

BIM-based Hybrid Knowledge Graph based on DSL for Computational Design Rule Formalization

Huzhou Deng^{1,2} and Dian Zhuang^{1,2*}

¹School of Architecture and Design, Harbin Institute of Technology, Harbin, Heilongjiang 150001, China

²Key Laboratory of Cold Region Urban and Rural Human Settlement Environment Science and Technology, Ministry of Industry and Information Technology, Harbin, China
zhuangdian@hit.edu.cn

Abstract

The digitization of building codes is critical for Automated Compliance Checking (ACC) within the BIM paradigm, yet faces challenges such as unstructured text, semantic ambiguity, and deep logical nesting. Traditional hard-coding lacks flexibility, while reliance solely on Large Language Models (LLMs) is prone to hallucinations and offers limited BIM integration. This study proposes a multi-stage framework integrating Domain-Specific Languages (DSL), hybrid Knowledge Graphs, and LLMs to automate the formalization of computational design rules from textual codes. Using the Code for Design of Residential Buildings (GB 50096-2011) as a case study, we parse and classify regulatory provisions into four categories, Numerical, Logical, Topological, and Fuzzy and employ Prompt Engineering to guide LLMs in generating structured DSL JSON. A hybrid storage architecture (Neo4j, PostgreSQL, Drools) forms a queryable knowledge base that bridges natural language regulations and executable logic, enabling automated reasoning and direct interoperability with BIM models for compliance verification.

Keywords –

Domain Specific Language ; Large Language Model; Building Code Digitization; Knowledge Graph; Automated Compliance Checking.

1 Introduction

Building Information Modeling technology supports structured representation of building information across the full lifecycle through the Industry Foundation Classes (IFC) data format, transforming design deliverables into semantically rich, computer-readable data objects and establishing a foundation for interoperability. However, a core bottleneck in Automated Compliance Checking remains the digitization of building codes, which are predominantly available as unstructured natural language texts. Current

reliance on manual review is not only inefficient and prone to errors but also leads to inconsistent enforcement, ultimately affecting building safety and approval efficiency.

Research on ACC spans half a century, aiming to replace manual checking with automated solutions [1, 2]. BIM and IFC provide structured information models for ACC [3]. Recent studies have explored ontologies, semantic web technologies, and knowledge graphs. Anastasiya et al. [4] formalized compliance requirements via ontologies, though manual construction remains inefficient [5]. Peng et al. [6] used NLP to transform code clauses into formats readable by machines, and so did Lu Jia et al. [7] combined semantic web with IFC for improved checking.

Artificial intelligence technologies, including deep learning, NLP, and large language models (LLMs), are increasingly applied to rule extraction and semantic understanding. Li et al. [8] developed ontologies and deep learning models for building standard labeling. Wang et al. [9] proposed an NLP-based ACC system for spatial relationship review. Domain-Specific Languages (DSLs) offer tailored syntax and semantics for domain specific tasks [10], enabling domain experts to formalize problems without deep programming knowledge [11]. DSLs have been applied in parametric design [12], generative AI integration [13, 14], and construction robotics [15, 16].

Despite these advances, ACC faces key challenges. First, knowledge formalization remains labor-intensive and lacks scalability, struggling with implicit logic and regional variations. Second, a semantic gap persists in translating ambiguous natural language provisions into computable rules. Third, LLMs, while promising, are prone to hallucinations, affecting reliability. Finally, DSL application in ACC is still nascent and warrants further exploration.

To address these challenges, this paper proposes a framework based on BIM with multiple stages that integrates DSL, hybrid knowledge graphs, and LLMs to automate the formalization of design rules from building codes. The framework converts regulatory

texts into structured, executable knowledge through a dedicated DSL, reducing semantic ambiguity and interpretability risks. A hybrid storage architecture combining graph, relational, and rule based databases enables efficient management of numerical, logical, topological, and vague clauses, establishing a verifiable foundation for automated compliance checking integrated with BIM.

2 Parsing and Transformation Technology

While standard Large Language Models can perform general semantic parsing, their inherent stochasticity often leads to hallucinations and inconsistent extraction of complex building codes. The methodological innovation of this section lies in a twofold advancement: a multidimensional guided parsing strategy and a four tier DSL schema. First, in the parsing phase, we propose a “deconstruct then extract” strategy that uses prompt engineering to force the LLM to perform semantic role labeling identifying subject, constraint, and condition thereby minimizing linguistic ambiguity before data extraction. Second, in the transformation phase, we introduce an original four tier DSL schema that acts as a semantic safety buffer. By enforcing this constrained intermediary grammar, the framework decouples nondeterministic AI outputs from deterministic IFC data structures. This dual layered approach ensures that unstructured clauses are systematically transformed into a verifiable, machine readable format, providing logic stability and traceability that existing end to end LLM methods lack (Figure 1).

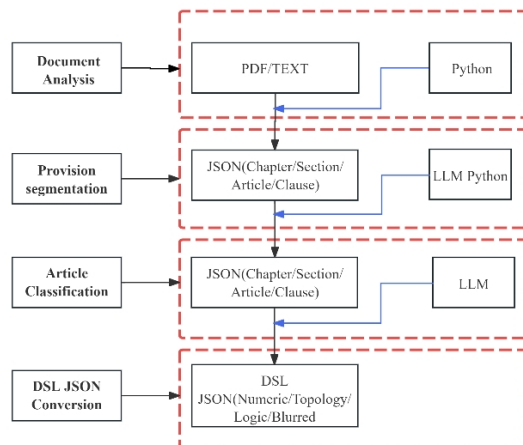


Figure 1: Technical Pathway Diagram

2.1 Document Preprocessing

To support the integration of design rules with BIM,

this paper designs a parsing workflow that transforms semi-structured code documents into machine readable hierarchical data objects aligned with BIM entities. The process involves three core steps. First, raw documents undergo text cleaning and standardization to remove formatting artifacts. Second, a parsing algorithm using regular expressions recognizes the hierarchical structure (e.g., chapter, section, article, clause) and reorganizes text into a nested tree. Concurrently, metadata such as code name and effective date are extracted. Finally, structured content and metadata are encapsulated into standardized JSON objects, providing a foundation for subsequent clause atomization and mapping to BIM component attributes.

2.2 Clause Segmentation

The clause segmentation process deconstructs complex regulatory sentences into minimal, machine executable constraint units enabling precise mapping to BIM components. The process includes three steps. First, syntactic parsing identifies logical boundaries to split long sentences. Second, LLMs perform semantic completion to resolve coreferences, ensuring each atomic clause contains complete “object, attribute, constraint” semantics relevant to BIM properties. Finally, a keyword algorithm detects modality terms (e.g., “shall”, “should”) and assigns mandatory or optional labels, producing a standardized set of candidate rules ready for formalization and execution integrated with BIM (Figure 2).

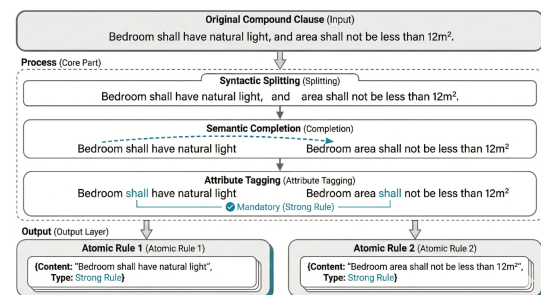


Figure 2: Detailed Processing Procedure for Article Segmentation

2.3 Multidimensional data classification

To implement differentiated digitalization strategies for different types of code clauses, this paper establishes a multi-dimensional data classification mechanism after clause atomization. Atomic clauses are precisely classified into four categories (Table 1):

1. Numerical Constraint Clauses: Identified by comparative operators, extracting specific quantitative indicators such as area and height.

2. Logical Constraint Clauses: Based on conditional sentence patterns, locking the dependency relationship between preconditions and subsequent requirements in building scenarios.
3. Topological Constraint Clauses: Focusing on spatial predicates such as "adjacent to," "connected to," and "contained within," defining geometric positional relationships between components.
4. Vague Constraint Clauses: Targeting qualitative descriptions like "should," "reasonable layout," and "aesthetic," identifying guiding principles that are difficult to quantify directly.

Type	Keywords	Process	Mandatory
Numerical	$\geq, \leq$, greater than, not less than	DSL	True
Logical	When... shall; If... then...	DSL	False
Topological	connected to, near	DSL	False
Blurred	Reasonable,s hould	DSL	False

2.4 DSL Semantic Transformation Based on LLM

To bridge the semantic gap between natural language regulations and executable logic within the BIM environment, we designed a Domain-Specific Language Schema adapted to the characteristics of building codes and aligned with common BIM data structures. This Schema includes fields such as target entity, check attribute, operator, and priority. Leveraging the semantic understanding capabilities of LLMs, we designed specific prompts to guide the model in transforming atomic natural language clauses into structured DSL JSON format. This process includes a "manual verification" step to ensure the accuracy and BIM integrability of the generated formal rules.

2.4.1 DSL Transformation for Logical Clauses

Logical clauses, characterized by conditional patterns such as "When... shall..." or "If... then...", define mandatory or recommended requirements under specific building scenarios. The core formalization challenge lies in accurately separating conditions from conclusions and establishing their logical mapping within a BIM aware framework.

To support this, the DSL Schema includes a

structured conditions array, where each condition object describes a premise using fields like parameter (often mapped to a BIM property), operator, and value. The conclusion is defined by target entity (linked to a BIM component type), check attribute, and operator. For instance, the clause "When a bedroom is located underground, a ventilation shaft shall be provided" is transformed into a rule where the condition checks location == "underground" and the conclusion targets the bedroom entity for ventilation facility provision."

This transformation leverages LLMs for semantic parsing and is followed by manual verification to ensure logical accuracy and alignment with BIM semantics. The output provides unambiguous if then rules ready for execution against BIM model data.

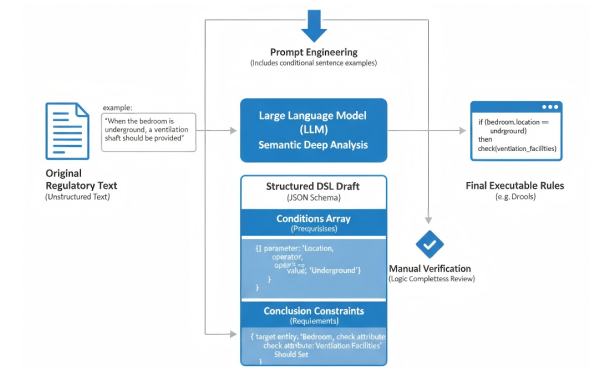


Figure 3: DSL Transformation for Logical Clauses

2.4.2 DSL Transformation for Numerical Clauses

The core feature of numerical clauses is the quantitative constraint on attributes (e.g., area, height, distance) of building spaces or components through specific numbers and comparison operators. To transform them into machine executable rules, this method designs a targeted DSL Schema and prompt engineering strategy.

The transformation is achieved using LLMs. By constructing prompts containing clear instructions, output format examples, and domain knowledge, the LLM is guided to accurately parse and map clauses like "bedroom area shall not be less than 12 square meters" to corresponding fields in the Schema. For instance, the LLM needs to correctly identify "not less than" as the operator ">=" and extract the pure number "12" and unit "square meters." A manual verification step is set in this stage. Verification focuses include: whether the value and unit correspond accurately; whether prerequisite conditions in conditional statements are fully captured and transformed into the conditions field; and whether the priority field for recommendatory clauses is

correctly marked as lower priority (Figure 4).

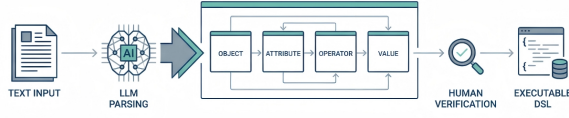


Figure 4: DSL Transformation for Numerical Clauses

2.4.3 DSL Transformation for Topological Clauses

Topological clauses regulate the relative position and connection relationships between building spaces or components, such as "adjacent to," "connected to," "contained within." These clauses are crucial for ensuring functional flow and spatial rationality but are highly challenging to formalize due to complex spatial semantics and geometric relationships. To achieve their transformation into machine executable rules, this paper specially designs an extended DSL Schema.

Addressing the characteristics of topological constraints, fields such as topology type, source entity, spatial relation operator, and target entity are added to the basic DSL Schema. A standardized set of topological operators is predefined in the Schema, including "adjacent to," "contains," "connected to," and "separate from." Secondly, the semantic reasoning capability of LLMs is utilized to accurately extract spatial features from clauses. Through prompt engineering, the model is guided to identify keywords like "immediately adjacent to," "abutting," "accessed via..." and map them to normalized DSL operators.

To ensure the geometric checkability of generated rules, strict manual verification is required after transformation. Verification focuses include: whether the relation predicate accurately reflects the original clause intent(Figure 5).

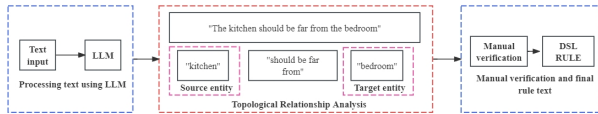


Figure 5: DSL Transformation for Topological Clauses

3 Database Construction

While existing ACC systems typically store rules in a single database format, this study proposes a hybrid storage architecture that matches storage paradigms to clause types a design choice that directly addresses the heterogeneous nature of building codes. The key innovations are: (1) a “ metadata rule body ”

separation pattern that decouples code version management from rule logic; (2) clause type specific storage strategies relational for numerical rules requiring efficient quantitative queries, rule engine for logical rules needing condition triggered reasoning, and graph database for topological rules requiring spatial relationship traversal; and (3) a unified knowledge graph that captures cross clause dependencies , enabling traceability and impact analysis beyond simple compliance checking. The database construction in this paper takes the structured DSL output from Chapter 2 as input, directly addressing the need for machine executable code clauses within a BIM supported environment. Based on the structured JSON data after DSL transformation, we designed a hybrid database architecture and a code knowledge graph to organize and manage rule knowledge in a manner aligned with BIM data structures and query requirements (Figure 6).

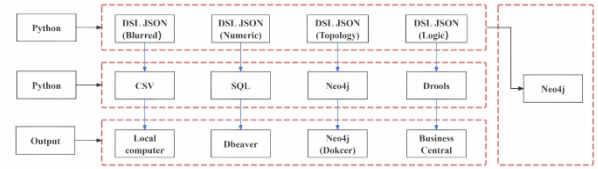


Figure 6: Database construction process

3.1 Database Structure

To achieve precise traceability and version management of code clauses, the database table structure adopts a "metadata rule body" separation design pattern.

1. Metadata Layer: Establish a specification index table to store global attributes of codes, including code name, version number, applicable region, and effective time. This design supports the system in quickly filtering applicable code sets based on project attributes.
2. Rule Body Layer: The core compiled rules table stores atomized rules. Field designs include: clause id, original text, target entity, dsl content, and, most importantly, executable code.

3.1.1 Database Construction for Numerical Clauses

For the numerical DSL JSON, this study employs Python scripts to import clauses into a PostgreSQL database. A relational hybrid storage architecture is implemented, organizing data into two BIM integrated dimensions. The structured dimension maps check targets, operators, and limit values to dedicated SQL columns, directly corresponding to BIM component attributes for efficient evaluation. The semistructured

dimension uses PostgreSQL's JSONB type to store applicability conditions, which are often linked to BIM project parameters. A GIN index is applied to the JSONB field to enable fast, contextaware rule retrieval across extensive clause sets, ensuring performant interoperability with BIM model queries.

3.1.2 Database Construction for Logical Clauses

For logical clauses, the system converts them into Drools rule language for storage, enabling checks to be triggered only when relevant BIM objects satisfy specific preconditions, thereby optimizing computation within the BIM context. A Python based translator parses the condition triggers and check logic from the DSL JSON, mapping them to Drools' MVEL language. At the data level, a dynamic Design Entity fact object is used, storing variable component attributes in an attributes Map to flexibly accommodate diverse BIM component schemas.

The generated drl files employ a “violation pattern matching” strategy, transforming code requirements into negative capture patterns. When a violating BIM component is matched, the system instantiates a Validation Result object, automatically injecting traceable metadata. This enables millisecond-level logical reasoning with full compliance traceability. Finally, the rule files are deployed to Business Central and exposed as a remote service for BIM integrated compliance checking (Figure 7).

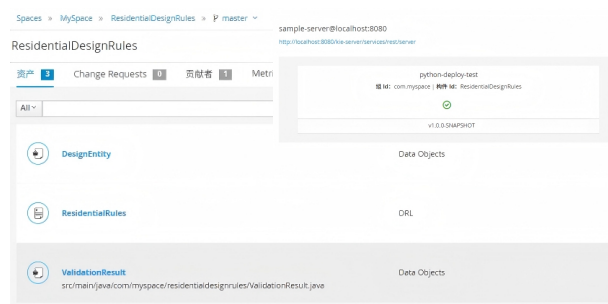


Figure 7: Related functional files and deployment on remote servers

3.1.3 Database Construction for Topological Clauses

For complex topological rules involving spatial adjacency, vertical stacking, and circulation connectivity, this study utilizes Neo4j to construct a code meta-knowledge graph.

Storage Architecture and Meta graph Construction: Based on the DSL JSON structure, the graph adopts a “Entity Type, Constraint Edge, Entity Type” triple pattern for storage.

1. Semantic Node Abstraction: Defines code objects

like "bedroom" and "exhaust duct" as Entity Type nodes, establishing mapping anchors to the IFC entity classification standard via the code field.

2. Graphical Representation of Constraint Logic: Stores clause logic as relationship edges with properties. Core properties include relation type and constraint level. This design makes each edge an independent, traversable logical unit.

3.1.4 Database for Vague Clauses

For the numerous subjective clauses in codes that cannot be quantitatively calculated, such as "reasonable layout" and "harmonious with the environment," this study constructs a lightweight knowledge base based on local CSV storage and semantic retrieval.

3.2 Knowledge Graph Database Construction

To manage complex relationships between clauses (Figure 8), such as parameter dependencies and potential conflicts, we constructed a Neo4j knowledge graph encompassing all clauses.

Node Design defines four core node types to structurally store code knowledge: Specification, Clause, Entity, Parameter.

Relationship Design defines directed connections between nodes to express logic and dependencies.

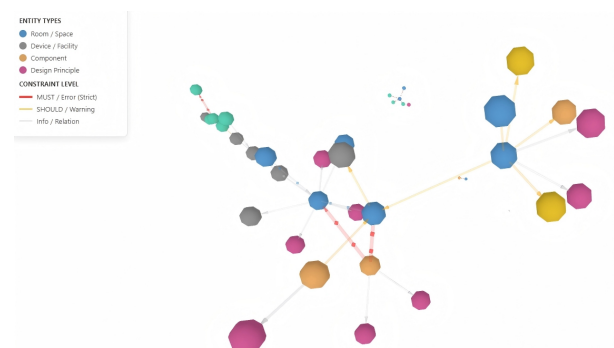


Figure 8: Dependency Relationships Between Clauses

4 Case Study

To validate the effectiveness of the proposed method, we selected the Residential Design Code GB 50096-2011 as the experimental subject, performing a full-process digitalization of its clauses.

4.1 Data Processing Results

To ensure reproducibility, the rule extraction and formalization process was standardized using the Qwen-Max model with fixed generation parameters:

temperature = 0.1, top_p = 0.95, and max_tokens = 2048. Consistent prompt templates were used for Numerical, Logical, Topological, and Fuzzy clauses. This systematic approach enables replication following the DSL specifications and prompt logic detailed in this study. After preprocessing, 338 atomic rule clauses were extracted (Table 2).

Table 2: Clause Segmentation Results

Type	Quantity	Percentage
Numerical	123	36.4%
Logical	159	47.0%
Topological	45	13.3%
Blurred	11	3.3%
Total	338	100%

4.2 DSL JSON Transformation and Code Base Construction

After completing the document preprocessing and clause atomization of *GB 50096-2011*, this study obtained 338 semantically independent and structurally clear atomic clauses. To achieve their transformation from natural language to machine executable rules, this stage focused on two core tasks:

Phase 1: Automated DSL JSON Generation. The core of this study lies in the "LLM+DSL" semantic transformation paradigm. First, we designed refined DSL Schemas for the three types of "clear clauses" (numerical, logical, topological), specifying the data type and semantic boundary of each field (as described in Section 2.4). Subsequently, for each atomic clause, we constructed highly customized prompts containing the following key elements:

1. **Role Definition:** Explicitly instructing the LLM to act as a "building code digitalization expert."
2. **Task Instruction:** Clearly requiring the LLM to transform the input natural language clause into a JSON object strictly adhering to the specified Schema.
3. **Output Format Example:** Providing correct transformation examples for the same type of clause to guide the model in learning the mapping pattern.
4. **Domain Knowledge Constraints:** Embedding lists of domain-specific entities and operators to constrain the model's output space and reduce hallucination risk.

By calling the LLM API, the system automatically transformed the 338 atomic clauses into structured DSL JSON objects one by one. To ensure transformation quality, all automatically generated JSON objects underwent a manual verification step.

Phase 2: Hybrid Database Construction and Storage. This study abandons the generic storage approach of a single database, adopting instead a "allocate on demand, each performing its own function" hybrid storage architecture to maximize the processing efficiency for different rule types.

1. **Numerical Clauses:** Verified numerical DSL JSON was batch-parsed by Python scripts and imported into the PostgreSQL relational database. The scripts mapped key fields like check attribute, operator, and value to standard SQL columns, while storing complex applicability conditions in the applicability conditions field in JSONB format. A GIN index was established to support efficient contextual retrieval.
2. **Logical Clauses:** Logical DSL JSON was fed into a specially developed rule translator. This translator accurately compiled the condition and conclusion logic from the JSON into drl files recognizable by the Drools rule engine. These rule files were then deployed to the Drools Business Central platform and encapsulated as remote service interfaces for on demand calling by the compliance checking system, enabling complex condition triggered reasoning.
3. **Topological Clauses:** Topological DSL JSON was used to construct the code meta knowledge graph in the Neo4j graph database. Scripts created nodes for source entity and target entity and stored information like spatial operator and constraint level as properties of relationship edges (Figure 9).

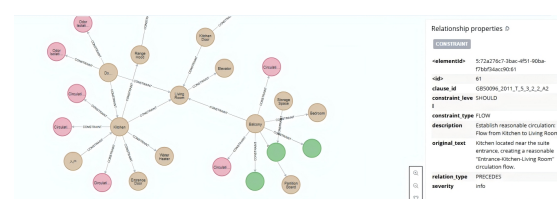


Figure 9: Relationships Among Topological Clauses in Neo4j (Partial GB 50096-2011 Examples)

4. **Vague Clauses:** For vague clauses like "reasonable layout" that cannot be quantified, their original text, associated chapter, and related entity information were structured and stored in a local CSV file.

4.3 Error Analysis and Accuracy Testing

During automated DSL JSON generation, the LLM occasionally produced incorrect outputs despite prompt engineering. Manual verification identified seven failure patterns, accounting for 22 erroneous outputs out of 338 atomic clauses. These included: misinterpretation of comparative operators (4), where “not less than” became “>” instead of “>=”; incomplete condition capture in logical clauses (3), omitting premises when conditions appeared mid sentence; entity role reversal in topological clauses (3), swapping source and target; unit inconsistency (2), extracting values without units or confusing millimeters with meters; target entity mismatch (3), assigning incorrect BIM entity types; missing optional/recommendation flag (2), failing to map “should” to the priority field; and fuzzy clause not recognized (5), misclassifying qualitative clauses as logical or numerical. We built a test set containing 100 BIM model fragments as test cases to ensure that the provisions in the database can be retrieved.

For numerical clauses, we created multiple test cases, stored them in a temporary SQL table, and used Python scripts to call the SQL database for compliance checking. The checking process first matched the check objects in the cases with the clauses, performing compliance checks only if the preconditions of the clauses (e.g., building completion year, location, object type) were satisfied (Figure 10).

A/Z check_item	A/Z actual_value	A/Z status	A/Z message	A/Z original_text
Usable Area	25.0	PASS	Satisfies >=10.0m ²	Living room (hall) usable area shall not be less than 10.0m ²
Net Interior Depth	2.8	FAIL	Actual 2.8m does not satisfy >	Net interior depth of living room (hall) shall not be less than 2.8m
Net Interior Depth	2.8	FAIL	Actual 2.8m does not satisfy >	Net interior depth of living room (hall) shall not be less than 2.8m
Net Interior Height	2.7	PASS	Satisfies >=2.4m	The overall net interior height of bedrooms and living rooms shall not be less than 2.4m
Usable Area	12.0	PASS	Satisfies >=9.0m ²	For a double bedroom (two people), usable area shall not be less than 9.0m ²
Net Interior Width	2.6	PASS	Satisfies >=2.5m	For a double bedroom (two people), net interior width shall not be less than 2.5m
Net Interior Height	2.5	PASS	Satisfies >=2.4m	The overall net interior height of bedrooms and living rooms shall not be less than 2.4m
Daylight Factor	0.008	FAIL	Actual 0.008% does not satisfy	Daylight factor for bedrooms, living rooms, and kitchen shall not be less than 0.008%
Stove Width	0.45	FAIL	Actual 0.45m does not satisfy	Stove width shall not be less than 0.5m
Net Height	1.0	FAIL	Actual 1.0m does not satisfy >	For residential balconies of six stories and below, net height shall not be less than 1.0m
Net Height	1.0	FAIL	Actual 1.0m does not satisfy >	For residential balconies of seven stories and above, net height shall not be less than 1.0m

Figure 10: Partial Test Cases (Numerical)

For logical clauses, we similarly stored cases in a temporary SQL table, then used Python scripts to call the deployed Drools remote server for compliance checking. We also inserted test cases for clauses into Business Central to verify whether our drl code rules were invoked (Figure 11).

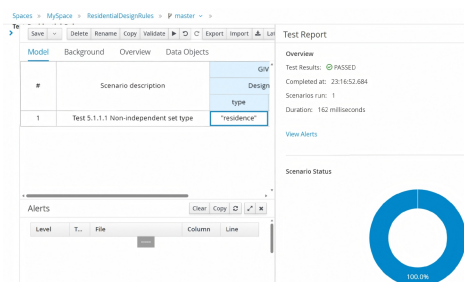


Figure 11: Checking if drl Files are Invoked in

Business Central

For topological clauses, we stored cases in a temporary SQL table and used Python scripts to call the deployed Neo4j for compliance checking.

4.4 Application Value of the Knowledge Graph

The constructed knowledge graph, designed to support BIM based analysis, contains over 300 nodes and 500 relationships. Graph analysis reveals strong couplings between design entities such as “Bedroom” and performance parameters including “Daylight Factor” and “Ventilation Opening Area.” In a typical BIM aided design scenario, if a designer modifies a window size, the system can utilize the graph’s path-search algorithms to not only evaluate compliance with daylighting standards but also proactively surface related clauses involving “façade aesthetics” or “cost indicators.” This demonstrates the potential of the graph to enable multidimensional, rule aware correlation analysis within the BIM workflow.

5 Scalability and Limitations

The proposed framework demonstrates strong generalizability across building codes from different regions and domains. The DSL Schema decouples logical structure from natural language, enabling systematic migration to other regulatory documents such as the International Building Code (IBC), European construction standards, and domain specific fire safety regulations without reconstructing the core architecture. Since the four category classification (numerical, logical, topological, fuzzy) covers the vast majority of regulatory provisions encountered in typical building codes, core prompt templates require no modification when adapting to new codes. Adaptation follows a calibration workflow: prepare 20 – 30 annotated clauses as few shot examples, feed them into the LLM to learn new terminology and patterns, validate output quality, then process the full code with manual verification only for ambiguous cases. This calibration takes 2-3 hours per new code, demonstrating the framework’s automation efficiency for large scale code digitization.

Compared to existing LLM, rule based, or ontology approaches which suffer from hallucinations, manual effort, or limited scalability our hybrid framework integrates LLM semantic understanding with DSL precision. This enables verifiable rule extraction and provides a modular architecture for rapid reconfiguration across diverse building codes.

Nevertheless, the current system has limitations. It relies on manual verification to ensure semantic

accuracy, preventing full end to end automation. This manual step, while necessary to eliminate LLM hallucinations and guarantee correct DSL generation, represents a bottleneck in achieving a fully automated pipeline for rule formalization.

6 Conclusion

This paper proposes a complete technical pathway for transforming building codes into a structured digital knowledge base within a BIM oriented framework. Using GB 50096-2011 as a case study, the resulting hybrid database and knowledge graph form a reasoning capable rule system that can serve as the core engine for BIM based automated compliance checking. The system supports automated verification of numerical, logical, and topological clauses and provides auxiliary decision support via a vague clause knowledge base.

Future work will focus on enhancing BIM integration through multidimensional IFC parsing for richer semantic and geometric data, automated version comparison, DSL schema refinement, and automating manual verification in the DSL pipeline via three strategies: a retrieval augmented generation mechanism with few shot examples, a rule based post-processor for recurring errors, and a self consistency validation loop using bidirectional translation. These enhancements will reduce manual effort while maintaining accuracy, strengthening end to end automation and deployability, alongside a scalable compliance-checking application on Dify.

7 Acknowledgments

This work was funded by the National Natural Science Foundation of China (Grant number 52408015), National Key Research and Development Program of China (Grant number 2024YFC3808901).

References

- [1] R. Amor and J. Dimyadi. The promise of automated compliance checking. *Developments in the Built Environment*, 5:100039, 2021.
- [2] Q. Ren, D. Zhang, M. Li, S. Chen, D. Tian, H. Li, and L. Liu. Automatic quality compliance checking in concrete dam construction: Integrating rule syntax parsing and semantic distance. *Advanced Engineering Informatics*, 60:102409, 2024.
- [3] S. Macit İlal and H. M. Günaydın. Computer representation of building codes for automated compliance checking. *Automation in Construction*, 82:43–58, 2017.
- [4] A. Yurchyshyna and A. Zarli. An ontology-based approach for formalisation and semantic organisation of conformance requirements in construction. *Automation in Construction*, 18(8):1084–1098, 2009.
- [5] P. Pauwels, S. Zhang, and Y.-C. Lee. Semantic web technologies in AEC industry: A literature overview. *Automation in Construction*, 73:145–165, 2017.
- [6] J. Peng and X. Liu. Automated code compliance checking research based on BIM and knowledge graph. *Scientific Reports*, 13(1), 2023.
- [7] L. Jia, M. Chen, C. Chen, and Y. Jin. A semantic web and IFC-based framework for automated BIM compliance checking. *Buildings*, 15(15):2633, 2025.
- [8] S. Li, J. Wang, and Z. Xu. Automated compliance checking for BIM models based on Chinese-NLP and knowledge graph: an integrative conceptual framework. *Engineering, Construction and Architectural Management*, 32(6):3832–3856, 2024.
- [9] Y. Wang, Y. Liu, H. Cai, J. Wang, and X. Zhou. An automated fire code compliance checking jointly using building information models and natural language processing. *Fire*, 6(9):358, 2023.
- [10] F. Ciccozzi, L. Addazi, S. A. Asadollah, B. Lisper, A. N. Masud, and S. Mubeen. A comprehensive exploration of languages for parallel computing. *ACM Computing Surveys*, 55(2):1–39, 2022.
- [11] E. D. Sozzo, D. Conficconi, A. Zeni, M. Salaris, D. Sciuto, and M. D. Santambrogio. Pushing the level of abstraction of digital system design: A survey on how to program FPGAs. *ACM Computing Surveys*, 55(5):1–48, 2022.
- [12] S. Wu, W.-Z. Lu, C. Qin, B. Wen, D. Wu, and Y. Xiang. A goal-oriented planning approach for two-dimensional cutting components in architectural design: Coupling BIM and Parametric design (PD). *Journal of Building Engineering*, 76:107156, 2023.
- [13] D. Jun and W. Zi Jia. A novel architectural design paradigm: Collaborative construction with design thinking semantic networks and large language models. *Preprints*, 2025051238, 2025.
- [14] J. Ko, J. Ajibefun, and W. Yan. Experiments on generative AI-powered parametric modeling and BIM for architectural design. *arXiv preprint arXiv:2308.00227*, 2023.
- [15] S. Leder and A. Menges. Architectural design in collective robotic construction. *Automation in Construction*, 156:105082, 2023.
- [16] Y. Liu, M. Habibnezhad, and H. Jebelli. Brainwave-driven human-robot collaboration in construction. *Automation in Construction*, 124:103556, 2021.