness and collaboration. Similarly, platforms such as *OntoPortal* [3] offer infrastructures for publishing, sharing, and exploring ontologies across multiple domains. These initiatives highlight the importance of high-quality ontologies and the need for systematic approaches to their development and evaluation.

Ontology engineering remains a complex and time-consuming process that often requires significant domain expertise and modeling experience. As a result, there has been growing interest in automated or semi-automated approaches for ontology construction, particularly through ontology learning techniques that extract concepts and relations from textual data [4, 5]. Large language models (LLMs) have started to play an increasing role in ontology engineering workflows, assisting in tasks such as ontology bootstrapping, schema generation, and knowledge graph construction [6]. While these models significantly accelerate ontology creation, they also introduce new types of modeling artifacts that are

Workshop on LLM-driven Knowledge Graph and Ontology Engineering (llms4kgoe), May 11th, 2026, Dubrovnik , Croatia
*Corresponding author.
pasquale.lisena@eurecom.fr (P. Lisena); julien@lettria.com (J. Plu); oscar@lettria.com (O. M. Escobar); edouard@lettria.com (E. Trouillez); raphael.troncy@eurecom.fr (R. Troncy)
0000-0003-3094-5585 (P. Lisena); 0000-0002-7876-3441 (J. Plu); 0000-0003-0457-1436 (R. Troncy)

[1]https://github.com/odpa/patterns-repository

rarely produced by human modelers. Examples include subtle semantic redundancies, inconsistent hierarchies, and property modeling patterns that emerge from the generative nature of LLMs.

Over the years, several studies [7, 8, 9] have identified recurring modeling mistakes or *pitfalls* that may lead to inconsistencies, redundancy, or poor semantic design. To support ontology engineers, various tools and libraries have been proposed to automatically detect such pitfalls through structural analysis, lexical heuristics, or logical reasoning, the most famous tool being the Oops! scanner[2]. However, some existing pitfall detection approaches have been developed in contexts where ontologies were primarily authored by human experts. Others assume a high level of expertise from generative systems, which is often lacking. As a consequence, new pitfalls might exist within generated ontologies but remain undetected by these automated evaluation tools.

In this paper, we address this challenge by proposing a new tool and method for detecting and analyzing these new ontology modeling pitfalls in automatically generated ontologies. In particular, we introduce a library for identifying structural, logical, naming, and semantic issues in ontologies. The library implements a collection of detection patterns that capture recurring modeling problems frequently observed in automatically generated ontologies. In addition, we present *Ontology Toolkit*, a framework designed to support the generation of ontologies from textual data using large language models. The toolkit provides a modular pipeline that facilitates the transformation of textual inputs into structured ontology components and integrates quality assessment mechanisms during the generation process.

The remainder of this paper is organised as follows. We review some related work in Section 2. We present the Ontology Toolkit framework in Section 3. In Section 4, we detail the Ontology Pitfalls Detector library, whose results are reported in Section 5. Finally, we conclude and outline some future work in Section 6.

2. Related work

2.1. Ontology Pitfall Detection

Ontology pitfall detection has been widely studied as a means to support ontology engineering and improve ontology quality. Pitfalls correspond to recurrent modeling patterns that may lead to inconsistencies, redundancy, or poor conceptual design. Early work in this area introduced catalogs of common ontology pitfalls and methods for their automatic detection. Several tools and libraries have been developed to operationalize these catalogs, such as OOPS! (Ontology Pitfall Scanner) [7]. OOPS! relies on a list of pitfalls that capture recurrent issues observed in ontology engineering, such as missing annotations, incorrect or missing domain and range declarations, improper class hierarchies, or missing inverse properties, categorized according to their severity. OOPS! performs its analysis through a combination of structural checks over the ontology graph, lexical analysis of class and property names, and logical reasoning over OWL axioms.

FOOPS! [10] is an extension of the OOPS! framework designed to assess the quality and FAIRness of ontologies according to the FAIR principles (Findable, Accessible, Interoperable, and Reusable). It evaluates ontologies against a set of FAIR indicators and provides automated feedback to help ontology developers improve metadata, documentation, and interoperability practices. OntoConnectLM [9] includes a common interface to interrogate both OOPS! and FOOPS!.

In [8], generated ontologies are compared to human-realised counterparts using structural semantic and task-based metrics. This demonstrates partial alignment and good performance although they tend to offer a flatter structure with lesser hierarchal depth.

2.2. The Limits of Direct Prompting and Unconstrained Generation

Recent research in LLM-driven ontology engineering reveals a clear dichotomy between requirement-guided prompting and structured, pipeline-driven extraction. Representing the former paradigm,

[2] https://oops.linkeddata.es/aboutus.jsp

Lippolis et al. [11] attempt to generate OWL ontologies directly from Competency Questions (CQs) using advanced prompting strategies (Memoryless CQbyCQ and Ontogenia). While their approach matches novice human performance, structural evaluations highlight persistent flaws inherent to unconstrained LLM generation, including superfluous elements, flat hierarchies, and inconsistent domain/range axioms. In contrast, our methodology replaces direct OWL generation with a deterministically constrained, multi-stage pipeline. By extracting a semantic network from unstructured text through a strict use-case filter, our approach enforces structural integrity via intermediate, auto-validated schemas equipped with error-correction loops. Furthermore, our pipeline programmatically aligns all extracted classes to a rigid seven-category upper ontology and explicitly reifies complex relations. By compartmentalizing extraction, hierarchical structuring, and formal serialization into isolated stages, our approach systematically prevents the taxonomic errors and structural anomalies that current prompt-based techniques struggle to eliminate.

2.3. From Reactive Debugging to Soundness by Design

Even when multi-stage pipelines are employed to mitigate the aforementioned structural flaws, existing approaches often depend heavily on reactive, external debugging. For instance, frameworks like NeOn-GPT [12] operationalize established ontology engineering methodologies by guiding models through competency question generation, domain reuse, and formalization. However, to ensure logical viability, NeOn-GPT relies on a post-hoc verification loop, coupling the LLM with external reasoners and evaluation tools (e.g., HermiT, OOPS!) to iteratively debug the generated Turtle drafts. In contrast, our proposed methodology ensures ontological soundness by design rather than through external debugging. We decouple conceptualization from formalization by enforcing a strict progression of schema-validated intermediate representations (YAML-based semantic networks and conceptual definitions). Furthermore, while NeOn-GPT anchors its models using domain-specific reuse fragments, our approach guarantees structural integrity by rooting the taxonomy in a rigid foundational upper ontology (e.g., MATERIAL, AGENT) and proactively prevents logical inconsistencies by enforcing the explicit reification of complex relations and deliberately excluding error-prone, blank-node OWL restrictions.

2.4. Complementary Paradigms: Generative Construction vs. Validation

Broadly, the recent advancements in LLM-driven ontology engineering fall into two complementary paradigms: generative construction and post-hoc validation. Our methodology focuses on the former, introducing a deterministic pipeline that transforms unstructured text into logically enriched OWL/RDFS ontologies. By enforcing strict schema validation and anchoring extractions to a foundational upper ontology, our approach proactively prevents structural anomalies during the initial modeling phase. Conversely, recent work by Choi et al. [13] addresses the validation phase via the VSPO framework, which fine-tunes an LLM to generate Competency Questions designed to expose "semantic pitfalls"—such as missing or misused TBox axioms—in existing ontologies. While VSPO operates reactively to diagnose misalignments between natural language definitions and formal axioms, our pipeline proactively couples them from the start. Ultimately, these methods address distinct bottlenecks: our pipeline automates the complex task of initial knowledge acquisition and formalization, whereas frameworks like VSPO provide the necessary diagnostic tools to audit and verify those independently generated structures.

3. Ontology Toolkit

Our proposed ontology generation methodology consists of a six-stage pipeline depicted in Figure 3 designed to transform unstructured, multilingual text into a formally structured, logically enriched OWL/RDFS ontology in the Turtle (TTL) syntax. The approach leverages a configurable state-of-the-art Large Language Model (LLM) configured with zero-temperature settings to ensure deterministic, highly reproducible outputs [14].

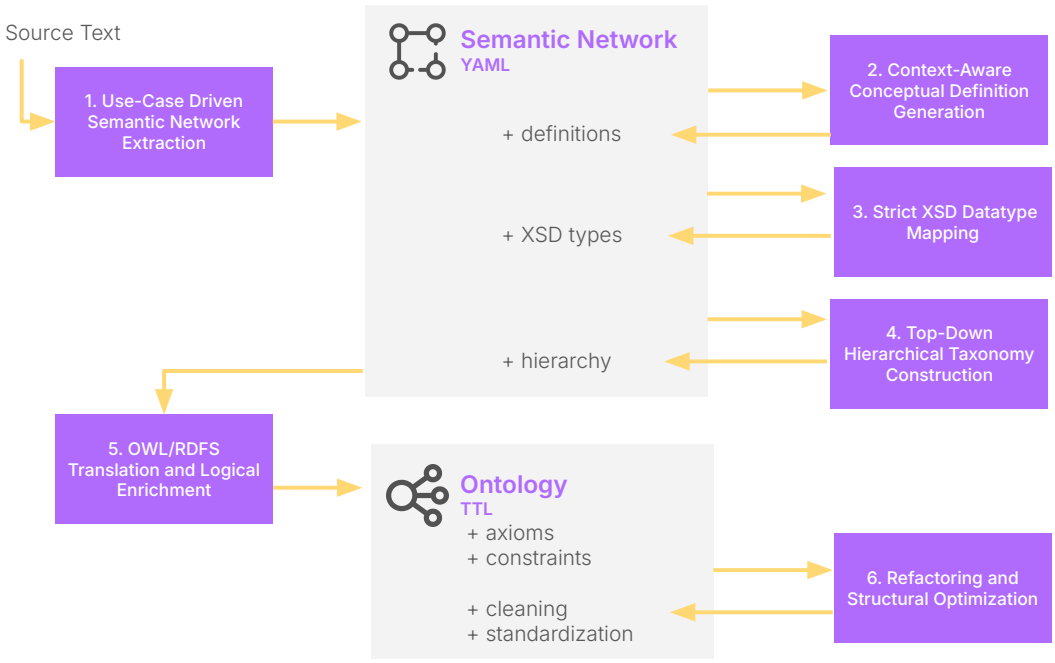

Figure 1: Diagram of the Ontology Toolkit six-stage pipeline

To bridge the gap between unstructured text and formal semantics, the pipeline is strictly use-case driven. A user-defined "use case" acts as a relevance filter throughout all stages, ensuring that only domain-salient entities, attributes, and relationships are extracted and modeled. Furthermore, to guarantee syntactic and structural integrity, each stage of the pipeline employs strict schema validation. Outputs failing validation are automatically fed back into the model along with explicit error diagnostics for iterative correction.

3.1. Stage 1: Use-Case Driven Semantic Network Extraction

The first stage processes the source text to extract a foundational semantic network. The extraction adheres to strict ontological principles rather than simple Information Extraction (IE) rules:

- **Entity and Type Resolution:** Entities are extracted and categorized based on three scenarios: explicit typing present in the text, inferred typing using domain knowledge (Named Entity Recognition), and non instantiated types (classes mentioned without specific instances). Suffixes indicating meta-classifications (e.g., _TYPE, _CATEGORY) are strictly prohibited to enforce concrete ontological typing.
- **Relationship Normalization:** Relationships between entities are extracted as directed edges. To ensure timeless ontological statements, all relations are strictly normalized to the present tense (e.g., "was founded by" becomes IS_FOUNDED_BY), with temporal aspects delegated to specific attributes.
- **Reification of Complex Constructs:** To capture advanced semantics, complex phenomena are actively reified into independent entities. This includes n-ary relations (e.g., events such as Appointments, Acquisitions) and complex quantitative values (values and units). Any measurement possessing a numeric value, a unit, or a currency is reified into a dedicated node linked via distinct HAS_VALUE, HAS_UNIT, or HAS_CURRENCY properties.
- **Temporal and Range Normalization:** Standardized formatting is enforced for all literals, such as ISO 8601 for durations, ISO 4217 for currencies, and standardized bounding models for numerical ranges.

3.2. Stage 2: Context-Aware Conceptual Definition Generation

To ensure the resulting ontology is self-documenting and interpretable, the second stage generates purely conceptual, human-readable definitions for every unique class, relationship, and property identified in the network.

- **Contextual Derivation:** The model analyzes the topological context of the semantic network to infer the precise meaning of each element. Definitions are constrained to explain the real-world concept rather than the graph mechanics (e.g., defining FOUNDING as "The pivotal event that marks the establishment of an organization," rather than "Connects a Person to a Company").
- **Iterative Reconciliation:** An automated reconciliation mechanism cross-references the generated definitions against the semantic network. Any nodes or edges lacking definitions trigger a recovery loop, which analyzes the surrounding network structure and grammatical directionality to infer and generate the missing semantic definitions.

3.3. Stage 3: Strict XSD Datatype Mapping

In the third stage, the pipeline analyzes all literal attributes to map their informal semantic types to strict XML Schema Definition (XSD 1.1) built-in datatypes.

- **Holistic Analysis:** The mapping relies on the synthesis of the attribute's semantic intent and its concrete value.
- **Principle of Maximum Specificity:** The system enforces the most restrictive valid datatype. For instance, identifiers are mapped to TOKEN rather than generic STRINGs; financial figures are strictly mapped to DECIMAL to preserve precision; and chronological data are mapped to explicit temporal types (e.g., GYEAR, DATE).

3.4. Stage 4: Top-Down Hierarchical Taxonomy Construction

This stage constructs a formal "sub class of" (subsumption) hierarchy for all extracted entity types and non instantiated classes.

- **Upper Ontology Alignment:** The taxonomy is strictly rooted in a foundational upper ontology comprising seven immutable top-level categories derived from established ontological frameworks: MATERIAL, AGENT, PROCESS, ABSTRACT_ENTITY, SPATIAL_REGION, TEMPORAL_REGION, and QUALITY. Every extracted class must eventually trace back to one of these foundational roots.
- **Intermediate Subclass Generation:** The system dynamically infers logical intermediate classes to ensure a balanced, multi-level hierarchy, avoiding overly flat structures while strictly prohibiting domain-based grouping wrappers (e.g., rejecting LEGAL_CONCEPT in favor of specific ontological classes like NORMATIVE_ARTIFACT or LEGAL_PROCEEDING).
- **Canonicalization:** True synonyms detected during the extraction phase are merged into a single canonical class, with the alternative terms retained as additional labels.

3.5. Stage 5: OWL/RDFS Translation and Logical Enrichment

The integrated YAML structures are then translated into a formal OWL/RDFS ontology serialized in Turtle (TTL). Beyond syntactic translation, this stage applies advanced logical enrichment:

- **Axiom Inference:** The system infers and injects rich logical axioms. Sibling classes under the same parent that represent fundamentally different concepts are declared mutually disjoint (`owl:disjointWith`).
- **Property Characteristics:** Object properties are augmented with semantic characteristics based on their conceptual definitions, including `owl:FunctionalProperty`, `owl:InverseFunctionalProperty`, `owl:SymmetricProperty`, `owl:TransitiveProperty`, and explicit inverses (`owl:inverseOf`).

- **Multilingual Support:** If target languages are specified, the pipeline generates parallel bilingual `rdfs:label` and `rdfs:comment` annotations, maintaining English as the baseline while providing localized definitions.
- **Restriction Strategy:** To maintain high computational tractability and reasoner compatibility, the methodology intentionally avoids complex blank-node `owl:Restriction` axioms in favor of explicit, named class relations and global property domains/ranges.

3.6. Stage 6: Refactoring and Structural Optimization

The final stage acts as an automated ontology engineer, performing a comprehensive review and structural optimization of the generated OWL code.

- **Logical Integrity Check:** The hierarchy is checked for semantic consistency, ensuring no circular dependencies exist and resolving instances of unnecessary multiple inheritance.
- **Syntax Standardization:** The Turtle syntax is refactored for human readability. This includes standardizing URI naming conventions (UpperCamelCase for classes, lowerCamelCase for properties), collapsing verbose blank nodes into concise anonymous node syntax, and logically grouping all triples relating to a single subject into cohesive blocks.

4. Ontology Pitfalls Detector

To support the systematic evaluation of automatically generated ontologies, we developed the *Ontology Pitfalls Detector*, an open-source library that identifies common modeling pitfalls occurring in ontology engineering. The library is open sourced at https://github.com/D2KLab/Ontology-Pitfalls-Detector. Its objective is to provide a practical and extensible framework for analyzing the quality of ontologies, with a particular focus on ontologies generated automatically through ontology learning pipelines, knowledge extraction workflows, or large language model–based generation systems.

Automatically produced ontologies frequently exhibit recurring modeling issues. These issues can affect the logical consistency of the ontology, reduce its interpretability, or introduce unnecessary redundancy in its vocabulary and structure. While such problems may not always prevent the ontology from being used, they can significantly limit its usefulness for reasoning, data integration, and knowledge reuse. The Ontology Pitfalls Detector addresses this problem by implementing a collection of reusable detection patterns designed to identify these issues in a systematic and automated way.

The library operates on ontologies expressed in RDF and OWL and performs a series of analyses on the ontology graph. It extracts classes, properties, and axioms from the ontology and applies a set of detection patterns corresponding to ontology pitfalls that have been manually observed by an ontology engineer in ontologies generated by LLMs. Depending on the nature of the pattern, different techniques are employed: this includes SPARQL queries over the ontology graph, lexical similarity measures, reasoning-based checks, and semantic evaluation using large language models as a judge. This hybrid approach allows the framework to address both formally detectable problems and more subtle semantic issues that cannot be identified through structural analysis alone. The pitfalls implemented in the library are organized into four main categories, as shown in Table 4. The table shows some overlaps with the OOPS! library [7], but most of our patterns extend, reverse or refine existing categories rather than fully matching them. This library is intended to be an addition to existing tools like OOPS! rather than an alternative or a replacement.

4.1. Logical Issues

Logical issues correspond to formal inconsistencies or problematic logical axioms that may affect reasoning over the ontology.

P1.1 Parent disjoint with child. This occurs when a superclass is declared `owl:disjointWith` one of its subclasses, creating a logical contradiction. *Example:* The class `Animal` is disjoint with

Table 1
Summary of ontology pitfalls detected by the Ontology Pitfalls Detector. The table reports the pitfall code, a short description, the detection method, and the closest corresponding OOPS! pattern.

Code	Pitfall	Detection Method	Related OOPS! Pitfall
Logical Issues			
P1.1	Parent disjoint with child	SPARQL pattern	P10 – Missing disjointness
P1.2	Subclass of both parent and grandparent	SPARQL pattern	–
P1.3	Logical inconsistencies	OWL reasoner	–
Structural Issues			
P2.1	Not connected hierarchies	Lexical similarity	P04 – Creating unconnected ontology elements
P2.2	Single subclass parent	Hierarchy analysis	P17 – Over-specialized hierarchy
P2.3	Superfluous disjointness	Structural filter + LLM	P10 – Missing disjointness
P2.4	Single subproperty parent	Hierarchy analysis	P17 – Over-specialized hierarchy
P2.5	Range / domain expansion	Hierarchy analysis + LLM	P19 – Multiple domains or ranges
P2.6	Possible hierarchy among properties	Similarity + domain/range check	–
Redundancy and Naming Issues			
P3.1	Properties replicating RDF vocabulary	String match	P03 – Creating the relationship "is"
P3.2	Range in property title	Substring match	–
P3.3	Domain in property title	Substring match	–
Semantic Issues			
P4.1	Overly generic classes	WordNet depth	P21 – Miscellaneous class
P4.2	Synonyms in classes	Semantic similarity	P02 – Creating synonyms as classes
P4.3	Conflicting hierarchy	Antonym filter + LLM	–
P4.4	Subclass equivalent to superclass	Semantic similarity	P17 – Over-specialized hierarchy
P4.5	Synonyms in properties	Semantic similarity	–
P4.6	Inverse properties not declared	Label similarity + domain/range	P13 – Missing inverse relationships
P4.7	DataProperty that should be ObjectProperty	String similarity + LLM	—

`Mammal`, even though `Mammal` is a subclass of `Animal`. Detection is performed using SPARQL queries that cross-reference subclass and disjointness assertions.

P1.2 Subclass of both parent and grandparent. A class is redundantly declared as a subclass of its immediate parent and its grandparent, creating unnecessary hierarchy redundancy. *Example:* `ElectricCar` is declared as a subclass of both `Car` and `Vehicle`, even though `Car` is already a subclass of `Vehicle`. Detected via SPARQL queries analyzing the subclass hierarchy.

P1.3 Logical inconsistencies. These are unsatisfiable classes or contradictions resulting from multiple modeling decisions, that can be performed using an OWL reasoner.

4.2. Structural Issues

Structural issues concern the organization of class and property hierarchies in the ontology.

P2.1 Not connected hierarchies. Semantically related hierarchies are unconnected, which may cause fragmentation of the ontology structure. *Example:* `RenewableEnergyTarget` is not linked to

EnergyTarget, despite clearly being a specialization. Detected using lexical similarity and overlap analysis.

P2.2 Single subclass parent. Classes with only one child may indicate unnecessary abstraction. *Example:* AdvancedCar is the only subclass of CarType, adding a layer without semantic gain. Detected by counting the number of subclasses per class.

P2.3 Superfluous disjointness. Classes are declared disjoint despite potentially overlapping semantics. *Example:* PreeclampsiaRisk disjoint with MaternalRisk, even though the former is a type of the latter. Detected via structural filtering and LLM-assisted semantic evaluation.

P2.4 Single subproperty parent. Properties with only one subproperty may signal unnecessary hierarchy. *Example:* hasPrimaryAuthor is the only subproperty of hasAuthor. Detected by analyzing property hierarchies.

P2.5 Range/domain expansion. Overly specific domain or range definitions reduce property reusability. *Example:* reportsTo defined only for Manager → Executive, instead of the more general Agent → Agent. Detected by checking whether broader superclasses exist and evaluating with LLM.

P2.6 Possible hierarchy among properties. Related properties could form a subproperty relationship. *Example:* hasInclusiveDefinition and hasExclusiveDefinition may benefit from a hierarchy. Detected using semantic similarity and domain/range analysis.

4.3. Redundancy and Naming Issues

These pitfalls relate to vocabulary design and naming conventions.

P3.1 Properties replicating RDF vocabulary. Custom properties duplicate standard RDF terms. *Example:* isA replicates rdf:type. Detected using string matching.

P3.2 Range in property title. Property names embed their range class. *Example:* strategicProgramHasActionPlan includes StrategicProgram. Detected via substring matching.

P3.3 Domain in property title. Property names embed their domain class. *Example:* hasFramework includes Framework. Detected via substring matching.

4.4. Semantic Issues

Semantic issues require interpretation of class and property meaning.

P4.1 Overly generic classes. Abstract classes add little semantic value. *Example:* BusinessConcept or MedicalConcept. Detected via distance of the label (or of its parts) from the WordNet concept *abstraction.n.06* and *entity.n.01* (they are considered abstract when the distance is less than two hops).

P4.2 Synonyms in superclasses. Multiple superclasses represent the same concept. *Example:* Target and Goal used as separate superclasses. Detected via semantic similarity.

P4.3 Conflicting hierarchy. Classes are under semantically incompatible parents. *Example:* SustainabilityOpportunity placed under Risk. Detected using antonym filters and LLM judgment.

P4.4 Subclass equivalent to superclass. Subclasses semantically overlap with their parent. *Example:* MedicalIntervention as subclass of Intervention. Detected via semantic similarity and property domain/range overlap.

P4.5 Synonymous properties. Two properties convey the same relation. *Example:* isA vs isSubTypeOf. Detected via semantic similarity.

P4.6 Inverse properties not declared. Properties express inverse relations without formal declaration. *Example:* describes and describedIn. Detected using semantic similarity and swapped domain/range patterns.

P4.7 DataProperty that should be modeled as ObjectProperty and Classes. Falls in this categories data properties that are strongly related and could instead be modeled using an object

property and an explicit class. *Example:* `hasSurfaceUnit` and `hasSurfaceValue` can be instead modeled with a `hasSurface` object properties, whose connected `Surface` class is on its own linked to the Unit and Value. Detected via string similarity and LLM evaluation.

Overall, the Ontology Pitfalls Detector provides a unified framework for identifying a wide range of ontology modeling issues. By combining graph analysis, reasoning, lexical resources, and semantic evaluation, it facilitates systematic evaluation of automatically generated ontologies and helps ontology engineers improve consistency, reusability, and semantic clarity.

5. Evaluation

Ontologies. We evaluate our system by detecting pitfalls over two ontologies, respectively named CSRD and RED, generated by the Ontology Toolkit framework (Section 3) and available in our repository.

The CSRD ontology is a representative use case coming from regulatory to sustainability reporting. The **Corporate Sustainability Reporting Directive (CSRD)** is a European regulation that requires companies to disclose standardized environmental, social, and governance (ESG) information. The documents used for generating this ontology describe reporting requirements, sustainability indicators, governance structures, and risk management concepts. More specifically, the documents used are: 1) the annual report from 2025 of Bouygues[3], 2) the annual report from 2024 of LVMH[4], 3) the annual report from 2024 of Nestle[5], and 4) the non financial report from 2024 of Nestle[6]. These reports are complex PDF documents that include a rich layout, text, tables, figures and images and amount of hundreds of pages.

The **RED** ontology describes regulatory and environmental disclosure elements related to sustainability strategies, policies, risks, and opportunities. A set of 30 short documents coming from the BBC has been used to generate this ontology.

Both document collections are characterized by a rich conceptual structure and a large number of domain-specific entities, making them suitable sources for ontology construction. In particular, they contain hierarchical concepts, relationships between organizational actors, policies, indicators, and reporting obligations. These characteristics make them appropriate testbeds for evaluating ontology generation systems and for analyzing the types of modeling artifacts that may arise during automated ontology construction. Table 2 reports the statistics of the two ontologies.

Table 2
Basic statistics of the generated ontologies.

Ontology	Classes	Object Properties	Datatype Properties
CSRD	361	162	142
RED	601	332	173

Generation Details. Prior to ontology generation, the source documents are converted from PDF to Markdown using Lettria's *Document Parsing* tool[7] in order to obtain a structured and machine-readable representation of the content. We then perform an in-document segmentation based on document headings, which preserves the original hierarchical structure while producing coherent textual units for processing.

For large documents, we additionally apply a manual segmentation to produce segments of approximately 50 pages. This granularity was chosen to balance the amount of information processed during

[3] https://www.bouygues.com/app/uploads/2026/03/bouygues_deu_2025_fr_web.pdf
[4] https://www.bnains.org/archives/communiques/LVMH/20250325_Document_d_enregistrement_universel_2024_LVMH.pdf
[5] https://www.nestle.com/sites/default/files/2025-02/annual-review-2024-en.pdf
[6] https://www.nestle.com/sites/default/files/2025-02/non-financial-statement-2024.pdf
[7] https://www.lettria.com/features/document-parsing

generation. Segments that are too small tend to cause over-extraction, producing ontologies that are excessively detailed and difficult to manage, while segments that are too large may lead to partial information loss, as relevant concepts can be overlooked.

Different strategies were adopted for the two document collections used in our experiments. For the RED corpus, the 30 documents were first merged into a single source document, from which one ontology was generated using the Ontology Toolkit. In contrast, for the CSRD corpus, an ontology was generated independently for each document chunks, and the resulting ontologies were subsequently merged into a single ontology representing the complete corpus.

Results and Discussion. Table 3 reports the ratio of elements affected by each pitfall, computed as the number of classes or properties involved in a pitfall divided by the total number of generated elements of the corresponding type. The Ontology Pitfalls Detector succeeds in detecting issues related to logic, structure, and semantics in both ontologies.

Overall, the results indicate that the majority of structural and logical issues remain relatively limited. Most detected issues belong to structural or modeling-quality categories rather than strict logical errors. In particular, hierarchy-related patterns such as single-subclass parents (P2.2), not-connected hierarchies (P2.1), and range/domain expansion (P2.5) appear with moderate frequency in both corpora.

Naming-related pitfalls show more variability across datasets. For instance, the repetition of the range in property names (P3.2) is noticeably higher in the CSRD ontology than in the RED ontology, suggesting a stronger tendency toward descriptive property naming in that corpus. Semantic similarity–based issues such as potentially synonymous properties (P4.5) or conflicting hierarchies (P4.3) also appear with moderate ratios, indicating that LLM-generated ontologies may introduce near-duplicate concepts or semantically inconsistent parent–child relations.

It is worth noting that not all detected pitfalls necessarily correspond to actual modeling errors. In some cases, what is flagged as a potential issue may still represent an acceptable modeling choice. For example, a property such as *hasAppointment*, which repeats its range *Appointment* in the property name (P3.2), may be considered reasonable, as using a generic property name such as *has* would arguably be less informative. These situations highlight the limits of purely pattern-based detection and suggest that some pitfalls require contextual interpretation. Future work will aim to better distinguish between genuine modeling problems (e.g. with regular expression and/or *LLM as a judge*) and acceptable design alternatives.

Another aspect to consider is that many of the detectors rely on semantic similarity measures. As a consequence, the same element may be flagged under multiple pitfalls, meaning that the detected pitfalls are not entirely independent from one another. Furthermore, the similarity metric may sometimes overemphasize general lexical or semantic proximity, while overlooking domain-specific distinctions. In practice, this may lead to cases where concepts that are related but not strictly synonymous within the domain are flagged as potential issues. For instance, *Corporate Acquisition* and *Corporate Takeover* are identified as possible synonyms (P4.2), as well as *Business Deal* and *Business Offer*. Future experiments will investigate the role of domain adaptation and fine-tuning in improving the robustness of similarity-based detectors and reducing such domain-misaligned detections.

The overall harmonic mean across pitfalls remains low for both ontologies. This suggests that while several categories of modeling imperfections are present, they affect only a small fraction of the generated entities and therefore do not fundamentally compromise the global structure of the ontologies in this case.

6. Conclusion and Future Work

We presented an extensible library for detecting ontology modeling pitfalls that specifically targets artifacts introduced by LLM-assisted ontology generation. The detector combines structural SPARQL checks, reasoning-based validation, lexical/semantic similarity measures, and LLM-based judgments to capture a broad spectrum of issues (logical, structural, naming, and semantic). We applied the approach

to two ontologies generated by the Ontology Toolkit and observed a good pitfall detection of our generation pipeline.

Limitations of the current study include the narrow corpus scope (two regulatory document sets) and the reliance on LLM judgement in some detectors, which requires careful calibration and human validation. Future work will (i) expand evaluation to additional domains and benchmarks, (ii) perform manual validation of automatically flagged issues, (iii) refine LLM-based thresholds and explainability, and (iv) explore automated repair and continuous integration hooks so that pitfall detection can be integrated directly into ontology-generation pipelines. The library and toolkit are released as open-source to facilitate community validation and extension.

Acknowledgments

This work was supported by the French Public Investment Bank (Bpifrance) i-Demo program within the LettRAGraph project (Grant ID DOS0256163/00).

Declaration on Generative AI

During the preparation of this work, the authors used both GPT-4 and Gemini for grammar and spelling check. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

Table 3
Ratio of detected pitfalls in the latest system, with the harmonic mean across all pitfalls.

ID	Pitfall Description	CSRD	RED	Detected example
P1.1	Parent disjoint with children	0.003	0.005	*Data Provider* is child of *Company* even though they are disjoint
P1.2	Entity as subclass of both parent and grandparent	0.003	0.003	*Court* is subclass of both *Government Body* (parent) and *Government Organization* (grand-parent)
P1.3	Logical inconsistencies	0.000	0.000	n.a.
P2.1	Not connected hierarchies	0.078	0.080	*Resale Policy* is not child of *Policy*
P2.2	Single subclass parent	0.086	0.101	*Building* has only one direct subclass (*Facility*)
P2.3	Superfluous disjointness	0.019	0.033	*Market Movement* and *Stock Price Movement* are disjoint but semantically similar (similarity 0.67)
P2.4	Single subproperty parent	0.030	0.030	*affects* has only one direct subproperty (*affectsCompany*)
P2.5	Range/Domain expansion	0.080	0.054	*initiates* has multiple domains (*Company*, *Person*, *Regulatory Body*) with shared superclasses (*Agent*)
P2.6	Possible hierarchy among properties	0.053	0.042	*isEmployedBy* and *wasEmployedBy* are similar properties and may need a shared parent (similarity 0.86, domain/range match: yes).
P3.1	Properties replicating standard RDF ones	0.000	0.000	n.a.
P3.2	Range in property title	0.241	0.081	*affectsCompany* repeats its range *Company* in the property name
P3.3	Domain in property title	0.000	0.009	*protestsAgainst* repeats its domain *Protest* in the property name
P4.1	Overly generic classes	0.030	0.033	*ConcreteEntity* appears overly generic (distance 0 from top-level concepts)
P4.2	Synonyms in superclasses	0.014	0.010	*Media Organization* and *News Organization* may be synonyms (combined similarity 0.81)
P4.3	Conflicting hierarchy	0.080	0.087	*Asset Protection* may conflict with parent *Wrongdoing* (low similarity). *Harassment* may conflict with parent *Challenge* (opposite polarity).
P4.4	Subclasses with same semantics as superclasses	0.003	0.000	*Governance Body* is semantically very close to parent *Governance Entity* (similarity 0.81)
P4.5	Synonyms in properties	0.059	0.057	*has Child* and *has Children* may be synonymous properties (combined similarity 0.88)
P4.6	Inverse properties not declared	0.003	0.008	*has Investor* and *is Investor* In may be undeclared inverse properties with partially mirrored domain/range (similarity 0.80)
P4.7	DataProperties that can become ObjectProperties	0.035	0.058	*hasRange* overlaps with *hasRangeUnit*
MEAN	Across pitfalls	0.009	0.015	

References

[1] P. Vandenbussche, G. Atemezing, M. Poveda-Villalón, B. Vatant, Linked Open Vocabularies (LOV): A gateway to reusable semantic vocabularies on the Web, Semantic Web 8 (2017) 437–452. doi:10.3233/SW-160213.

[2] R. Jackson, N. Matentzoglu, J. A. Overton, R. Vita, J. P. Balhoff, P. L. Buttigieg, S. Carbon, M. Courtot, A. D. Diehl, D. M. Dooley, W. D. Duncan, N. L. Harris, M. A. Haendel, S. E. Lewis, D. A. Natale, D. Osumi-Sutherland, A. Ruttenberg, L. M. Schriml, B. Smith, C. J. Stoeckert Jr., N. A. Vasilevsky, R. L. Walls, J. Zheng, C. J. Mungall, B. Peters, OBO Foundry in 2021: operationalizing open data principles to evaluate ontologies, Database 2021 (2021). doi:10.1093/database/baab069.

[3] C. Jonquet, J. Graybeal, S. Bouazzouni, M. Dorf, N. Fiore, X. Kechagioglou, T. Redmond, I. Rosati, A. Skrenchuk, J. L. Vendetti, M. Musen, Ontology Repositories and Semantic Artefact Catalogues with the OntoPortal Technology, in: 22nd International Semantic Web Conference (ISWC), Springer, Athens, Greece, 2023, pp. 38–58. doi:10.1007/978-3-031-47243-5_3.

[4] Y. Rebboud, L. Tailhardat, P. Lisena, R. Troncy, Can LLMs generate competency questions?, in: Springer (Ed.), Extended Semantic Web Conference (ESWC), Special Track on Large Language Models for Knowledge Engineering, Hersonissos, Greece, 2024.

[5] J. Plu, O. Moreno Escobar, E. Trouillez, A. Gapin, R. Troncy, A Comprehensive Benchmark for Evaluating LLM-Generated Ontologies, in: CEUR (Ed.), 23rd International Semantic Web Conference (ISWC), Special Session on Harmonising Generative AI and Semantic Web Technologies (HGAIS, Baltimore, USA, 2024.

[6] H. Babaei Giglou, J. D'Souza, N. Mihindukulasooriya, S. Auer, LLMs4OL 2025 Overview: The 2nd Large Language Models for Ontology Learning Challenge, Open Conference Proceedings 6 (2025). doi:10.52825/ocp.v6i.2913.

[7] M. Poveda-Villalón, A. Gómez-Pérez, M. C. Suárez-Figueroa, OOPS! (OntOlogy Pitfall Scanner!): An On-line Tool for Ontology Evaluation, International Journal on Semantic Web and Information Systems (IJSWIS) 10 (2014) 7–34.

[8] M. Llugiqi Rexha, F. Ekaputra, M. Sabou, From Experts to LLMs: Evaluating the Quality of Automatically Generated Ontologies, in: 2nd International Workshop on Evaluation of Language Models in Knowledge Engineering, CEUR, 2025.

[9] E. Amdouni, A. Belfadel, M. Gagnant, J. Kattan, OntoConnectLM : Un outil de génération des ontologies guidé par les LLMs et les connaissances ouvertes, in: 26ème Congérence Francophone sur l'Extration et la Gestion des Conaissance, Anglet, France, 2026. URL: https://hal.science/hal-05470168.

[10] D. Garijo, O. Corcho, M. Poveda-Villalón, FOOPS!: An Ontology Pitfall Scanner for the FAIR Principles, in: International Semantic Web Conference (ISWC), Posters, Demos, and Industry Tracks, volume 2980, CEUR-WS.org, 2021. URL: http://ceur-ws.org/Vol-2980/paper321.pdf.

[11] A. S. Lippolis, M. J. Saeedizade, R. Keskisärkkä, S. Zuppiroli, M. Ceriani, A. Gangemi, E. Blomqvist, A. G. Nuzzolese, Ontology Generation Using Large Language Models, in: European Semantic Web Conference (ESWC), Springer, 2025, pp. 321–341.

[12] N. Fathallah, A. Das, S. De Giorgis, A. Poltronieri, P. Haase, L. Kovriguina, E. Simperl, A. Meroño-Peñuela, S. Staab, A. Algergawy, NeOn-GPT Extended: An Extended Framework for LLM-based Ontology Engineering, https://github.com/NadeenAhmad/neon-gpt-extended, 2025.

[13] H. Choi, S. Hwang, K.-H. Lee, VSPO: Validating Semantic Pitfalls in Ontology via LLM-Based CQ Generation, 2025. URL: https://arxiv.org/abs/2511.07991. arXiv:2511.07991.

[14] J. Plu, O. M. Escobar, E. Trouillez, A. Gapin, P. Lisena, T. Ehrhart, R. Troncy, Text2KGBench-LettrIA: A refined benchmark for Text2Graph systems, in: CEUR (Ed.), International Workshop on Knowledge Base Construction from Pre-trained Language Models (KBC-LM), Nara, Japan, 2025.