2026 European Conference on Computing in Construction
Corfu, Greece
July 12–15, 2026

A NEURO-SYMBOLIC AI PIPELINE FOR GENERATING SHACL-BASED DIGITAL BUILDING REGULATIONS

Alex Donkers and Ekaterina Petrova
Eindhoven University of Technology, Eindhoven, The Netherlands

Abstract

Automated compliance checking requires the transformation of natural language building regulations into machine-interpretable code. Current Large Language Model (LLM) approaches for generating digital rules often suffer from hallucinations and structural inconsistencies. This paper proposes a neuro-symbolic AI pipeline that grounds LLM reasoning in symbolic logic and domain ontologies to generate SHACL shapes. Evaluation against Dutch construction regulations demonstrates that the pipeline successfully generates functional SHACL shapes for relational, mathematical, and conditional constraints. This work contributes a scalable framework for translating complex legislative code into executable, interoperable digital rule sets for the AECO industry.

Introduction

Digital transformation of the Architecture, Engineering, Construction, and Operations (AECO) industry is increasingly dependent on digitization of building regulations. Traditionally, regulations are authored as natural language documents, which are inherently ambiguous and require manual interpretation by experts. Not only is this process time-consuming and prone to human error, it also limits innovations in BIM-based design workflows. To bridge the gap between human-readable regulations and machine-interpretable code, the research community has explored various ways of digitizing building regulations. One of the emerging ideas is the use of the Shapes Constraint Language (SHACL) to validate Linked Building Data (Nuyts et al., 2024).

However, manual creation of SHACL shapes is a highly specialized task that requires deep knowledge of both semantic web technologies and the specific domain of the regulations, such as fire safety. Recent advancements in Natural Language Processing (NLP), specifically Large Language Models (LLMs), offer a potential solution for automating this process. These LLMs introduce risks when applied to the regulatory domain, including generating terms that do not match with the target ontology, and the appearance of structural inconsistencies in the digital rule representation. Such challenges prevent digital rules from being operationally executable and negatively affect the human trust in these rules.

To overcome these issues, this paper proposes a neuro-symbolic AI pipeline to generate SHACL-based digital building regulations. The aim is to find how symbolic logic representations, such as ontologies and SHACL templates, can be used to enhance reasoning of the LLM. After exploring related work in Section 2, Section 3 explains this neuro-symbolic AI approach in more detail. The approach is tested with Dutch construction regulations in Section 4, after which Section 5 and 6 discuss and conclude this work.

Related work

Digital building codes

Digitization of building codes is a long-standing research topic, and systems to assist in (automated) compliance checking (ACC) have been around for over 50 years (Amor and Dimyadi, 2021). A shared understanding of requirements for such digital building codes is presented by Noardo et al. (2022). These include that normative text should be interpretable by humans and machines, and that we should aim for interoperability through shared dictionaries. Early ideas on the use of Artificial Intelligence (AI) to assist ACC exist since roughly 1990 (Niwa, 1989). Much work in digitization of rules was driven by developments in Building Information Modeling (BIM). In recent years, developments in Natural Language Processing and Large Language Models drove research in rule digitization (Fuchs, 2021).

Ontologies for digital building codes

Ontologies can be used to enhance the interoperability of digital building regulations. As described by Amor and Dimyadi (2021), compliance checking happens on the intersection of building information and normative knowledge, and therefore ontologies representing both sides of the intersection are needed. The research community has increasingly adopted the Linked Building Data (LBD) concept, and the W3C LBD community group presented ontologies that can represent topological building information (Building Topology Ontology, BOT, by Rasmussen et al. (2021)), building elements (Built Element Ontology, BEO, by Ramonell Cazador (2026)), properties (Ontology for Property Management, by Rasmussen et al. (2018)), and geometry (Ontology for Managing Geometry, OMG, Wagner et al. (2019)). Next to these general building ontologies, a wide variety of domain ontologies exists that enable modeling specific domain knowledge. Ontologies representing (AECO-specific) normative knowledge are

more scarce (Werbrouck et al., 2026).

NLP and the generation of digital building codes

The transformation of natural language regulations into machine-interpretable formats has progressed from manual rule interpretation to computer-aided interpretation. A clear example is the semantic tagging of rules with the RASE (Requirement, Applicability, Selection, Exception) method, that moves from manual tagging to human-in-the-loop LLM-based tagging (Al-Turki et al., 2024). The transition is driven by using NLP and domain ontologies to map regulatory terms to semantic concepts, frequently targeting IFC-based compliance (Zhang and El-Gohary, 2016). Subsequent efforts expanded these NLP techniques to extract spatial terms for reasoning (Li et al., 2016) and used models like BERT to transform fire safety regulations into SPARQL queries via semantic labeling and keyword matching (Zheng et al., 2022). Recent research in rule interpretation highly focused on the application of LLMs. Fuchs et al. (2024) applied ChatGPT to transform natural language to LegalRuleML, while Chiappini et al. used ChatGPT to convert text to SPARQL. Currently, the field is moving toward neuro-symbolic AI, which combines neural AI approaches with formal symbolic logic. While early work relied on structured spreadsheets (Costa et al., 2024), current work uses ontologies to improve the reasoning of LLMs (Donkers and Petrova, 2025).

Methodology

System architecture

Figure 1 shows the system architecture for the SHACL converter. It contains a column with input data (the regulatory statement, a domain ontology, W3C LBD ontologies, and SHACL shape instructions), a middle block that prepares input for the LLM (represented by the colored arrows), and the last column with the LLM functionalities. The following subsections discuss the internals of each block in detail.

Linguistic pre-processing

The process begins with the extraction of features from a regulatory statement. The spaCy NLP framework is used to identify noun chunks, which are base phrases with a noun as their head. These noun chunks are later used by the LLM to correctly map terms in the regulations to terms in the ontology. Using noun chunks instead of single tokens in the sentence overcomes incorrect mapping of nouns to terms in the ontology. For example, "A portable fire extinguisher" has universal part-of-speech (POS) tags `DET-ADJ-NOUN-NOUN`, however, "fire" and "extinguisher" should not be mapped to ontology terms in isolation, as "fire" is a compound modifier of "extinguisher". Individual tokens are also extracted, where specific POS tags are excluded to not clutter the ontology term selection. The `nl_core_news_lg` model is used, given that this paper converts Dutch regulations.

Ontologies

To prevent the LLM from hallucinating terms, domain ontologies are used that contain classes that should be used by the LLM. This paper makes use of the Fire-BIM Building Ontology, which is an extensive collection of terms related to the fire safety domain. To enable bridging from these terms to the domain of building information modeling, a subset of the W3C Linked Building Data ontologies (Building Topology Ontology (BOT), Built Element Ontology (BEO), Ontology for Property Management (OPM), Ontology for Geometry Management (OMG)) is used in this work. Listing 1 shows how these ontologies could be linked to make linked building data statements about fire compartments and the spaces they contain.

```
inst:FireCompartment_1 a fbo:FireCompartment ;
    bot:hasZone inst:Space_1, inst:Space_2 ;
    omg:hasGeometry inst:Geometry_1 .
inst:Space_1 a fbo:Space .
inst:Space_2 a fbo:Space .
```

Listing 1: RDF snippet of linking a fire compartment to the spaces it contains

Optimizing ontology representation for LLMs - Encoding rdfs:label and rdfs:comment into vectors - Cosine similarity to measure the distance between the noun chunks and the embedded ontology terms - Selection: n=30

One of the major limitations of LLMs is the amount of tokens that can be used in the prompt, as larger token numbers are more costly and tend to worsen the LLM results. Token counts of the used ontologies (retrieved programmatically using the `google-genai` SDK's native `count_tokens` method to ensure alignment with the model's tokenizer) are presented in Table 1. Striking are the high token counts of BOT (caused by long rdfs:comments in multiple languages) and FBO and BEO (caused by an extensive amount of classes).

Table 1: Ontology token count in Gemini's tokenizer

Ontology	Tokens
fbo.ttl	96.384
bot.ttl	32.042
opm.ttl	4.207
omg.ttl	4.024
beo.ttl	51.900
Total	**188.557**

To limit the high token counts, two methods are applied. For ontologies with `rdfs:comments` in multiple languages, all terms are queried using `FILTER(LANG(?comment)='nl'))` to only select the Dutch comments. For ontologies with a large number of classes and properties, a cosine similarity algorithm is ran between the terms and noun chunks selected during the linguistic pre-processing, and the `rdfs:labels` and

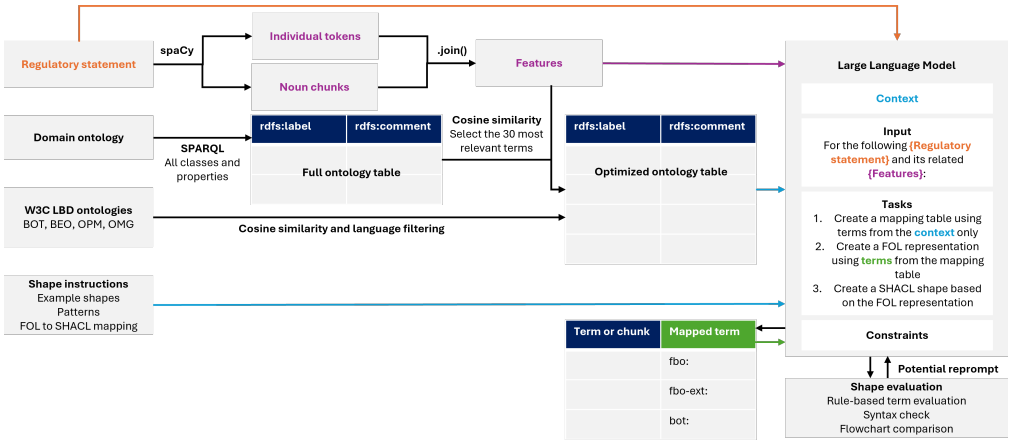

Figure 1: A neuro-symbolic pipeline for generating SHACL shapes

`rdfs:comments` of all terms in the ontology. First, both the regulatory terms and the ontology terms are embedded using the `sentence-transformers/all-mpnet-base-v2` model with Mean Pooling. Then, `sklearn.metrics.pairwise.cosine_similarity` is used to compute the cosine similarity of each entity in the regulatory terms list with each term from the ontologies. For each regulatory term, the top 10 terms are selected, after which these terms are combined and filtered for duplicates. "10" is a rather arbitrary number, that should be evaluated case-by-case, depending on the complexity of regulations and the ontology.

Digital representations of regulations

While SHACL has shown to be able to digitally represent regulations (Donkers and Petrova, 2024), it is a constraint language, not a logic language. Laws are naturally structured as logical implications, where:

$$\forall x(\text{Antecedent}(x) \rightarrow \text{Consequent}(x))$$

First-Order Logic mapping This work aims to preserve this natural structure of the laws (isomorphic modeling) using First-Order Logic (FOL) mapping. Some benefits are that 1) human experts are able to verify the mapping, 2) FOL is implementation agnostic and could be used to generate digital building codes in other formats, and 3) FOL explicitly splits the antecedent and the consequent. The latter if of great importance, as LLMs tend to mix these up when generating SHACL in a zero-shot prompting architecture. The FOL antecedent and consequent can be mapped to other rule markup languages, such as SHACL (`sh:SPARQLTarget` and `sh:PropertyShape`), RASE (Applicability/Selection and Requirement/Exception), IDS (Applicability and Requirements), SWRL (Rule body and Rule head), N3, (LHS and RHS).

There is room for discussion on how to model exceptions in FOL, as one could either model an exception as a negation $(A \wedge \neg E) \rightarrow R$ (making it part of the antecedent) or as a disjunction to the requirement $A \rightarrow (R \vee E)$ (making it part of the consequent). In this paper, we assume exceptions to be modeled using the latter method, as this relates best to how exceptions are written in Dutch regulations, and thus how a rule interpreter should interpret the rule.

SHACL Patterns While FOL serves as an intermediate representation to ensure logical isomorphism with the legal text, SHACL enables making the digital rule operationable. SHACL shapes can be used to validate Linked Building Data graphs for compliance with regulations, and provide means to generate a validation report once the data graph violates the rules dictated in the shapes graph.

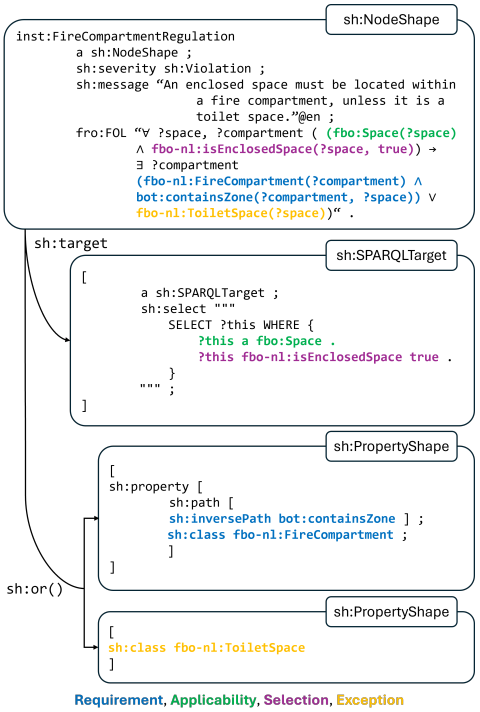

Figure 2: FOL & SHACL representation of a regulation with RASE tags

Figure 2 shows how a text-based regulation can be repersented in FOL, and how that FOL would logically convert to a SHACL shape, where the antecedent is represented in a `sh:SPARQLTarget` and the consequent in

sh:PropertyShapes. The figure also shows how this shape would map to RASE.

While SHACL is a W3C standard, it has a lot of structural ambiguity, meaning rules can be represented using different SHACL representations. These representations do cause SHACL validation engines to behave differently, e.g. by changing how exceptions are modeled, or by different use of logical operaters. An LLM without guidance typically hallucinates a hybrid syntax, making it difficult to evaluate the generated SHACL shape and to predict its constraint behavior. To guide the LLM in generating predictable shapes, a set of standardized SHACL shapes and patterns are prepared. These templates serve to constrain the LLMs output to specific modeling idioms, such as property constraints, inverse relationships, cardinality constraints, topological containment, and logical operators when modeling multiple requirements. This eliminates the structural inconsistencies that occur in zero-shot SHACL generation.

Large language model prompt strategy

A Large Language Model (gemini-2.5-flash) was used via Google's genai Python library. The study applies a structured prompting strategy following In-Context Learning and Constrained Decoding principles. The LLM is provided with explicit Inductive Bias through predefined SHACL templates, minimizing the structural inconsistency of the SHACL generation. Model configurations include temperature=0.0 and max_output_tokens=2048. The prompt contains of 1) Semantic context, 2) Input data, 3) Tasks, 4) Constraints, 5) SHACL generation rules. Those individual items are discussed in the following paragraphs. Listing 2 shows the prompt that was used in Python.

```
You are a Specialist in Semantic Web and Building Regulations.

### SEMANTIC CONTEXT
- Domain ontology: {fbo_str}
- Building ontologies: {w3c_str}
- SHACL templates: {shacl_templates}

### INPUT DATA
Regulatory Text: {regulatory_text}
Extracted Features: {', '.join(features)}

### TASKS
1. MAPPING TABLE: Link statement concepts to context URIs.
2. FIRST-ORDER LOGIC (FOL): Formalize the requirement logic.
3. SHACL SHAPE: Generate a Turtle-formatted constraint shape.

### CONSTRAINTS
- Only use URIs from the provided context.
- If you cannot map a term to the terms from the ontologies in the context, create a new term in the fbo-ext: namespace
- For properties, check directionality. bot:containsZone means that X contains Y, not Y contains X. bot:containsElement means that X contains Y, not Y contains X.
- Output the SHACL shape within code blocks.
- Classes always start with Uppercase, properties lowercase.
- In the FOL representation, preferably use the terms from the mapping table with correct prefixes.

### SHACL generation
- the →'' of the FOL logically splits the FOL into 'applicability' (left of →'') and 'requirements' (right of →''). 'applicability' should be modeled as sh:SPARQLtarget.
- The Rule that needs to be checked (REQUIREMENT) is always part of the consequent, and thus right of the →'', and thus should be in the property node in SHACL.
```

Listing 2: The LLM prompt in Python

Context injection The first part of the prompt involves semantic priming, where the LLM is provided with a domain ontology, a set of building-related W3C LBD ontologies, and SHACL templates. The ontologies are loaded as string-based representations instead of the original Turtle files, based on the optimization method in Section 'Optimizing ontology representation for LLMs'. This significantly reduces the token count of ontologies, e.g. this method optimizes the BOT tokens from 32042 to 2107.

After injecting the semantic context, the input data for the LLM in inserted, being the regulatory text as a string, and a list with features. This feature list is a combination of the noun chunks and the individual tokens selected by the NLP algorithm.

Multi-stage prompting The context is followed by a set of tasks following a multi-stage prompting method, to enable managing the high cognitive load associated with mapping context logic to SHACL. The tasks use a Chain-of-Thought reasoning to decouple the generation of a mapping table, the generation of an FOL representation, and the generation of a SHACL shape. By first mapping features to terms from the ontology in a mapping table, the prompt aims to reduce the number of hallucinated terms. The generation of FOL is then using the mapping table, reusing the mapped terms from the multiple ontologies. The FOL is then converted to a SHACL shape, using the constraints and templates loaded in the prompt. The mapping table, the FOL, and the SHACL shape are all returned by the LLM, enabling human validation.

Evaluation

The generated SHACL shapes are finally evaluated using automated and human-in-the-loop methods. A RegEx-based Python function evaluates the existance of certain syntactical elements in the SHACL shape, such as sh:SPARQLTarget and sh:NodeShape. It also evaluates the existence of the domain ontology prefixes. Secondly, a set of human-in-the-loop evaluation methods were developed. First, human experts can select a list of terms that should be in the returned SHACL shape. A RegEx search looks for the term, allowing for prefixed and unprefixed terms. If not available, the LLM is reprompted, with the previously generated SHACL shape and an error message stating that the term is missing. Secondly, a rule-based method to convert SHACL shapes to the Mermaid

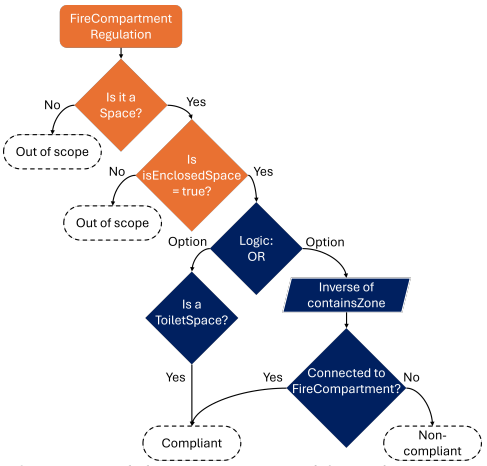

Figure 3: Mermaid diagram generated from the SHACL shape in Figure 2

diagramming language is developed, which enables presenting the SHACL shape as a human-readable flowchart. As Mermaid is code-based, it is possible to include the URIs of terms in the decision tree, making it interactive. This way, a human expert can click on the diagram to open definitions of terms. The antecedent and consequent are colored orange and blue, respectively, to visually help the expert. Finally, two data graphs are modeled for each SHACL shape; one deliberately violating the shape, and one that should pass the check. Since the shapes make use of `sh:SPARQLTarget`, which is a SHACL Advanced Feature, not all SHACL validators are able to run these shapes. The Python-based `pySHACL` engine supports SHACL Advanced Features, which is why an evaluation script is built using this library.

Dataset description To evaluate the developed method, regulations from the Dutch construction law (*'Besluit bouwwerken leefomgeving'*) were converted to SHACL shapes. Steutel (2026) developed a method to convert Dutch fire safety regulations to SHACL. The same set of regulations is manually filtered for regulations that are described by the domain ontology, after which that subset is used in this work. The regulations are categorized using the same categories proposed by Nuyts et al. (2024) to get a sense of the strengths and weaknesses of this approach for each category.

Results

The converter is tested on articles or combinations of articles, as presented in 2. Categories are numbered as follows: 1. information availability, 2. value, 3. relational, 4. mathematical, 5. conditional.

All categories described by Nuyts et al. (2024) are tested and for each of the categories, the converter managed to generate a functional SHACL shape. To test category 1, an Information Requirements rule was modeled to check whether Abox building information was correctly modeled to be checked by rule 4.50(2d). Listing 3 shows the generated SHACL shape. The generator correctly distinguished the antecedent logic (placed in the `sh:SPARQLTarget` and the consequent logic (placed in the `sh:PropertyShape`). While `sh:SPARQLTarget` is not strictly necessary for this specific rule (`sh:target fbo-nl:TechnicalRoom` would suffice), the goal is to generate a set of SHACL shapes that has a stable structure across the set, and future shapes will likely need this modeling method.

Table 2: Article Mapping Results

Article	Category	Mapping	SHACL
4.50(1)	3	●	●
4.50(1)+(2a)	3, 5	●	●
4.50(1)+(2a)+(2b)+(2d)	2, 3, 5	●	●
4.50(1)+4.50(4)	3, 5	●	●
4.50(5)	3	●	●
4.50(1)+4.50(5)+4.50(8)	3, 5	●	○
4.51(2)	3, 4	●	●
4.50(2d) IR	1	●	●

```
fbo-nl:TechnicalRoomShape a sh:NodeShape ;
    sh:target [
        a sh:SPARQLTarget ;
        sh:select """
            SELECT ?this WHERE {
                ?this a fbo-nl:TechnicalRoom .
            }
        """ ;
    ] ;
    sh:property [
        sh:path fbo:hasUsableArea ;
        sh:message "A technical room must have a usable area."@en ;
        sh:minCount 1 ;
    ] .
```

Listing 3: An information requirement rule checking whether Abox building information is suitable for a compliance check

Almost all evaluated articles are categorized as category 3. relational. The SHACL shape generated from article 4.50(1) is shown in Listing 4. Noticable is the different URI representation of the `sh:NodeShape`, which is something that the LLM is not trained for. Listing 4 shows how the LLM correctly interpreted the inverse logic of `bot:containsZone`. The use of `sh:qualifiedValueShape` is somewhat redundant, but does influence the results of the compliance check.

```
<http://www.firebim.org/shapes/enclosed-space-in-fire-compartment> a sh:NodeShape ;
    sh:target [
        a sh:SPARQLTarget ;
        sh:select """"""
            SELECT ?enclosedSpace WHERE {
                ?enclosedSpace a bot:Space .
                ?enclosedSpace fbo-nl:isEnclosedSpace true .
            }
        """""" ;
    ] ;
    sh:property [
        sh:path [ sh:inversePath bot:containsZone ] ;
        sh:class fbo-nl:FireCompartment ;
        sh:qualifiedValueShape [
            a sh:NodeShape ;
            sh:targetClass fbo-nl:FireCompartment ;
```

```
        ] ;
        sh:message ""An enclosed space must be
     contained within a fire compartment."" ;
    ] ."
```

Listing 4: A generated SHACL shape for a relational regulation

Category 2 represents regulations with value checks. Listing 5 shows how the maximum usable area of a technical room and a minimum nominal power of combustion appliances in that technical room are modeled in SHACL by the converter. The units of these values are not yet modeled by the LLM; these are already given in the ontology and can be checked by a separate information requirements check. However, the converter could in the future be extended to include units, e.g. using an extra SHACL template.

```
[ a sh:NodeShape ;
    sh:class fbo-nl:TechnicalRoom ;
    sh:and (
        [ sh:property [ sh:path fbo:hasUsableArea ; sh
     :datatype xsd:decimal ; sh:maxInclusive "50"^^xsd:
     decimal ] ]
        [ sh:not ( [ a sh:NodeShape ;
            sh:property [ sh:path fbo-nl:
     hasCombustionAppliance ; sh:class fbo-nl:
     CombustionAppliance ;
                sh:node [ sh:property [ sh:path fbo-nl
     :hasNominalPower ; sh:datatype xsd:decimal ; sh:
     minInclusive "130"^^xsd:decimal ] ]
            ] ] )
        ]
    )
] .
```

Listing 5: A generated SHACL shape for a value check

Mathematical regulations (category 4) are represented by article 4.51(2). Listing 6 shows 2 of such mathematical functions. The first half of the listing presents a `sh:PropertyShape` that checks whether a fire compartment contains maximum four caravans or ancillary functions. The listing shows that the LLM modeled ancillary functions in the `fbo-ext:` namespace, following the instructions in Listing 2. This is an indication to the ontology engineer that the current domain ontology does not suffice in representing this regulation. The second part of the listing checks whether the total area of these caravans does not exceed 1000 m^2, for which it uses a `sh:SPARQLConstraint` that first queries the area of each individual caravan, then sums these areas, and finally checks whether these areas don't exceed 1000 m^2.

```
fbo-ext:
     FireCompartmentMaxWagonsAndAncillaryFunctionsShape
      a sh:NodeShape ;
  sh:targetClass fbo-nl:FireCompartment ;
  sh:property [
    sh:path bot:containsZone ;
    sh:qualifiedValueShape [
      a sh:NodeShape ;
      sh:or (
        [ sh:class fbo-nl:ResidentialFunctionCaravan ]
        [ sh:class fbo-ext:AncillaryFunction ]
      ) ;
    ] ;
    sh:qualifiedMaxCount 4 ;
    sh:message "Max 4 caravan or ancillary functions."
     @en ;
  ] ;

  sh:sparql [
    a sh:SPARQLConstraint ;
    sh:message "Total area must not exceed 1000 m2."
     @en ;
    sh:select """
      SELECT ?this (SUM(?area) AS ?totalArea)
      WHERE {
        ?this bot:containsZone ?zone .
        ?zone a ?zoneType ; fbo:hasUsableArea ?area .
        FILTER (?zoneType IN (fbo-nl:
     ResidentialFunctionCaravan,
          fbo-ext:AncillaryFunction))
      }
      GROUP BY ?this HAVING (SUM(?area) > 1000)
    """ ;
  ] .
```

Listing 6: A generated SHACL shape for mathematical regulations

Finally, conditional constraints (category 5) can be modeled using the logical operators in SHACL. Figure 2 shows how multiple conditions (in this case exceptions) are modeled by the LLM.

Article 4.50(1)+4.50(5)+4.50(8) did not correctly convert to SHACL. Listing 7 shows the input to the converter. While the mapping from features to terms in the ontology is correctly executed, the generated SHACL shape shows mistakes. A likely cause is the fact that the exception in statement 8 relates to the function of a building, whereas the requirements in statements 1 and 5 relate to a space. Although the LLM generated a syntactically correct SHACL shape, it assumed that `fbo-nl:LightIndustrialFunctionForCultivation` is a class that can be assigned to a space, while this is meant to be a function type of a building. It will therefore in practice not match with Abox building information.

```
1. An enclosed space is located in a fire compartment.
5. A non-enclosed functional area is located in a fire
     compartment.
8. The first and fifth statement do not apply to a
    light industrial function for the cultivation,
    growing, or storage of crops or comparable
    products, with a permanent fire load no greater
    than 150 MJ/m2, determined in accordance with NEN
    6090.
```

Listing 7: A translation of article 4.50(1)+4.50(5)+4.50(8) of the Dutch construction law

```
[ sh:and (
  [ sh:property [
    sh:path rdf:type ;
    sh:hasValue fbo-nl:
     LightIndustrialFunctionForCultivation ]
  ]
  [ sh:property [
    sh:path fbo-nl:hasPermanentFireLoad ;
    sh:maxInclusive 150 ]
  ]
) ]
```

Listing 8: The exception logic of 4.50(8)

Nested exceptions The converter is designed to specify complex exceptions as separate `sh:Nodeshapes`. Listing 5 shows how a complex exception (article 4.50(2d))to the regulation in Figure 2 is modeled by the converter. This rather atomic modeling principle allows a more flexible use of SHACL shapes, allowing end users to query and apply such complex exceptions separately (Donkers and Petrova, 2025), or allowing other SHACL shapes to reuse such exceptions.

Human-in-the-loop validation The generated Mermaid diagrams for each SHACL shape enable human-in-the-loop validation. We validated the generated SHACL shapes using these Mermaid diagrams. Although this method enables non-SHACL-experts to validate SHACL shapes, such methods should still be used with caution. For example, the incorrect use of light industrial functions for cultivation in Listing 8 did result in a syntactically correct SHACL shape, and thus formed a Mermaid diagram. While this Mermaid diagram is arguably easier to read than the SHACL shape, such minor errors are still hard to spot by the human eye.

Validating a data graph To test the functionality of the SHACL shapes in practical use cases, data graphs were created for each SHACL shape, and they were subsequently tested in a `pySHACL`-based script. Listing 9 shows the data graph that was modeled to violate the SHACL shape of article 4.51(2) (which is shown in Listing 6). It violates the rule twice: 1. the fire compartment contains too many caravans or ancillary functions, and 2. the total area of these functions is larger than 1000 m^2.

```
inst:InvalidCompartment a fbo-nl:FireCompartment ;
    bot:containsZone inst:C1, inst:C2, inst:C3, inst:
     C4, inst:A1 .
inst:C1 a fbo-nl:ResidentialFunctionCaravan ;
    fbo:hasUsableArea 250 .
inst:C2 a fbo-nl:ResidentialFunctionCaravan ;
    fbo:hasUsableArea 250 .
inst:C3 a fbo-nl:ResidentialFunctionCaravan ;
    fbo:hasUsableArea 250 .
inst:C4 a fbo-nl:ResidentialFunctionCaravan ;
    fbo:hasUsableArea 250 .
inst:A1 a fbo-ext:AncillaryFunction ;
    fbo:hasUsableArea 200 .
```

Listing 9: Data graph that violates article 4.51(2)

The validation report by pySHACL is printed in Listing 10, and correctly shows the two violations of Article 4.51(2).

```
Conforms: False
Results (2):
Constraint Violation in
     QualifiedValueShapeConstraintComponent:
  Severity: sh:Violation
  Source Shape: *The max count sh:PropertyShape*
  Focus Node: ex:InvalidCompartment
  Result Path: bot:containsZone
  Message: A fire compartment may contain a maximum of
      4 ResidentialFunctionCaravan or AncillaryFunction
     .
Constraint Violation in SPARQLConstraintComponent:
  Severity: sh:Violation
  Source Shape: fbo-ext:
     FireCompartmentMaxWagonsAndAncillaryFunctionsShape
  Focus Node: ex:InvalidCompartment
  Value Node: ex:InvalidCompartment
  Source Constraint: *The sh:SPARQLConstraint*
  Message: The total usable area of
     ResidentialFunctionCaravan and AncillaryFunction
     within a fire compartment must not exceed 1000 m2.
```

Listing 10: Validation report generated by pySHACL

Discussion

The results demonstrate that grounding LLMs in domain-specific ontologies via a neuro-symbolic AI pipeline enables the generation of SHACL shapes out of natural language regulations. However, a few critical challenges remain unanswered.

The current injection of ontology terms uses a table with the `rdfs:label` and `rdfs:comment`. This loses part of the context of each term, e.g. how classes are associated to other classes, and with what properties. This knowledge is normally strongly represented in ontologies, and future work should answer whether injecting these relationships into the context of the LLM improves the results.

The converter takes individual regulatory statements, or manually inserted combinations of statements. It cannot deal with external sources or references to other articles unless they are explicitly included in the regulatory statement variable. Combining this neuro-symbolic pipeline with the regulation graph as presented by Donkers and Petrova (2025) would enable retrieving regulatory statements from a structured RDF database that helps navigating to the correct articles, tables, figures, and so forth.

While the token optimization method succesfully reduced the number of tokens of each ontology, the n=10 retrieval parameter for ontology terms could cause difficulties for complex regulations. More research is necessary to determine what ontology optimization strategy fits to different scenarios. The converter showed difficulties when converting regulations with a large number of requirements and exceptions, especially with multiple levels of nesting.

The choice of SHACL as a regulatory language poses difficulties in practice. Most SHACL validators traverse all parts of a `sh:and`-statement, even if the first statement fails, causing computational overhead. The nested SHACL approach also has difficulties with 'unknowns': SHACL is a binary that assumes that values that are not in the data graph do not exist. Especially in early design stages, some exceptions in the regulations might be unknown (e.g. whether a building has a sprinkler system), and SHACL shapes natively encounter difficulties with distinguishing between violations and data incompleteness. How to solve such practical challenges remains future research, and results of such research should then be reflected in this pipeline.

Conclusion

This paper presented a neuro-symbolic AI pipeline that leverages the strengths of Large Language Models grounded in symbolic logic and domain ontologies to automate the generation of SHACL-based digital building regulations. By using First-Order Logic as an intermediate representation and applying a structured prompting strategy with predefined SHACL templates, the approach significantly reduces the structural inconsistencies and hallucinations typically associated with zero-shot LLM outputs. The evaluation against Dutch construction regulations demonstrated that the pipeline can effectively handle various rule categories, including relational, mathematical, and conditional constraints, while providing human-interpretable validation via Mermaid diagrams.

However, the transition from individual regulatory state-

ments to a fully automated compliance ecosystem requires addressing several complexities. Future research will focus on integrating this pipeline with comprehensive regulation graphs. While the current method processes isolated articles, a graph-based approach would allow the system to navigate and resolve cross-references between different articles, annexes, and external guidelines. Furthermore, extending neuro-symbolic reasoning to interpret non-textual regulatory elements such as tables, formulas, and diagrams is essential for a holistic digitization process. By linking the generative power of LLMs with structured regulation databases, we can move toward a scalable framework capable of translating vast bodies of legislative code into executable, interoperable, and trustworthy digital rule sets for the AECO industry.

Acknowledgements

This paper is funded by the Eureka ITEA4 FireBIM project (22003) and the Netherlands Enterprise Agency.

References

Al-Turki, D., Hettiarachchi, H., Medhat Gaber, M., Abdelsamea, M. M., Basurra, S., Iranmanesh, S., Saadany, H., and Vakaj, E. (2024). Human-in-the-loop learning with llms for efficient rase tagging in building compliance regulations. IEEE Access, 12:185291–185306.

Amor, R. and Dimyadi, J. (2021). The promise of automated compliance checking. Developments in the Built Environment, 5:100039.

Costa, G., Vakaj, E., Beach, T., Lavikka, R., Lefrançois, M., Zimmermann, A., Mecharnia, T., Alexiev, V., Dridi, A., Hettiarachchi, H., and Keberle, N. (2024). Formalization of building codes and regulations in knowledge graphs. In Noardo, F. and Fauth, J., editors, Digital Building Permit Conference 2024, pages 142–146, Barcelona, Spain.

Donkers, A. and Petrova, E. (2024). Converting fire safety regulations to shacl shapes using natural language processing. In NLP4KGC 2024 : Natural Language Processing for Knowledge Graph Creation 2024, CEUR Workshop Proceedings. CEUR-WS.org.

Donkers, A. and Petrova, E. (2025). Retrieval of digital regulations using neuro-symbolic ai with graphrag. In Proceedings of the Digital Building Permit Conference 2025, Singapore. Springer Nature.

Fuchs, S. (2021). Natural language processing for building code interpretation: systematic literature review report. August, 21:2023.

Fuchs, S., Witbrock, M., Dimyadi, J., and Amor, R. (2024). Using large language models for the interpretation of building regulations.

Li, S., Cai, H., and Kamat, V. R. (2016). Integrating natural language processing and spatial reasoning for utility compliance checking. Journal of Construction Engineering and Management, 142(12):04016074.

Niwa, K. (1989). Use of artificial intelligence confirming compliance to building codes during process of making architectural drawings. Future Gener. Comput. Syst., 5(1):129–130.

Noardo, F., Guler, D., Fauth, J., Malacarne, G., Mastrolembo Ventura, S., Azenha, M., Olsson, P.-O., and Senger, L. (2022). Unveiling the actual progress of digital building permit: Getting awareness through a critical state of the art review. Building and Environment, 213:108854.

Nuyts, E., Bonduel, M., and Verstraeten, R. (2024). Comparative analysis of approaches for automated compliance checking of construction data. Advanced Engineering Informatics, 60:102443.

Ramonell Cazador, C. (2026). Built Element Ontology (BEO). `https://w3id.org/beo\#`. Revision 1.0.

Rasmussen, M., Lefrançois, M., Schneider, G., and Pauwels, P. (2021). Bot: The building topology ontology of the w3c linked building data group. Semantic Web, 12(1):143–161.

Rasmussen, M. H., Lefrançois, M., Bonduel, M., Hviid, C. A., and Karlshøj, J. (2018). OPM: An ontology for describing properties that evolve over time. In Proceedings of the 6th Linked Data in Architecture and Construction Workshop, London, United Kingdom.

Steutel, B. T. R. (2026). A framework for fire safety: Processing regulatory texts for automated compliance checking using natural language processing and semantic web techniques. Master's thesis, Eindhoven University of Technology.

Wagner, A., Bonduel, M., Pauwels, P., and Uwe, R. (2019). Relating geometry descriptions to its derivatives on the web. In Proceedings of the 2019 European Conference for Computing in Construction, pages 304–313. European Council on Computing in Construction (EC3).

Werbrouck, J., Hagedorn, P., Donkers, A., and Fauth, J. (2026). Semantic Web Technologies for Building Permitting.

Zhang, J. and El-Gohary, N. M. (2016). Semantic nlp-based information extraction from construction regulatory documents for automated compliance checking. Journal of Computing in Civil Engineering, 30(2):04015014.

Zheng, Z., Zhou, Y.-C., Lu, X.-Z., and Lin, J.-R. (2022). Knowledge-informed semantic alignment and rule interpretation for automated compliance checking. Automation in Construction, 142:104524.