Using LLMs to Develop Legal Rule Ontologies for Compliance Checking

Galileo Sartor[1,*], Enrico Francesconi[2] and Adam Wyner[1]

[1]Swansea University, Department of Computer Science, Swansea, UK
[2]IGSG-CNR, Institute of Legal Informatics and Judicial Systems, National Research Council, Italy

Abstract

This paper presents a new approach to text-to-knowledge graph conversion using large language models. Extracting, updating, and maintaining structured rulebases and ontologies from natural language text has been a labor-intensive process. While Large Language Models (LLMs) have increased the accessibility of automated extraction, they introduce significant risks regarding accuracy and logical consistency. This paper proposes a pipeline that improves accuracy and logical consistency in the automated extraction and validation of rule ontologies for legal reasoning, a domain where quality is paramount. To improve the correctness of the generated ontology, it is continuously evaluated using methods such as language syntax checkers and competency questions, so to ensure the integrity of the resulting knowledge graph. Preliminary results discuss the efficacy of these methods and their broader implications for automating legal rule modelling in highly regulated environments. The worked example applies to traffic regulations for Autonomous Vehicles (AVs). Our approach leverages the natural language understanding of LLMs to formalise legal norms as an OWL ontology, utilizing class restrictions to define Compliant and Non-Compliant behaviour. The paper presents the different methods used in the pipeline as well as the translation and validation phases. We discuss how the ontology could be used in the legal domain.

Keywords

Legal Reasoning, Semantic Web, OWL, Norm Compliance, Large Language Models, Autonomous Systems

1. Introduction

Ontologies[1] have become a foundational tool in the legal domain, enabling structured representation, management, and retrieval of complex legal knowledge. Early work emphasised the role of explicit domain conceptualisations to support legal information systems, where ontologies can clarify commonalities across legal approaches and help to create reusable legal knowledge libraries [1]. However, the development and maintenance of legal ontologies remains a complex, resource-intensive task requiring significant expertise in both law and ontology engineering. This has given rise to a *knowledge acquisition bottleneck* [2], limiting the scalability and adaptability of legal ontologies in practice.

Encoding traffic rules for Autonomous Vehicles (AVs) is an interesting use of legal rule ontologies, as proposed AV platforms tend to rely on black box systems or ad hoc structures to govern vehicle behaviour and ensure legal compliance. However, these systems lack an explicit, shared, and structured representation of norms.

To address both the knowledge acquisition bottleneck and the need for explicit legal knowledge in AV systems, we propose leveraging Large Language Models (LLMs), broadly to understand their use and limits for legal reasoning[3]. Recent advances in LLMs have dramatically improved their ability to parse, interpret, and restructure natural language text, making it feasible to partially automate the extraction of structured legal knowledge from natural language sources. Critically, however, LLM outputs must be systematically verified as correct or consistent with the intended interpretation before being incorporated into a knowledge graph or ontology, as generated content can appear plausible while being semantically incorrect.

ESWC'26: 5th Workshop on LLM-Integrated Knowledge Graph Generation from Text (TEXT2KG), May 10-14, 2026 | Dubrovnik, Croatia

*Corresponding author.
galileo.sartor@swansea.ac.uk (G. Sartor); enrico.francesconi@cnr.it (E. Francesconi); wyner@swansea.ac.uk (A. Wyner)
0000-0001-6355-851X (G. Sartor); 0000-0001-8397-5820 (E. Francesconi); 0000-0002-9421-8566 (A. Wyner)

[1]For our purposes, *ontology* and *knowledge graph* can be used interchangeably.

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-08-01

This paper presents a pipeline that takes natural language legal norms as input and produces OWL (the W3C Web Ontology Language) ontologies in which compliant and non-compliant agent behaviours are encoded as class restrictions. The pipeline couples a structured prompting strategy with a *synchronous* validation phase that checks generated ontologies for syntactic correctness and structural consistency throughout the generation process.

While the prompting strategy is an important component, the central focus is the validation phase and its effect on the quality and consistency of the generated ontologies. Running validation continuously during generation allows errors to be identified and corrected iteratively; this is especially important when building up a set of rules incrementally, since errors in one rule can propagate to subsequent rules that share classes or properties.

The output of LLMs is not deterministic and can vary across queries within an LLM or across LLMs - different LLMs can introduce distinctions based on different training data, fine-tuning, and randomness. While these differences may be minor style or syntax changes, they might also alter the logical structure of a norm. In a legal context, such variation could lead to substantive legal issues. We argue that AVs should abide by traffic rules in a consistent, uniform, and convergent manner. Thus an explicit, formal, and validated representation is even more important.

This paper focuses on syntactic and structural validity of the KG rather than drawing inferences. With the structure of the pipeline, additional tests can be added to evaluate the logical consistency and correctness of inferences, though one method currently in evaluation is the instantiation of individuals that correspond to known scenarios.

While the pipeline is tailored to traffic rules for AVs, the overall approach is general and could, in principle, be adapted to other legal domains. The modular design allows LLMs, prompting strategies, and validation techniques to be substituted as needed. By demonstrating the feasibility of LLM-guided ontology extraction in a complex legal domain, we contribute to the broader effort to leverage AI for scalable, transparent legal knowledge representation.

The main novel contributions of this paper are: (i) a modular pipeline for LLM-guided extraction and validation of legal ontologies from natural language text; (ii) an ontology model that encodes deontic norms as property restrictions, with disjoint *Compliant* and *Violating* subclasses; (iii) a comparative evaluation of multiple LLMs on a set of traffic rules; and (iv) a discussion of how the pipeline can be generalised to other legal domains.

The paper is structured as follows. Section 2 discusses the context, challenges, and requirements of autonomous driving. Section 3 reviews related work on legal ontologies and the use of LLMs in ontology engineering. Section 4 describes the ontology model and norm representation. Section 5 presents the proposed pipeline and methodology. Section 6 provides a running example from the UK Highway Code [2], highlighting the prompting strategy and LLM interaction. Section 7 discusses the *synchronous* validation method. Sections 8, 9, 10, respectively present an example, comparison amongst LLMs, and limitations of the study. Section 11 concludes with remarks and directions for future work.

2. Autonomous Driving

As mentioned, most AV implementations rely on machine learning (ML) models to govern vehicle behaviour and ensure legal compliance. These systems lack an explicit knowledge representation of legal rules, meaning they do not possess a structured understanding of traffic laws, nor a way to evaluate their compliance in accordance to the existing traffic norms. This lack of explicit specification is a central challenge, as decision-making algorithms must also provide legally defensible explanations for their actions. Instead, compliance is mostly derived through data-driven approaches, where the predicted behaviour is evaluated in a testing phase. Lawful behaviour can be induced by tweaking the loss function, but due to the *black-box nature of neural networks*, there is no guarantee that the actual behaviour is compliant with the relevant legal norms [4, 5].

[2]Available at https://www.gov.uk/guidance/the-highway-code/using-the-road-159-to-203

Conversely, most research in decision-making for AVs considers only collision free driving to be compliant, removing the complexity of the environment and constructing controlled models that consider traffic laws [6]. While the use of ML models is effective and an important piece of the driving puzzle, an approach solely based on inferred behaviour lacks explainability and robustness in complex factual, legal, and ethical scenarios. This is potentially even more relevant if we consider highly automated vehicles (SAE Levels 4 and 5, with minimal to no human interaction required [7]), which operate in environments where the available data for training is not only currently lacking, but likely to be too *thin* to be adequately captured and represented in ML models, yet highly relevant legally.

Another consideration is the range of legal jurisdictions which may be parallel (e.g., crossing national boundaries) or hierarchical (e.g., the Vienna Convention on Road Traffic in conjunction with national and local regulations). Here too, ontologies could help in developing a tiered knowledge base, where the different legal (and societal) norms can be merged together.

3. Related Work

Representing and reasoning with legal norms is a long-standing topic in the field of legal informatics, which has explored various approaches to formalising and automating legal knowledge. The complexity of legal language and the need for precise interpretation have led to the development of different methodologies for representing legal norms, including rule-based systems, logic programming, and ontological modelling. From the early work on representing legal knowledge using rule-based systems and logic programming [8], to more recent approaches leveraging Large Language Models [9], the field has evolved to address the challenges of legal reasoning and compliance checking in various domains. In recent years there has been a growing interest in the *Rules as Code* movement, which advocates for the formalisation of legal rules in a machine-readable format to facilitate their automated application [10].

To deal with legal compliance, a number of different approaches are being evaluated, broadly grouped in two categories: scenario testing, which deals with specific complex situations; and more general rule representation, mainly using logic formalisms such as Prolog [11], Linear Temporal Logic [12], or Defeasible Deontic Logic [13]. This paper addresses the second category, adopting an ontology-based approach using OWL to develop a comprehensive, machine-readable rulebase.

The related work can be divided into two main areas: a) the use of ontologies in the considered legal domains, and b) the use of LLMs to create and update ontologies. This work will not go into the use of LLMs directly for legal reasoning, as it is out of scope for this paper.

Since [1], research introduced core legal ontologies such as FOLaw and LRI-Core, which model dependencies in legal reasoning and capture abstract, commonsense concepts underlying legal knowledge[14]. These ontologies have been applied in various European ICT projects, demonstrating practical value in legal information processing and reasoning tasks [15]. The evolution of legal ontologies aligns with the aims of the Semantic Web towards scalable, reusable, and interoperable models. Modern legal ontologies are often built using W3C standards (e.g., OWL), and their development is guided by principles of knowledge reuse and modularity [16]. Comparative analyses have catalogued a wide range of legal ontologies, focusing on their features, implementation details, and suitability for reuse across different legal frameworks. Ontologies have also been leveraged to improve legal information retrieval, addressing challenges such as synonymy and ambiguity in legal texts. Ontology-based search frameworks and knowledge graphs have demonstrated superior performance over traditional keyword-based methods, enabling more accurate and semantically rich retrieval of legal documents [17]. Additionally, ontologies support advanced applications such as legal argumentation, statutory reasoning, and the semantic annotation of legal texts, further enhancing the capabilities of legal information systems. Ontologies have modelled deontic norms as classes and restrictions [18], an approach that can handle defeasible norms and leverage the OWL reasoners to make decisions and perform legal compliance checking. Recent work has explored the integration of foundational ontologies (e.g., UFO) to foster interoperability and conceptual rigour as well as the use of ontology design patterns to streamline the modelling of recurring legal knowledge [19]. The field continues to address challenges related to the dynamic and

heterogeneous nature of legal concepts, the need for domain expert involvement, and the adaptation of ontologies to evolving legal systems and technologies [20].

In the AV domain, ontologies have been employed to model traffic regulations, road environments, and vehicle behaviours, as well as to define scenarios and situations. These ontologies facilitate the interpretation and application of traffic rules by AV systems, supporting decision-making processes in complex driving scenarios [21, 22, 23]. For instance, ontologies have been used to represent traffic signs, signals, and road layouts, enabling AVs to understand and comply with legal requirements while navigating diverse road conditions [24, 25].

An example of an ontology in the AV domain is the ASAM OpenXOntology, which provides a comprehensive framework for modelling various aspects of the driving environment, including traffic rules, road infrastructure, and vehicle dynamics [26]. This ontology supports the development of AV simulations, focusing on the ASAM standard for scenario description (OpenSCENARIO) and the ASAM standard for road network description (OpenDRIVE). These are very useful for testing and simulation, but they do not provide a structured representation of legal norms that can be used for compliance checking.

With the increasing ability of LLMs to manipulate text and to assist with coding tasks, it becomes possible to use these tools to generate a structured version of a source in natural language. Recent work has explored the use of LLMs to assist in ontology engineering tasks, such as ontology learning[27], population, and alignment. For example, LLMs have been employed to extract concepts and relationships from unstructured text, facilitating the semi-automated construction of ontologies [28]. Additionally, LLMs have been used to generate ontology axioms and definitions, leveraging their language understanding capabilities to produce coherent and contextually relevant representations.

While the focus has often been on ontology population on the basis of an existing schema, there is also work on the use of LLMs to create an ontology from scratch or to extend an existing ontology with new concepts and relationships [29]. This involves prompting the LLM with relevant domain knowledge and examples, and iteratively refine the ontology through validation and correction.

4. Ontology Model and Norm Representation

In this section we describe the ontology model used to represent rules from the UK Highway Code, the information about traffic rules and signs that must be obeyed by drivers in the UK. In particular we focus on the *Using the road (159 to 203)* section that deals with road junctions.

THe highway code states in its introduction that

> Many of the rules in the Code are legal requirements, and if you disobey these rules you are committing a criminal offence. You may be fined, given penalty points on your licence or be disqualified from driving. [...] Such rules are identified by the use of the words 'MUST/MUST NOT'. [...]
>
> Although failure to comply with the other rules of the Code will not, in itself, cause a person to be prosecuted, The Highway Code may be used in evidence in any court proceedings under the Traffic Acts (see The road user and the law) to establish liability. This includes rules which use advisory wording such as 'should/should not' or 'do/do not'.

In the proposed ontology these legal norms are represented as class restrictions, encoding compliant and violating behaviour.

The ontology model used in this work consists of a rule-based representation of legal norms, where the norms are represented as class restrictions on agent behaviour, capturing the essential components of legal rules such as subjects, actions, conditions, and modalities such as in [30]. Legal norms are represented as a set of conditions that define compliant and violating behaviours for a given situation.

Norms, expressing obligations, permissions, and prohibitions, are represented as restrictions on the properties of legal agents (e.g., vehicles, drivers) within specific contexts (e.g., road environments). Each agent is modelled as a class in the ontology, with properties corresponding to actions they can perform

and conditions under which these actions are regulated. Other entities in the environment (e.g., traffic signs, road types) are also modelled as classes with relevant properties. Determining the compliance status then consists of evaluating these properties against the defined norms.

In our ontology, we define the basic structure of legal norms as follows. We have a top-level class `Agent` representing any entity that can perform actions (e.g., the Vehicle). We then define subclasses of `Agent` to represent situations in which the agent finds itself. For example, in the AV domain we might have a class `AtRedLight` representing the situation where the Vehicle is at a red traffic light. We then define two disjoint subclasses of `AtRedLight`: `Compliant` and `Violating`. The former represents the compliant behaviour (e.g., stopping at the red light), while the latter represents the violating behaviour (e.g., running the red light).

This general structure is fed to the LLM during the generation phase in abstract terms, where the LLM is tasked with filling in the specific class restrictions that define compliant and violating behaviour for a given legal norm. We can then query the ontology to determine if an individual is compliant in a given situation, leveraging the available OWL reasoners. Compliance checking amounts to checking which *Vehicle* individuals belong to either *Compliant* or *Violating* class.

5. Methodology

The proposed methodology consists of a multi-step pipeline for extracting and validating legal ontologies from natural language text using LLMs, as shown in Figure 1. To achieve this in a controlled and systematic way, we define atomic steps for the translation and validation phases, making use of different strategies and tools at each stage. The translation is not left solely to the LLM, rather it is guided by a more structured *feedback loop* approach, where the LLM is prompted to generate an initial output, which is then validated with autonomous and semi-autonomous techniques (e.g., syntactic checks, test scenario evaluation) before being accepted or sent back for revision. As the main goal is to generate a correct and consistent ontology from scratch, or to extend an existing one, the validation phase is crucial to ensure that the output of the LLM is correct and consistent with the intended interpretation. The validation phase is designed to be *synchronous*, meaning that it runs continuously during the generation process, allowing for iterative refinement and correction of the generated ontology. This is particularly important in a legal context, where errors or inconsistencies in the representation of norms can have significant consequences. If the ontology is generated from scratch, the LLM can be provided with a general structure of the ontology (e.g., the one described in Section 4) and a *one-shot* example of how we want the ontology to be structured. This provides a clear template for the LLM to follow, while still allowing for flexibility in how the specific norms are represented. If the ontology is being extended, the LLM can be provided with the existing ontology and prompted to add new classes and properties as needed, while ensuring that they fit within the existing structure and adhere to the defined validation requirements. While LLMs are getting better at following instructions and generating structured output, they can still produce errors or inconsistencies, especially when dealing with complex structures (code, legal norms, etc.). The validation phase is designed to catch these issues early in the generation process, allowing for corrections to be made before the ontology is finalised. This iterative approach helps to ensure that the resulting ontology is both accurate and consistent with the intended interpretation of the legal norms.

The pipeline is designed to be modular, allowing for substitution of different LLMs, prompting strategies, and validation techniques as needed. In fact, this allows for use in different domains, replacing the example and the structure of the ontology in the initial prompt as needed. Other LLMs can be used, as well as altering the validation techniques, all without changing the overall structure of the pipeline. The reasoners used for validation can also be changed, depending on the specific requirements of the domain and the tooling offered by the reasoner.

The pipeline is also developed in a way that is accessible to experts to validate at runtime, with logs and references to every change made by the LLM. This approach could be further extended with an interactive view that the experts could use to alter the output, with the LLM pausing for confirmation

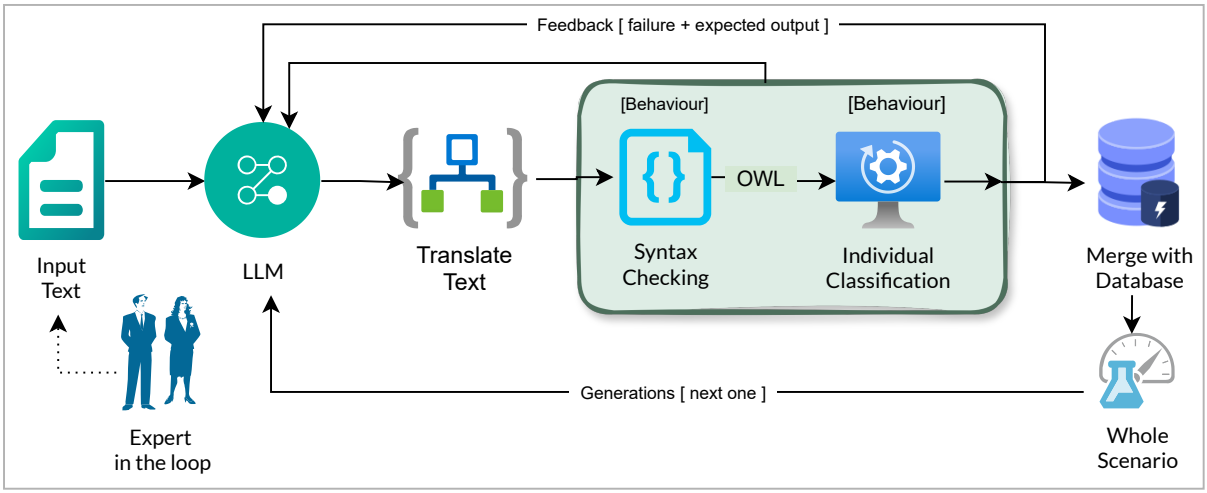

Figure 1: High level view of the proposed pipeline for LLM-guided rule extraction and validation.

of the generated structure.

We theorise that this structured approach will lead to more accurate and reliable ontology generation, while also providing a clear framework for evaluating the performance of different LLMs and techniques in the context of legal knowledge extraction.

Let us briefly outline the main steps of the pipeline, before going into more detail in the next sections. In the first step the LLM is prompted to generate an initial translation of the natural language norm in the Turtle syntax of OWL, following a structured prompt that includes the abstract ontology model and a one-shot example. The LLM is guided to produce an output that adheres to the defined structure of compliant and violating subclasses. The guidance provided to the LLM in this case is crucial to ensure that the generation follows a predefined structure. This structure is defined in the prompt, so it can be easily modified and adapted to different domains or requirements. The one-shot example serves as a concrete illustration of how the LLM should structure its output, providing a clear template for the generation process.

6. Prompt

The initial prompt is designed to guide the LLM in translating a natural language legal norm into a structured OWL ontology. The prompt is structured to provide clear instructions and examples to the LLM, ensuring that the generated output adheres to the desired format and structure.[3].

To guide the LLM, we use a combination of 1-shot and Chain-of-Thought (CoT) prompting. In the context of LLMs, CoT guides model reasoning via intermediate steps, where we break down the task into more manageable steps, with specific instructions and examples providing explanations of each stage. 1-shot prompting provides a single example to illustrate the desired output format and structure. This approach helps to guide the LLM and reduce ambiguities or inconsistencies in the model's output, while also providing a human understandable guide.

The translation phase is itself structured, as can be seen in the System prompt in Listing 1. The same prompt can be used for different LLMs with minor changes. Depending on the LLM used, the rule to convert is added in the "User" Prompt or appended to the "System" prompt.

The prompt starts by extracting the input text (e.g., a norm from a legal code) and breaks it into segments, e.g., sentences or clauses. Each segment is then processed individually, with the LLM being prompted to identify key concepts, such as subjects, actions, conditions, and deontic modalities (e.g., must, may, should). The extracted concepts are then mapped, when possible, to existing ontology

[3]The code for the prompt and examples is available at https://github.com/LyzardKing/llm_owl

```
# Prompt: Transform Legal Text into OWL

## Objective
Convert the provided legal text into OWL [...]

## Step by Step
1. Split the paragraph into sentences.
2. For each sentence identify: Subject, Conclusion, Deontic Modality, and Conditions.
   [...]
3. Present the JSON with the extracted elements
4. For each JSON object [...] encode the OWL representation:
   - **Conclusion**: Predicate name is the deontic modality [...]. Arguments are the subject, action
    ↪ and specifications.
   - **Conditions**: Use the verb as the predicate name. The arguments are the subject and
    ↪ specifications.
5. Build the rules [...] using the identified structure.

## Validation Requirements
- **Hierarchy Completeness:** Verify every class ending in Default/Mitigated is a subclass of its
 ↪ Compliant counterpart [...] and mapped as subclasses of their base Root class.
- **Violation Disjointness:** Confirm Compliant and Violating siblings are explicitly defined as
 ↪ owl:disjointWith.
- **Standard Suffix Enforcement:** Every class under a Root must end in one of four suffixes:
 ↪ Compliant, Violating, Default, or Mitigated.
- **Rule ID Consistency:** Check that any outcome class containing a Rule ID [...] starts with the
 ↪ exact name of its parent Root.

## Logical Handling
- If a rule has multiple condition sets, they should be represented as unions in OWL.
- If conditions are connected by "or", create separate rules for each alternative condition set.
- Ensure new properties or classes do not already exist in the ontology.
```

Listing 1: Structure of the initial prompt used to guide the LLM in translating natural language norms into OWL ontologies.

templates (i.e., the example in the prompt). Finally, the structured components are recombined using logical operators (e.g., AND, OR) to reconstruct the legal rule in a formalised, machine-readable format.

This analysis is mostly left to the LLM, but the structure of the analysed norms (e.g., from the UK Highway Code) and the examples in the prompt help to guide the LLM in performing the task correctly. More complex domains might require a more structured extraction process.

To address issues of consistency in the output of an LLM, we include a single example from the domain in question (e.g., the UK Highway Code) to illustrate the expected output format and structure. This example serves as a reference for the model, helping to align its output with the desired ontology representation. Specifically, it provides the model with a concrete reference it can replicate when generating the new output, reducing unwanted linguistic variations and ensuring greater consistency in the logical transformation process. The inclusion of additional information is thus used to explicitly limit the "autonomy" of the model in both phases (data extraction and generation) in order to minimise ambiguity and hallucination while enforcing strict adherence to the desired structure.

The more structured information is given to the LLM, the better the results. This structured information includes contextual information and examples (in 1\multi shot prompting). The LLM tends to fill gaps with information in its training dataset, so it is important to guide or limit this. As an example, with a shorter prompt the tested LLMs would include information about green traffic lights (not explicitly mentioned in the given rules), as well as other road traffic rules, not necessarily from the UK. With the current prompt, we give the necessary contextual information, provide details about the rule extraction process, and include one example of a modelled rule. It might be useful to include additional information, especially for more complex rules, or additional examples, although this is out of the scope for this work.

7. Validation Process

7.1. Syntax Validation

The first step in the validation process is to check the syntactic correctness of the generated OWL ontology. This can be done using existing tools and libraries that parse OWL ontologies and check for syntax errors. If the generated ontology fails this check, it is sent back to the LLM for revision, with feedback on the specific syntax issues that need to be addressed.

For the syntax validation we developed a small tool that gives the LLM access to a language service server (via the Language Server Protocol, LSP), specifically the Stardog Language Server[4], which provides syntax checking and error reporting for different ontology languages, including Turtle, the syntax we use for our OWL ontologies. This allows the LLM to consult the language server to get immediate feedback on potential syntax errors. This integration could be further expanded to use the LSP capabilities to automatically fix certain minor issues without needing the LLM.

The code is further validated for syntax correctness by loading it as a graph in Python using the RDFLib library, which also provides error reporting for syntax issues. Finally, the generated ontology is loaded into a reasoner (using the owlready2 library) to check for logical consistency and to ensure that the class restrictions are properly defined. In this stage we are particularly interested in reporting *OwlReadyInconsistentOntologyError*.

These errors are reported in a human understandable way to the LLM, and logged for further analysis. The LLM is then prompted to revise the generated ontology based on the feedback received, and the process is repeated until a syntactically correct ontology is produced.

7.2. Competency Question validation

Once the ontology is syntactically correct, the next validation step is to check the structure of the ontology against a set of competency questions (CQs) — natural language or SPARQL queries used to validate an ontology. These CQs fall in two categories: (i) those that check the structure of the ontology, ensuring for instance that the compliant and violating subclasses are properly defined and disjoint; and (ii) those that check the reasoning capabilities of the ontology, ensuring that compliant and violating behaviours are correctly classified based on the defined norms.

The CQs used in this work were manually encoded as targeted, structural tests to be used as a small suite of machine-checkable engineering invariants (e.g., correct subclassing, explicit disjointness, etc). These invariants are effective at catching generation errors that escape purely syntactic checks and are therefore encoded as executable SPARQL tests that run independently of the LLM.

After some experimentation we inserted the natural language version of the CQs into the prompt to provide context for the LLM, but intentionally kept the testing as part of the validation phase. The LLM may still fail to properly generate exactly valid code, so the CQ tests act as an independent verification layer that returns precise diagnostics into the generate-validate-revise loop.

The CQs are developed ad hoc for the desired ontology, as they need to evaluate the specific structure defined in the prompt. Specific tools or methodologies for generating CQs exist [31, 32] but they were not used in this stage. Their inclusion may be evaluated in further developments.

The first CQ set is focused on the overall structure of the ontology. Examples of such CQs include:

- Which rule classes are not properly subclasses of their implied parent? (Handles both R-code branches and Default/Mitigated leaves).
- Which sibling outcomes are missing explicit 'disjointWith' assertions?
- Does every subclass of the subclasses of Vehicle have both a Compliant and a Violating subclass?
- Do all subclasses of subclasses of the Root (and its children) use standard suffixes (Compliant, Violating)?

[4]Available at https://github.com/stardog-union/stardog-language-servers

- Does every outcome class with a Rule ID (e.g., R171) actually belong to a Root with the correct Base Name?

The natural language CQs are translated into SPARQL (the RDF query language) queries that are executed against the generated ontology, and the results are reported back to the LLM. As these are custom queries, their translation is a manual process. More advanced tools for adding, managing, and translating CQs would be useful, but is not yet part of the system. Listing 2 shows as an example the CP *Which sibling outcomes are missing explicit 'disjointWith' assertions?* expressed as a SPARQL query.

```
SELECT DISTINCT ?Parent ?ClassA ?ClassB WHERE {
  ?ClassA rdfs:subClassOf ?Parent .
  ?ClassB rdfs:subClassOf ?Parent .
  FILTER(?Parent != owl:Thing)
  FILTER(?ClassA != ?ClassB)
  FILTER(REGEX(STR(?ClassA), 'Compliant|Violating'))
  FILTER(REGEX(STR(?ClassB), 'Compliant|Violating'))
  FILTER NOT EXISTS { ?ClassA owl:disjointWith ?ClassB }
  FILTER NOT EXISTS { ?ClassB owl:disjointWith ?ClassA }
}
```

Listing 2: SPARQL CQ test for disjoint sibling classes.

If any of the CQs fail, the LLM is prompted to revise the ontology based on the feedback received, and the process is repeated until all CQs are satisfied. In the current state of the pipeline we do not carry over context from one LLM call to the other, so we can deterministally pass the information we consider to be relevant.

The second set of CQs can be considered as a form of test scenario evaluation, where specific scenarios are defined and the expected classification (compliant or violating) is checked against the output of the OWL reasoner. If the result of the reasoner is incorrect, the error is raised and the ontology snippet sent back to the LLM for revision, with feedback on the specific issues that need to be addressed.

This can be done with CQs (SPARQL queries) or by instantiating in the ontology individuals that correspond to the scenarios and checking their classification.

The basic SPARQL query used for compliance checking could for instance be:

```
SELECT ?subject WHERE { ?subject rdf:type :VehicleR172Compliant. }
```

The LLM can also be prompted to generate test scenarios, or to convert natural language scenarios to the relevant OWL representation, which can then be used for validation. This would allow for a more flexible validation process, that can be tailored to the specific ontology (classes and properties).

7.2.1. Ontology Pitfalls validation

Additional validation steps can be added to check for common pitfalls in ontology engineering, such as the presence of synonym classes, incorrect built-in relationships, or missing properties. These checks can be performed using reasoners and ontology analysis tools, and the results can be used to provide feedback to the LLM for further revision.

Examples of the reasoners and tools are the HermiT reasoner [33], which can be used to check for logical consistency and to identify potential issues in the ontology structure; and the OOPS! (Ontology Pitfall Scanner!) tool [34], which can be used to identify common pitfalls in ontology engineering, such as missing annotations, incorrect use of properties, or the presence of redundant classes.

The first is accessed via the owlready2 Python library, as mentioned previously, while the second is accessed via its API, which allows us to submit the generated ontology for analysis and receive a report on any identified pitfalls.

While there may be some overlap between the CQs and the ontology pitfall checks, they serve different purposes. The CQs are designed to check for domain-specific structural and logical requirements of the ontology, while the ontology pitfall checks are designed to identify common issues in ontology

```
system_prompt = """
  Your task is to fix the provided OWL Turtle code based on the error message from a validator.
"""

user_prompt = f"""
  Here is the original Turtle code:
  {content}
  The Turtle code has the following issues:
  {error_msg}
  Provide a corrected version of the code above using the identified structure.
"""
```

Listing 3: The structure of the error evaluation prompt.

engineering that may not be specific to the domain or structure of the ontology. Both types of checks are important for ensuring the quality and correctness of the generated ontology.

The *error-correcting* prompt skeleton can be seen in Listing 3. The system prompt is a very simple instruction, while the main content is in the user prompt: the existing (incorrect) ttl code and the error messages reported by the automated assessment. In this phase the prompt itself is less important than the error message (error_msg) passed as context

8. Example Process

Let us now consider the translation process for a specific rule from the UK Highway Code, Rule 171, which states that,

> You **MUST** stop behind the line at a junction with a 'Stop' sign and a solid white line across the road. Wait for a safe gap in the traffic before you move off.

The LLM identifies the structure of the rule, extracting the different components as mentioned in the prompt, and creates the corresponding OWL representation, cf. Listing 4. The generated ontology follows correctly the structure defined in the prompt, with the *Compliant* and *Violating* subclasses properly defined as disjoint, and both subclasses of the *situation* class `VehicleAtJunctionWithStopAndSolidWhiteLine`. The LLM identifies as properties `stopBehindLine` and `waitForSafeGap`, defined as functional datatype properties with boolean range, and the class restrictions correctly capture the compliant and violating behaviours as defined in the original rule.

8.1. Batch Rule Generation

The same process can be applied to a batch of rules, with the LLM generating the corresponding OWL representation for each rule in sequence. Once a rule is correctly generated it can be added to a single ontology file, which is passed as context to the LLM for the generation of the next rule. This allows the LLM to build on the previously generated content, ensuring consistency across the generated ontology and allowing for the reuse of concepts and properties as needed.

For example, after generating the representation of rule 171, the LLM can reuse (if necessary) the properties `stopBehindLine` and `waitForSafeGap` when generating the representation for another rule that involves similar actions or conditions, thus ensuring a more coherent and interconnected ontology.

Each rule is first validated individually for syntax and structure, and then the entire ontology is validated as a whole to ensure that all the generated rules are consistent with each other and that the overall structure of the ontology is sound.

Rule 172, for instance, states that,

> The approach to a junction may have a 'Give Way' sign or a triangle marked on the road. You **MUST** give way to traffic on the main road when emerging from a junction with broken white lines across the road.

In one of the generated ontology logs we can see that the LLM correctly identified and modelled this simple rule, but failed to set the `VehicleR172Compliant` and `VehicleR172Violating` classes as

```
# --- DATATYPE PROPERTY DEFINITIONS ---
:stopBehindLine rdf:type owl:DatatypeProperty ;
                rdfs:domain :Vehicle ;
                rdfs:range xsd:boolean ;
                rdf:type owl:FunctionalProperty .

:waitForSafeGap rdf:type owl:DatatypeProperty ;
                rdfs:domain :Vehicle ;
                rdfs:range xsd:boolean ;
                rdf:type owl:FunctionalProperty .

# --- CLASSES ---
:VehicleAtJunctionWithStopAndSolidWhiteLine rdf:type owl:Class ;
  owl:equivalentClass [
    owl:intersectionOf ( [
      rdf:type owl:Restriction ;
      owl:onProperty :junctionHasSolidWhiteLine ;
      owl:hasValue "true"^^xsd:boolean ;
    ] [
      rdf:type owl:Restriction ;
      owl:onProperty :junctionHasStopSign ;
      owl:hasValue "true"^^xsd:boolean ;
    ] ) ;
  ] ;
  rdfs:subClassOf :Vehicle .

:VehicleR171Compliant owl:disjointWith :VehicleR171Violating .

:VehicleR171Compliant rdf:type owl:Class ;
  owl:equivalentClass [
    owl:intersectionOf ( [
      rdf:type owl:Restriction ;
      owl:onProperty :stopBehindLine ;
      owl:hasValue "true"^^xsd:boolean ;
    ] [
      rdf:type owl:Restriction ;
      owl:onProperty :waitForSafeGap ;
      owl:hasValue "true"^^xsd:boolean ;
    ] ) ;
  ] ;
  rdfs:subClassOf :VehicleAtJunctionWithStopAndSolidWhiteLine .

:VehicleR171Violating rdf:type owl:Class ;
  owl:equivalentClass [
    owl:unionOf ( [
      rdf:type owl:Restriction ;
      owl:onProperty :stopBehindLine ;
      owl:hasValue "false"^^xsd:boolean ;
    ] [
      rdf:type owl:Restriction ;
      owl:onProperty :waitForSafeGap ;
      owl:hasValue "false"^^xsd:boolean ;
    ] ) ;
  ] ;
  rdfs:subClassOf :VehicleAtJunctionWithStopAndSolidWhiteLine .
```

Listing 4: LLM generated OWL representation of Rule 171 from the UK Highway Code.

disjoint, which was correctly identified in the validation phase and sent back to the LLM for revision. The CQ validation process presented a log of the failed test, including the “Which sibling outcomes are missing explicit ’disjointWith’ assertions?” test (Listint 2).

Having identified the issue, the LLM was prompted to revise the generated ontology, and the process

was repeated until the ontology passed all the validation checks. In this case the LLM correctly added the missing assertions, and the revised ontology passed the validation phase.

With the batch generation, the generated ontology can also be evaluated in its entirety, to check if there is any consistency issue across different rules, such as incorrect subclassing or reuse of properties, which can be identified and corrected in the same way as for individual rules.

There is no guarantee that the LLM will always produce the same output, even keeping the same prompt and context, so the validation phase is crucial to ensure that the generated ontology is correct and consistent. The feedback loop allows for iterative refinement of the ontology, with the LLM being guided to correct any issues identified during validation, thus improving the overall quality and reliability of the generated ontology.

9. LLM Comparison

In the evaluation phase, the pipeline was run using different LLMs - Google (Gemini 2.5 and Gemma3), Mistral (Small 3.1 24B), Claude (Haiku 4.5), and Meta (Llama 3.3 70B Instruct). This compares the models in terms of their ability to generate accurate and consistent ontologies from the same set of legal norms and to compare what the generated ontologies would look like.

While the output of any LLM is not deterministic and can vary across different runs, we can see that the prompt has already a significant influence on the output structure. Nonetheless, different LLMs can still produce different outputs, possibly due to differences in their training data, fine-tuning process, and the inherent randomness in the generation. This leads to variations in the detail, the structure of the generated ontology, and the class and property naming. Smaller models may also struggle more to extract and represent the logic structure of the norms.

The code in Listing 5 shows the different output of the Gemini (latest tested version being google/gemini-2.5-flash) and Mistral (latest tested version being mistralai/mistral-small-3.1-24b-instruct) models when given rule 172. As can be seen, some models are more verbose (Gemini), while others were more concise (Mistral) in specific ways. Further validation tests can refine the output further.

Gemini had issues with subclassing correctly, as it generated the *Compliant* and *Violating* classes with longer names (e.g., `VehicleAtJunctionWithStopAndSolidWhiteLineR171Compliant` instead of `VehicleR171Compliant`) and made them subclasses of both the situation class and the root `Vehicle` class, which caused issues in the validation phase. Mistral instead correctly generated the compliant and violating classes as subclasses of the situation class and with the correct naming convention.

The syntax validation phase, while critical, also resulted in the more manageable task for smaller models, such as a local quantized version of Gemma (gemma3:1b-it-qat). This model was able to produce syntactically correct output in a reasonable number of iterations (capped at 4) without producing a verbose output.

The CQ validation, depending on the specific CQs, could also be addressed by smaller models. Specific issues, e.g., missing disjointness assertions, were identified and fixed by the smaller model, while the more complex situations related to the correct classification of compliant and violating behaviours were more challenging. In this case a callback to larger models (e.g., mistral-small-3.1-24b-instruct) was necessary to correctly address the issue.

10. Limitations

While the proposed pipeline shows promise in generating and validating legal ontologies from natural language text, there are important limitations to consider. First, the pipeline as presented focuses primarily on syntactic and structural validation of generated ontologies rather than exhaustive inferential completeness. A more in depth validation of the semantic correctness and the reasoning capabilities of the generated ontologies would be necessary to ensure that they accurately capture the intended legal knowledge and can support the desired reasoning tasks. This could be achieved by expanding the competency questions to cover a wider range of scenarios and by including expert review in the

```
# --- DATATYPE PROPERTY DEFINITIONS ---

:giveWayToTrafficOnMainRoad rdf:type
↪  owl:DatatypeProperty ;
    rdfs:domain :Vehicle ;
    rdfs:range xsd:boolean ;
    rdf:type owl:FunctionalProperty .

# --- OBJECT PROPERTY DEFINITIONS ---

:hasRoadMarking rdf:type owl:ObjectProperty ;
    rdfs:domain :Junction ;
    rdfs:range :RoadMarking .

# --- CLASSES ---

:Junction rdf:type owl:Class .

:RoadMarking rdf:type owl:Class .

:BrokenWhiteLines rdf:type owl:Class ;
    rdfs:subClassOf :RoadMarking .

:VehicleEmergingFromJunctionWithBrokenWhiteLines
↪  rdf:type owl:Class ;
    owl:equivalentClass [
        owl:intersectionOf (
            :Vehicle ;
            [
                rdf:type owl:Restriction ;
                owl:onProperty :isEmergingFrom ;
                owl:someValuesFrom [
                    rdf:type owl:Restriction ;
                    owl:onProperty :hasRoadMarking ;
                    owl:hasValue :BrokenWhiteLines
                ]
            ]
        )
    ] ;
    rdfs:subClassOf :Vehicle .

:VehicleR2Compliant rdf:type owl:Class ;
    owl:equivalentClass [
        rdf:type owl:Restriction ;
        owl:onProperty :giveWayToTrafficOnMainRoad ;
        owl:hasValue "true"^^xsd:boolean
    ] ;
    rdfs:subClassOf
    ↪  :VehicleEmergingFromJunctionWithBrokenWhiteLines
    ↪  .

:VehicleR2Violating rdf:type owl:Class ;
    owl:equivalentClass [
        rdf:type owl:Restriction ;
        owl:onProperty :giveWayToTrafficOnMainRoad ;
        owl:hasValue "false"^^xsd:boolean
    ] ;
    rdfs:subClassOf
    ↪  :VehicleEmergingFromJunctionWithBrokenWhiteLines
    ↪  .

:VehicleR2Compliant owl:disjointWith :VehicleR2Violating
↪  .

:isEmergingFrom rdf:type owl:ObjectProperty ;
    rdfs:domain :Vehicle ;
    rdfs:range :Junction .
```

```
# --- DATATYPE PROPERTY DEFINITIONS ---
:giveWayToTraffic rdf:type owl:DatatypeProperty ;
                  rdfs:domain :Vehicle ;
                  rdfs:range xsd:boolean ;
                  rdf:type owl:FunctionalProperty .

# --- CLASSES ---
:VehicleAtJunctionWithGiveWayAndBrokenWhiteLines
↪  rdf:type owl:Class ;
  owl:equivalentClass [
    owl:intersectionOf ( [
      rdf:type owl:Restriction ;
      owl:onProperty :junctionHasBrokenWhiteLines ;
      owl:hasValue "true"^^xsd:boolean ;
    ] [
      rdf:type owl:Restriction ;
      owl:onProperty :junctionHasGiveWaySign ;
      owl:hasValue "true"^^xsd:boolean ;
    ] ) ;
  ] ;
  rdfs:subClassOf :Vehicle .

:VehicleR172Compliant owl:disjointWith
↪  :VehicleR172Violating .

:VehicleR172Compliant rdf:type owl:Class ;
  owl:equivalentClass [
    owl:intersectionOf ( [
      rdf:type owl:Restriction ;
      owl:onProperty :giveWayToTraffic ;
      owl:hasValue "true"^^xsd:boolean ;
    ] ) ;
  ] ;
  rdfs:subClassOf
  ↪  :VehicleAtJunctionWithGiveWayAndBrokenWhiteLines .

:VehicleR172Violating rdf:type owl:Class ;
  owl:equivalentClass [
    owl:intersectionOf ( [
      rdf:type owl:Restriction ;
      owl:onProperty :giveWayToTraffic ;
      owl:hasValue "false"^^xsd:boolean ;
    ] ) ;
  ] ;
  rdfs:subClassOf
  ↪  :VehicleAtJunctionWithGiveWayAndBrokenWhiteLines .
```

Listing 5: Rule 172 generated by Gemini 2.5 and Mistral

validation process. This could be integrated in the currrent pipelin by extending the competency questions to include more complex scenarios and by involving legal experts in the review of the generated ontologies, providing feedback on their accuracy and relevance to the legal domain.

Another possibility, specific to the domain of AVs, would be to integrate the generated ontologies with existing datasets of real-world scenarios, such as the Waymo Open Dataset, or nuScenes[5], to test the reasoning capabilities of the ontology in more realistic and complex situations. This would allow for a comprehensive evaluation of the generated ontologies and their applicability to real-world scenarios.

Second, our evaluation is preliminary in scope (limited number of Highway Code rules and prompting configurations). A more comprehensive evaluation would also include a larger set or rules, as well as rules from different legal domains, to assess the generalizability of the approach. Additionally, a more systematic comparison of different LLMs and prompting strategies would provide insights into the factors that influence the quality and consistency of the generated ontologies and help to assess the capability of the approach to produce reliable outputs across different models and configurations.

11. Conclusion

We have proposed a structured pipeline for the extraction and validation of legal ontologies from natural language text using LLMs. By breaking down the process into distinct phases of translation and validation and by providing clear guidance and feedback to the LLM, we aim to produce accurate and reliable ontologies that can be used for legal reasoning and compliance checking.

An example use case in the domain of AVs was presented, demonstrating the potential of this approach to capture complex legal norms in a structured format. The iterative feedback loop allows for continuous refinement of the generated ontology, ensuring that it meets both syntactic and semantic requirements necessary to support legal reasoning tasks.

The pipeline is designed to be modular, as integration with different LLMs, prompting strategies, and validation techniques can be easily achieved. Future work will focus on expanding the validation process, exploring additional use cases, and further refining the prompting strategies to enhance the quality and applicability of the generated ontologies.

The pipeline is a promising approach to address the knowledge acquisition bottleneck in ontology engineering. There are several directions for future work to make the approach more effective and applicable. The validation process can be expanded to include more comprehensive and domain-specific competency questions, as well as additional validation through expert review or stress testing with real-world scenarios. This would ensure that the generated ontologies not only adhere to syntactic and structural requirements but also accurately capture the intended legal knowledge and reasoning capabilities. The scenario tests could be expanded, where integrating existing datasets would improve the capability and adaptability of the system.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] T. J. M. Bench-Capon, P. R. S. Visser, Ontologies in legal information systems; the need for explicit specifications of domain conceptualisations, in: Proceedings of the 6th international conference on Artificial intelligence and law, ICAIL '97, Association for Computing Machinery, 1997, pp. 132–141. URL: https://dl.acm.org/doi/10.1145/261618.261646. doi:10.1145/261618.261646.

[2] E. A. FEIGENBAUM, Knowledge engineering: The applied side of artificial intelligence, Annals of the New York Academy of Sciences 426 (1984) 91–107. doi:10.1111/j.1749-6632.1984.tb16513.x.

[5] Available respectively at https://waymo.com/open/ and https://www.nuscenes.org/

[3] H. T. Nguyen, W. Fungwacharakorn, M. M. Zin, R. Goebel, F. Toni, K. Stathis, K. Satoh, Llms for legal reasoning: A unified framework and future perspectives 58 (2025) 106165. doi:https://doi.org/10.1016/j.clsr.2025.106165.

[4] J. Zhao, W. Zhao, B. Deng, Z. Wang, F. Zhang, W. Zheng, W. Cao, J. Nan, Y. Lian, A. F. Burke, Autonomous driving system: A comprehensive survey, Expert Systems with Applications 242 (2024) 122836. doi:10.1016/j.eswa.2023.122836.

[5] K. Manas, M. Keser, A. Knoll, Integrating legal and logical specifications in perception, prediction, and planning for automated driving: A survey of methods, 2025. doi:10.48550/ARXIV.2510.25386.

[6] X. Ma, L. Song, C. Zhao, S. Wu, W. Yu, W. Wang, L. Yang, H. Wang, Law compliance decision making for autonomous vehicles on highways, Accident Analysis & Prevention 204 (2024) 107620. URL: https://www.sciencedirect.com/science/article/pii/S0001457524001659. doi:10.1016/j.aap.2024.107620.

[7] O.-R. A. D. O. Committee, Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles, SAE international, 2021.

[8] M. J. Sergot, F. Sadri, R. A. Kowalski, F. Kriwaczek, P. Hammond, H. T. Cory, The british nationality act as a logic program 29 (1986) 370–386. doi:10.1145/5689.5920.

[9] E. Horner, C. Mateis, G. Governatori, A. Ciabattoni, Toward robust legal text formalization into defeasible deontic logic using llms (2025). doi:10.48550/arXiv.2506.08899.

[10] J. Mohun, A. Roberts, Cracking the code: Rulemaking for humans and machines, Text, 2020. URL: https://www.oecd-ilibrary.org/governance/cracking-the-code_3afe6ba5-en. doi:10.1787/3afe6ba5-en.

[11] J. Collenette, L. A. Dennis, M. Fisher, Advising autonomous cars about the rules of the road, Electronic Proceedings in Theoretical Computer Science 371 (2022) 62–76. URL: http://arxiv.org/abs/2209.14035. doi:10.4204/EPTCS.371.5. arXiv:2209.14035 [cs].

[12] A. Rizaldi, J. Keinholz, M. Huber, J. Feldle, F. Immler, M. Althoff, E. Hilgendorf, T. Nipkow, Formalising and monitoring traffic rules for autonomous vehicles in isabelle/HOL, in: N. Polikarpova, S. Schneider (Eds.), Integrated Formal Methods, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2017, pp. 50–66. doi:10.1007/978-3-319-66845-1_4.

[13] H. Bhuiyan, G. Governatori, A. Rakotonirainy, M. W. Wong, A. Mahajan, Driving decision making of autonomous vehicle according to queensland overtaking traffic rules, The Review of Socionetwork Strategies 17 (2023) 233–254. URL: https://doi.org/10.1007/s12626-023-00147-x. doi:10.1007/s12626-023-00147-x.

[14] J. Breuker, A. Valente, R. Winkels, Legal ontologies in knowledge engineering and information management 12 (????) 241–277. doi:10.1007/s10506-006-0002-1.

[15] T. Van Engers, A. Boer, J. Breuker, A. Valente, R. Winkels, Ontologies in the legal domain, Springer US, 2008, pp. 233–261. URL: https://doi.org/10.1007/978-0-387-71611-4_13. doi:10.1007/978-0-387-71611-4_13.

[16] V. Leone, L. Di Caro, S. Villata, Taking stock of legal ontologies: a feature-based comparative analysis, Artificial Intelligence and Law 28 (2020) 207–235. doi:10.1007/s10506-019-09252-1.

[17] K. D. Ashley, Artificial Intelligence and Legal Analytics: New Tools for Law Practice in the Digital Age, Cambridge University Press, Cambridge, 2017. URL: https://www.cambridge.org/core/books/artificial-intelligence-and-legal-analytics/E7D705EEF392501A1DB180645917E7E0. doi:10.1017/9781316761380.

[18] E. Francesconi, G. Governatori, Patterns for legal compliance checking in a decidable framework of linked open data 31 (2023) 445–464. URL: https://doi.org/10.1007/s10506-022-09317-8. doi:10.1007/s10506-022-09317-8.

[19] G. Guizzardi, A. Botti Benevides, C. M. Fonseca, D. Porello, J. P. A. Almeida, T. Prince Sales, Ufo: Unified foundational ontology, Applied Ontology 17 (2022) 167–210. doi:10.3233/ao-210256.

[20] D. V. Dang, H. D. Nguyen, H. Ngo, V. T. Pham, D. Nguyen, Information retrieval from legal documents with ontology and graph embeddings approach, in: H. Fujita, Y. Wang, Y. Xiao, A. Moonis (Eds.), Advances and Trends in Artificial Intelligence. Theory and Applications, Springer

Nature Switzerland, 2023, pp. 300–312. doi:10.1007/978-3-031-36819-6_27.
[21] R. Provine, C. Schlenoff, S. Balakirsky, S. Smith, M. Uschold, Ontology-based methods for enhancing autonomous vehicle path planning, Robotics and Autonomous Systems 49 (2004) 123–133. URL: https://www.sciencedirect.com/science/article/pii/S0921889004001423. doi:10.1016/j.robot.2004.07.020.
[22] R. Regele, Using ontology-based traffic models for more efficient decision making of autonomous vehicles, 2008, pp. 94–99. URL: https://ieeexplore.ieee.org/document/4488328. doi:10.1109/ICAS.2008.10, ISSN: 2168-1872.
[23] S. Ulbrich, T. Nothdurft, M. Maurer, P. Hecker, Graph-based context representation, environment modeling and information aggregation for automated driving, 2014, pp. 541–547. URL: https://ieeexplore.ieee.org/document/6856556. doi:10.1109/IVS.2014.6856556, ISSN: 1931-0587.
[24] L. Huang, H. Liang, B. Yu, B. Li, H. Zhu, Ontology-based driving scene modeling, situation assessment and decision making for autonomous vehicles, 2019, pp. 57–62. URL: https://ieeexplore.ieee.org/document/8935984. doi:10.1109/ACIRS.2019.8935984.
[25] M. Hülsen, J. M. Zöllner, C. Weiss, Traffic intersection situation description ontology for advanced driver assistance, 2011, pp. 993–999. URL: https://ieeexplore.ieee.org/document/5940415. doi:10.1109/IVS.2011.5940415, ISSN: 1931-0587.
[26] M. Dupuis, Vehicle testing in cyber-physical environments: Enabled by asam standards, in: 2025 IEEE International Automated Vehicle Validation Conference (IAVVC), 2025, pp. 1–6. doi:10.1109/IAVVC61942.2025.11219515.
[27] H. Babaei Giglou, J. D'Souza, S. Auer, LLMs4OL: Large Language Models for Ontology Learning, Springer Nature Switzerland, 2023, p. 408–427. doi:10.1007/978-3-031-47240-4_22.
[28] G. Ciatto, A. Agiollo, M. Magnini, A. Omicini, Large language models as oracles for instantiating ontologies with domain-specific knowledge, Knowledge-Based Systems 310 (2025) 112940. URL: https://www.sciencedirect.com/science/article/pii/S0950705124015740. doi:10.1016/j.knosys.2024.112940.
[29] M. A. Cappelli, G. Di Marzo Serugendo, Methodological exploration of ontology generation with a dedicated large language model, Electronics 14 (2025) 2863. URL: https://www.mdpi.com/2079-9292/14/14/2863. doi:10.3390/electronics14142863.
[30] A. Z. Wyner, W. Peters, On rule extraction from regulations, in: K. Atkinson (Ed.), Legal Knowledge and Information Systems - JURIX 2011: The Twenty-Fourth Annual Conference, University of Vienna, Austria, 14th-16th December 2011, volume 235 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2011, pp. 113–122. URL: https://doi.org/10.3233/978-1-60750-981-3-113. doi:10.3233/978-1-60750-981-3-113.
[31] C. M. Keet, Z. Mahlaza, M.-J. Antia, Claro: A controlled language for authoring competency questions (2019) 3–15.
[32] R. Alharbi, V. Tamma, F. Grasso, T. R. Payne, A review and comparison of competency question engineering approaches, in: M. Alam, M. Rospocher, M. van Erp, L. Hollink, G. A. Gesese (Eds.), Knowledge Engineering and Knowledge Management, Springer Nature Switzerland, 2025, pp. 271–290. doi:10.1007/978-3-031-77792-9_17.
[33] R. D. Shearer, B. Motik, I. Horrocks, Hermit: A highly-efficient owl reasoner., in: Owled, volume 432, 2008, p. 91.
[34] 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.