Automating AI Risk Profiling with LLM-Engineered OWL Axioms and Reasoning

Manas Goyal[1], Yogesh Kamath[1,*] and Swathi Shyam Sunder[1]

[1]Siemens Technology and Services Private Limited, Bengaluru, India

Abstract

AI governance frameworks increasingly require systematic risk assessment, yet industrial practice relies on manual evaluation of lengthy questionnaires by domain experts, a process that is slow, inconsistent, and unauditable. We present a neuro-symbolic approach that formalizes industrial AI risk profiling as a knowledge graph reasoning task. Starting from a 33-attribute risk questionnaire with 63 IF/AND/OR business rules, we use a Large Language Model (LLM) to co-engineer OWL 2 General Concept Inclusion (GCI) axioms that encode these rules over two reference ontologies: AIRO, a public foundational ontology for AI risk concepts, and GARM, a Siemens-internal extension providing a hazard taxonomy. A project's risk profile is captured as an ontology instance describing its AI models, output modalities, training techniques, data handling, and automation level; a Description Logic (DL) reasoner (HermiT) automatically infers which hazards from a catalog of 63 categories are relevant. We evaluate three axiom design patterns and document an error taxonomy of five LLM-generated defect types, four of which are invisible in the serialized syntax and detectable only by a formal reasoner. Validation on two project profiles yields full recall and zero false positives, matching expert assessment. The pipeline reduces manual cross-referencing of 33 attributes against 63 rules to sub-second automated inference. The results demonstrate that a generate-then-verify workflow with a reasoner-in-the-loop is both effective and essential for LLM-driven axiom authoring in safety-critical domains.

Keywords

LLM ontology engineering, OWL 2 reasoning, neuro-symbolic AI, knowledge graph, AI governance

1. Introduction

AI governance frameworks including the EU AI Act [1], the NIST AI Risk Management Framework, and ISO/IEC 42001 increasingly mandate systematic risk assessment for AI systems. In industrial practice at Siemens, this assessment is operationalized through the *AI Risk Management* (ARM) questionnaire: a structured instrument comprising 33 binary attributes (e.g., "Does the system use a pre-trained model?", "Does it process personal data?") whose answers trigger a subset of ∼63 hazard-flagging rules. Today, this evaluation is performed manually by domain experts, a process that is time-consuming, inconsistent across evaluators, and difficult to audit.

To illustrate the complexity, consider a hypothetical *HR Chatbot undergoing AI risk assessment by a human evaluator*, a fully autonomous system that fine-tunes a large language model on personal employee data and produces text output. An evaluator must trace through rules such as:

- IF has_AI_model AND (output_text OR … OR output_video) ⇒ flag *Hallucination Risk*
- IF processes_personal_data AND automation_level_L5 ⇒ flag *Restrictions on Automated Decision Making*

Even for four rules the cross-referencing is manageable; for the full set of 63 rules, some sharing attributes, some with nested disjunctions, manual evaluation becomes combinatorially tedious and difficult to audit. Moreover, the rule set is *not static*: new risk conditions require end-to-end re-validation.

LLMs4KGOE 2026: Workshop on LLM-driven Knowledge Graph and Ontology Engineering, co-located with ESWC 2026, May 10–14, 2026, Dubrovnik, Croatia

*Corresponding author.

manas.goyal.ext@siemens.com (M. Goyal); yogesh.kamath@siemens.com (Y. Kamath); swathi.sunder@siemens.com (S. S. Sunder)

0009-0007-5502-9838 (M. Goyal); 0009-0008-3482-9331 (Y. Kamath); 0000-0002-8375-5569 (S. S. Sunder)

Why ontologies, and why LLMs? A conventional rule engine (e.g., Drools[1] or OMG Decision Model and Notation[2]) could automate the IF/AND/OR evaluation. However, an ontological formalization offers three advantages that a flat rule engine lacks: (1) *class-hierarchy reasoning* inferred hazards are automatically grouped into risk categories via `rdfs:subClassOf` chains, enabling aggregated risk reporting without additional rules; (2) *open-world extensibility* new modalities, techniques, or hazard categories can be added to the TBox without rewriting existing rules; and (3) *interoperability* the OWL 2 knowledge graph can be queried via SPARQL, linked to external vocabularies (e.g., AIRO [2], itself aligned with EU AI Act and ISO 31000 risk concepts), and deployed to standards-compliant triple stores.

Yet authoring OWL 2 General Concept Inclusion (GCI) axioms manually is itself a bottleneck: it requires simultaneous expertise in the domain (risk management), in Description Logic theory (punning, subsumption, open-world semantics), and in serialization formats (Turtle, OWL/XML). LLMs bridge this gap: they can perform the syntactic translation from semi-structured business rules to axiom candidates at pace but, as we show, they introduce subtle errors, some causing inconsistency (Section 5, E1), others producing logically valid but semantically incorrect inferences detectable only via expert-designed test profiles (E3–E4).

We formalize the risk profiling process as a knowledge graph reasoning task: questionnaire answers become an OWL 2 ABox (instance data), hazard rules become GCI axioms (schema-level rules), and a DL reasoner infers which hazards apply. The central contribution is investigating whether an LLM can reliably author these GCI axioms and what (generate-then-verify) safeguards are needed.

Our contribution is threefold:

1. A **neuro-symbolic pipeline** where an LLM co-engineers GCI axioms from business rules, with HermiT as a formal verification backend.
2. A **systematic comparison** of three OWL axiom design patterns for encoding business rules, with documented failure modes per pattern.
3. An **error taxonomy** of LLM-generated ontology defects discovered through reasoner-in-the-loop validation, grounded in a real industrial AI risk assessment use case.

2. Related Work

LLMs for Ontology Engineering. Recent work on LLM-driven ontology construction consistently identifies axiom generation as the weakest link. Babaei Giglou et al. [3] find strong GPT-4 performance on lexical tasks but poor results for complex axioms, corroborated by Saeedizade and Blomqvist [4], who report logically fragile axioms from competency-question-driven generation, and Lippolis et al. [5], who document persistent axiom-level errors despite improved prompting strategies. Doumanas et al. [6] show ontology accuracy degrades as human involvement decreases, a trade-off we address through formal reasoner verification rather than increased human review. Our work differs in two respects: we target translation of industrial AI risk rules into OWL 2 GCI axioms, a task not addressed by the above work, and we contribute a formal error taxonomy of LLM failures discovered through reasoner-in-the-loop validation.

AI Risk Ontologies. Our pipeline builds on AIRO [2], a foundational ontology for AI risk concepts with semantic specifications for high-risk AI classification under the EU AI Act. Hernandez et al. [7] construct a KG mapping requirements across the EU AI Act, ISO/IEC 42001, and NIST AI RMF, cross-standard compliance reasoning that our GARM extension supports at the project level. The most closely related system is CO2 [8], which combines an LLM with a risk ontology and Neo4j for AI compliance checking; however, CO2 extracts compliance information from regulatory documents, whereas our pipeline translates structured business rules into formally verified OWL axioms.

[1] https://github.com/apache/incubator-kie-drools
[2] https://www.omg.org/spec/DMN

Neuro-Symbolic Approaches. Combining neural and symbolic components for knowledge engineering is well-established [9]. Our contribution is a concrete instance: the LLM generates candidate GCI axioms and a DL reasoner validates them, creating a feedback loop where each rejection drives the next iteration. Section 5 shows why this is essential: four of five error types are invisible in serialized syntax.

3. Pipeline and Architecture

The pipeline translates the problem framed in Section 1 into a concrete architecture. Its key design principle is *separation of concerns*: a stable domain ontology, a dynamic rule layer, and per-project instance data are maintained in distinct, independently versionable files. Figure 1 provides an overview.

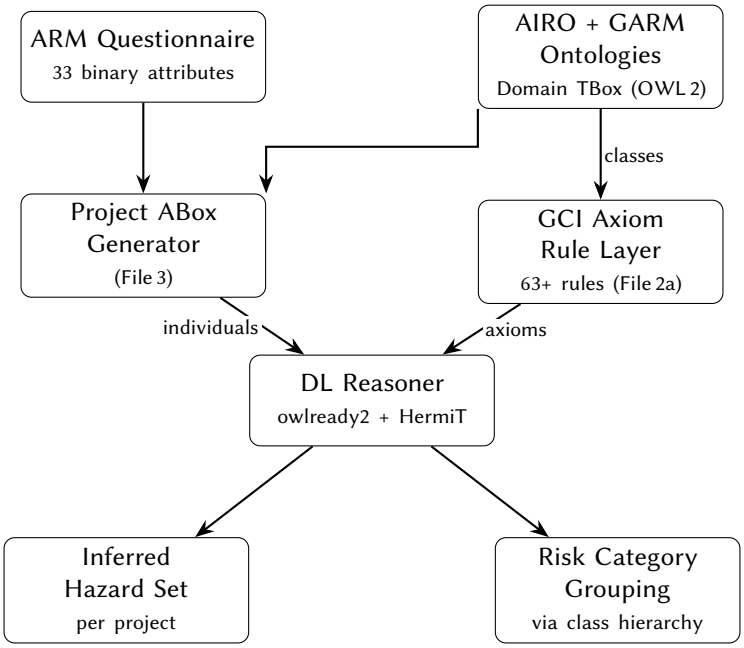

Figure 1: Pipeline overview. Questionnaire responses are instantiated as an OWL 2 ABox; GCI axioms encode hazard rules over the AIRO/GARM TBox; the HermiT reasoner infers project-specific hazards and groups them by risk category.

3.1. Input Artifacts

The pipeline ingests three input artifacts:

1. **ARM Questionnaire**: 33 binary attributes organized into categories (model properties, data characteristics, outputs, automation level, etc.), with 63 IF/AND/OR rules mapping attribute combinations to hazards.
2. **AIRO Ontology** [2]: Foundation TBox (schema) providing classes such as `airo:AISystem`, `airo:AIModel`, `airo:Output`, `airo:Hazard`, and properties such as `airo:hasModel`, `airo:producesOutput`.
3. **GARM Ontology**: A Siemens-internal extension of AIRO containing 130 classes organized in a two-level hazard taxonomy (L1: risk category, L2: specific hazard type), plus domain-specific classes for modalities (`garm:Text`, `garm:Audio`), techniques (`garm:FineTuning`), and automation levels (`garm:L5_Observer`).

3.2. Four-File Architecture

The knowledge graph is split into four files: (1) a stable GARM TBox, (2a) GCI axioms encoding rules, (2b) named hazard individuals, and (3) a per-project ABox. This separation enables independent versioning and correct deployment to triple stores.

3.3. LLM Co-Engineering Workflow

The LLM contributes to both axiom authoring and ABox generation, but the critical challenge lies in Steps 1–2 below: translating business rules into GCI axioms requires DL expertise that the LLM approximates but cannot guarantee. ABox generation from binary questionnaire responses is a straightforward mapping where the LLM assists but errors are immediately visible.

The axiom engineering process follows a *generate-then-verify* loop, using an LLM (Claude 3.5 Sonnet) as the co-engineering agent. Claude 3.5 Sonnet was selected for its strong structured-code generation on SWE-bench Verified [10], BigCodeBench [11], and EvalPlus [12]; multi-model comparison remains future work:

1. **Attribute Mapping**: The LLM reads each ARM attribute definition and proposes a mapping to AIRO/GARM class expressions. Example: ARM attribute #11 "has_AI_model" → `airo:hasModel some airo:AIModel`. The complete GARM TBox is provided as input context; undefined IRIs are caught at reasoner loading.
2. **Rule Translation**: For each ARM rule (e.g., `IF AND(has_AI_model, OR(Text, Audio))`), the LLM generates a candidate GCI axiom in Turtle syntax.
3. **Reasoner Verification**: The candidate axiom is loaded alongside the full ontology into owlready2 + HermiT. Verification checks: (a) ontology consistency, (b) expected inferences on a test ABox, (c) absence of subsumption pollution.
4. **Error Correction**: If verification fails, the error is fed back to the LLM with the reasoner output, and a corrected axiom is generated.

The human expert's role is to validate the attribute mappings (Step 1), design the test ABox (Step 3), and adjudicate when the LLM and reasoner disagree. The LLM contributes speed and syntactic fluency; the reasoner contributes formal correctness guarantees; the human contributes domain grounding and intent validation. A multi-agent architecture could improve modular reliability; our single-model design prioritizes pipeline simplicity and full interaction traceability.

Each prompt provided the full GARM TBox, the ARM rule in natural language, and one completed axiom as a few-shot exemplar.

3.4. Running Example: From Questionnaire to Hazard

We illustrate the end-to-end pipeline using the HR Chatbot introduced in Section 1. First, a project ABox is created from the questionnaire answers:

```
project:HRChatbot a airo:AISystem ;
  airo:hasModel project:LLM_FT ;
  airo:producesOutput project:TextOut ;
  garm:hasAutomationLevel garm:L5_Observer ;
  garm:processesPersonalData "true"^^xsd:boolean .

project:LLM_FT a airo:AIModel ;
  garm:usesTechnique garm:FineTuning .

project:TextOut a garm:Text .
```

Next, a GCI axiom encodes one ARM rule (H045 Hallucination) in Description Logic notation:

$$\texttt{AISystem} \sqcap \exists \texttt{hasModel}.\texttt{AIModel} \sqcap \exists \texttt{producesOutput}.(\texttt{Text} \sqcup \texttt{Audio} \sqcup \texttt{Image} \sqcup \texttt{Video}) \sqsubseteq \exists \texttt{hasRelevantHazard}.\{\texttt{Hallucination}\}$$

Because `HRChatbot` satisfies the left-hand side (it is an `AISystem` with at least one `AIModel` and a `Text` output), HermiT infers:

```
project:HRChatbot garm:hasRelevantHazard hazard:Hallucination .
```

The same mechanism applies to the remaining three rules; after reasoning, the HR Chatbot is linked to all four hazard individuals. Section 4 explains why this particular axiom shape Pattern C (`owl:hasValue` GCI) was selected over two alternatives.

4. Three Axiom Design Patterns

A critical design decision is how to encode the connection between an AI system's risk profile and its applicable hazards. Each pattern must express the same intent *if a project matches certain conditions, infer a link to a hazard individual* but the OWL 2 semantics differ substantially. We evaluated three patterns, illustrating each with the Hallucination rule from the HR Chatbot example. These patterns emerged iteratively: the LLM proposed Pattern A, HermiT rejected it (inconsistency); the LLM then proposed Pattern B, which caused subsumption pollution; Pattern C passed all checks. We applied the same selection process across all four structural rule types (conjunction-only, nested disjunction, boolean guard, and multi-attribute); in every case Pattern C produced correct inferences while A and B exhibited the same failure modes.

4.1. Pattern A: `owl:equivalentClass`

Idea: define a named class for each risk profile; systems matching the conditions become members of that class.

The LLM's first proposal used named classes with equivalence axioms:

```
garm:HallucinationRiskProfile
   owl:equivalentClass [
    owl:intersectionOf (
     airo:AISystem
     [owl:onProperty airo:hasModel ;
      owl:someValuesFrom airo:AIModel] ...
    )] .
```

Failure: The equivalence axiom itself is sound; `equivalentClass` works well for classifying instances into categories (our TBox uses it for broad risk-category gates). The problem arises because the pipeline also needs to assign a specific hazard *individual* via `owl:hasValue`. Reusing the same IRI (`garm:HallucinationRiskProfile`) as both a class (in the equivalence) and an individual (in `hasValue`) constitutes OWL 2 DL *punning*; HermiT rejected this as inconsistent.

4.2. Pattern B: Anonymous `rdfs:subClassOf` GCI

Idea: avoid the named class by using an anonymous GCI, but the right-hand side accidentally makes the system *a kind of hazard* rather than *linked to a hazard.*

The corrected version used anonymous class expressions with GCI:

```
[owl:intersectionOf (airo:AISystem ...)]
   rdfs:subClassOf garm:HallucinationHazard .
```

Failure: This caused *subsumption pollution* HRChatbot, which matched the LHS conditions, was reclassified as a member of `garm:HallucinationHazard`, conflating AI systems with hazard classes.

4.3. Pattern C: `owl:hasValue` GCI (Selected)

Idea: instead of classifying the system *into* a hazard class, assign it a *property value* pointing to a hazard individual.

The final pattern uses `owl:hasValue` with hazard *individuals*:

```
[owl:intersectionOf (airo:AISystem ...)]
   rdfs:subClassOf
      [owl:onProperty garm:hasRelevantHazard ;
       owl:hasValue hazard:Hallucination] .
```

Result: Clean inference the system gains a *property value* pointing to a hazard individual, without being reclassified. No punning, no pollution. This pattern is validated by precedent in the Financial Industry Business Ontology (FIBO) [13] and the W3C Semantic Web Best Practices note on specified values [14].

Table 1
Comparison of three axiom design patterns. ✓ = criterion met, × = criterion not met.

Criterion	A: equivalentClass	B: subClassOf	C: hasValue
OWL 2 DL compliant	×	✓	✓
Punning-free	×	✓	✓
Pollution-free	N/A	×	✓
Correct inference	×	×	✓

5. LLM Error Taxonomy

During the co-engineering process, we cataloged errors introduced by the LLM that were caught by the reasoner or by manual inspection. We organize these into five categories:

E1: Punning Violations. The LLM used the same IRI as both a class and an individual (Pattern A), violating OWL 2 DL's species separation. This error is invisible in Turtle syntax and can only be detected by a DL reasoner.

E2: Ontology IRI Errors. The LLM generated import statements with incorrect namespace suffixes (`garm#` vs. `garm`), causing silent import failures in Protégé. The root cause was an inconsistency in the source ontology between `rdf:about` and `xml:base` attributes, which the LLM propagated without detecting.

E3: Subsumption Pollution. Pattern B caused the project AI system to be reclassified as a hazard class member. The LLM did not anticipate this side effect because subsumption is an emergent property of the class expression interaction, not locally visible in the axiom.

E4: Type Granularity Mismatch. When generating a project ABox from documentation, the LLM typed modality instances as `airo:Modality` (the generic superclass) instead of `garm:Text` or `garm:Audio` (the specific subclasses referenced in the GCI axiom conditions). This caused rules to silently fail the reasoner correctly reported no inference, but for the wrong reason.

E5: Graph Deployment Errors. The LLM generated ontology classes, GCI axioms, and project instance data in a single monolithic file. While semantically correct for a local reasoner (which merges everything), keeping reusable schema and axioms separate from per-project instance data is essential

Table 2
LLM error taxonomy with detection mechanism.

Error	Type	Detected By	Visible in Syntax?
E1 Punning	Logical	HermiT	No
E2 IRI mismatch	Syntactic	Protégé import	Partially
E3 Pollution	Logical	HermiT + test ABox	No
E4 Type granularity	Semantic	Rule non-firing	No
E5 Graph separation	Deployment	Triple store reasoning	No

for deployment: triple stores restrict OWL inference to a designated schema graph, and new project ABoxes must be generated independently as they enter the pipeline.

A striking observation is that 4 of 5 error types are *not visible in the Turtle syntax*. An LLM reviewing its own output textually would not catch them. This underscores the necessity of a formal reasoner in the loop.

6. Evaluation

6.1. Test Setup

We validated the pipeline using owlready2 (v0.50) with HermiT as the DL reasoner. The ontology comprises AIRO (558 triples), GARM (498 triples), 50 Pattern C GCI axioms spanning all four GARM risk categories, corresponding hazard individuals, and per-project ABoxes. We evaluate correctness, completeness (zero false negatives), and precision (zero false positives) on two profiles: (1) an **HR Chatbot** (synthetic), a fully autonomous chatbot (L5) with fine-tuned GPAI model, personal data, external source, public UI, designed to trigger the broadest rule set; and (2) a **Voice Assistant** (real), three task-specific models (LLM, STT, TTS) with audio/text output, prompt engineering, L2 automation, no personal data.

6.2. Results

All 15 representative axioms match expert assessment with zero false positives and zero false negatives. Over all 50 axioms the HR Chatbot triggers 42 hazards and the Voice Assistant 11, consistent with differing risk exposure. Eight axioms requiring absent conditions (e.g., emotion/biometric output, web scraping) correctly remain silent.

HermiT reasoning completes in under one second per profile. Inferred hazards are *automatically grouped* into GARM risk categories via the class hierarchy (e.g., `hazard:Hallucination` $\in$ `garm:ReliabilityAndRobustness`), serving as both classification schema and aggregation mechanism.

Data Availability. The AIRO ontology is available at https://w3id.org/airo. Pipeline code, prompt templates, and validated axioms will be released upon publication. The full GARM ontology is Siemens-proprietary; the published subset suffices to reproduce the demonstrated patterns.

7. Discussion and Conclusion

The Dual Role of the LLM. The LLM generated correct GCI axiom *structure* remarkably quickly; the initial prototype was constructed in hours rather than weeks, yet every axiom required at least one correction after reasoner verification. It excels at *syntactic* pattern generation but lacks *semantic* grounding in DL theory. Four of five error categories (Section 5) are invisible in serialized syntax; our generate-then-verify loop operationalizes neuro-symbolic control [9], ensuring every deployed axiom carries a formal proof of consistency.

Table 3
Hazard inference on 15 representative axioms (of 50 validated). Categories: RR=Reliability & Robustness, LC=Legal & Compliance, TR=Transparency, HO=Human Oversight.

Rule (LHS pattern)	HR	VA
RR-01 Hallucination (model+output)	✓	✓
RR-06 Wrong Data (learn | RAG)	✓	×
RR-09 Misuse (UI+user)	✓	✓
RR-11 Data Splitting (learning)	✓	×
RR-13 ODD Spec (safety/impact)	✓	×
RR-18 Distrib Shift (unconditional)	✓	✓
LC-02 Synthetic Transparency	✓	✓
LC-04 Input IP (ext source)	✓	×
LC-09 3rd Party Provider	✓	×
LC-10 Personal Data Use	✓	×
LC-13 Auto Decisions (PD+L5)	✓	×
TR-01 UI Transparency	✓	✓
TR-04 GPAI Provider Info	✓	×
HO-01 Org Measures (user type)	✓	✓
HO-02 Tech Measures (UI)	✓	✓
Repr. (of 15)	**15**	**7**
All 50 axioms	**42**	**11**
False positives	0	0

Why Not SHACL? SHACL 1.2 Rules [15] could in principle encode equivalent conditions. We followed the traditional OWL 2 DL path because the existing AIRO/GARM class hierarchy already groups inferred hazards into risk categories via `rdfs:subClassOf` chains, and GCI axioms integrated naturally with this structure. Whether SHACL 1.2 rules would yield a simpler or more maintainable encoding is an open question that we leave for future work.

Provenance and Governance. Every inferred hazard is traceable to a specific GCI axiom and ABox assertion, producing audit trails by construction. The four-file architecture lets GCI axioms evolve independently of the stable TBox, enabling version-controlled updates without schema-wide regressions.

Limitations and Future Work. Scaling to the full ARM rule set may reveal additional error patterns. We plan to link inferred hazards to mitigation methods, KPIs, and regulatory requirements. We do not claim OWL reasoning is computationally superior to production rule engines; rather, class-hierarchy grouping, open-world extensibility, and SPARQL interoperability justify the DL complexity for this use case.

Conclusion. We presented a neuro-symbolic pipeline that combines LLM-driven axiom engineering with DL reasoner verification for AI risk assessment. The error taxonomy provides concrete evidence that LLM-generated ontologies require formal verification, and the pipeline demonstrates that neural generation, symbolic reasoning, and human oversight can produce auditable, formally grounded knowledge graphs for safety-critical domains.

Acknowledgments

This work was conducted within the expert reference group for Trustworthy AI initiative at Siemens. We thank the ARM questionnaire authors for providing the rule definitions and the Siemens AI governance team for domain validation.

Declaration on Generative AI

During the preparation of this manuscript, the authors used an LLM to assist with drafting and structuring the paper text. All content was reviewed and edited by the authors, who take full responsibility for the publication. The use of an LLM as a co-engineering tool for ontology axiom generation is the research object of this paper and is described in detail in Section 3.

References

[1] European Parliament and Council, Regulation (EU) 2024/1689 laying down harmonised rules on artificial intelligence (AI Act), Official Journal of the European Union, L series, 2024.

[2] D. Golpayegani, H. J. Pandit, D. Lewis, To be high-risk, or not to be—semantic specifications and implications of the AI Act's high-risk AI applications and harmonised standards, in: Proc. ACM FAccT, 2023, pp. 905–915.

[3] H. Babaei Giglou, J. D'Souza, S. Auer, LLMs4OL: Large language models for ontology learning, 2023.

[4] M. J. Saeedizade, E. Blomqvist, Navigating ontology development with large language models, in: Proc. ESWC, 2024, pp. 143–161. doi:10.1007/978-3-031-60626-7_8.

[5] 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: Proc. ESWC, 2025, pp. 321–341. doi:10.1007/978-3-031-94575-5_18.

[6] D. Doumanas, G. Bouchouras, A. Soularidis, K. Kotis, G. A. Vouros, From human- to LLM-centered collaborative ontology engineering, Applied Ontology 19 (2024) 334–367. doi:10.1177/15705838241305067.

[7] J. Hernandez, D. Golpayegani, D. Lewis, An open knowledge graph-based approach for mapping concepts and requirements between the EU AI Act and international standards, AI and Ethics 5 (2025) 4463–4474. doi:10.1007/s43681-025-00708-6.

[8] V. S. P. Turaga, T. Pahi, S. Tjoa, S. Siami-Namini, A. Siami Namin, CO2 (co-compliance officer): An LLM-based ontology-driven methodology for generating knowledge graphs and AI compliance checking, IEEE Access 13 (2025) 210576–210606. doi:10.1109/ACCESS.2025.3639228.

[9] P. Hitzler, M. K. Sarker (Eds.), Neuro-Symbolic Artificial Intelligence: The State of the Art, volume 342 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2022.

[10] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, K. Narasimhan, SWE-bench: Can language models resolve real-world GitHub issues?, in: Proc. ICLR, 2024. ArXiv:2310.06770.

[11] T. Y. Zhuo, M. C. Vu, J. Chim, H. Hu, W. Yu, R. Ratber, J. Zhu, J. Shi, F. K. Wang, F. Ziber, Others, BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions, in: Proc. ICLR (Oral), 2025. ArXiv:2406.15877.

[12] J. Liu, C. S. Xia, Y. Wang, L. Zhang, Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation, 2023. URL: https://arxiv.org/abs/2305.01210. arXiv:2305.01210.

[13] M. Bennett, The financial industry business ontology: Best practice for big data, Journal of Banking Regulation 14 (2013) 255–268.

[14] A. Rector, Representing specified values in OWL, W3C Semantic Web Best Practices Working Group Note, 2005. URL: https://www.w3.org/TR/swbp-specified-values/.

[15] R. David, D. Habgood, A. Seaborne, S. Steyskal, SHACL 1.2 rules, W3C Editor's Draft, 2026. URL: https://w3c.github.io/data-shapes/shacl12-rules/, published 20 April 2026.