From Automatic to Autonomous: a Large Language Model-driven Approach for Generic Building Compliance Checking

Huaquan Ying and Rafael Sacks

November 5, 2024

From Automatic to Autonomous: A Large Language Model-driven Approach for Generic Building Compliance Checking

Huaquan Ying, enochying@campus.technion.ac.il
Faculty of Civil and Environmental Engineering, Technion-Israel Institute of Technology, Israel

Rafael Sacks, cvsacks@technion.ac.il
Faculty of Civil and Environmental Engineering, Technion-Israel Institute of Technology, Israel

Abstract
Despite extensive research and development in automating compliance checking (ACC) of building designs, a generic, scalable, and automated system remains elusive. Current ACC systems are limited to specific domains and rule types. The main challenges lie in the vast number of design requirements and the wide diversity of object concepts, attributes, constraints, and relations. This complexity demands an intelligent, autonomous system capable of independent planning, decision-making, and task execution to adapt to new situations, rather than traditional pre-programmed automation paradigms. Inspired by the agency capabilities of Large Language Models (LLMs) in solving complex tasks, this study explores their application in building compliance checking. The goal is to develop an LLM agent that can understand design requirements, plan the checking process, retrieve relevant BIM data, and execute checks autonomously. A proof-of-concept prototype was developed and an autonomous rule-checking experiment conducted with it has successfully demonstrated the potential of the LLM-driven approach.

Keywords: Building Information Modelling, Automatic Compliance Checking, Autonomous Compliance Checking, Large Language Model, Artificial Intelligence Agent

1 Introduction

In the Architectural, Construction, and Engineering (AEC) industry, building designs must meet various requirements stipulated in codes, standards, and project-specific specifications. Ensuring compliance with these requirements is essential for building functionality, safety, and client satisfaction. Traditionally, this process involves manual checking using two-dimensional drawings and documents. Automatic compliance checking (ACC) using Building Information Modelling (BIM) could potentially improve accuracy and efficiency significantly. However, despite decades of research and development, a generic, scalable, and fully automated ACC system remains elusive (Amor & Dimyadi 2021).

A compliance checking process contains two main steps: (1) interpreting design requirements from texts into computational rules and (2) preparing high-quality and accessible BIM data for rule execution (Eastman et al 2009). Both steps present significant automation challenges.

With respect to rule interpretation, design requirements are expressed in numerous forms with great diversity, involving various types of objects, attributes, constraints, and relationships. The logic of the rules is embedded in multimodal formats including text, tables, and figures (Solihin & Eastman 2015; Amor & Dimyadi 2021). This diversity poses substantial challenges in developing a generic and fully automated rule interpretation approach. Currently, most commercial ACC systems, such as Solibri (2024), rely on manual rule interpretation by rule experts and programmers, resulting in hard-coded systems that support limited rule types and lack scalability for new rules. Despite extensive research on automating rule interpretation,

current tools remain limited to certain types of textual rules or design requirements in specific domains, without scalability, due to the static and pre-programmed automation paradigms.

Challenges in BIM data preparation arise from frequent quality issues, such as data inaccuracies and missing information for rule checking. These issues are mostly addressed manually in current practice, making the process laborious, time-consuming, and error-prone. While various semantic enrichment methods have been proposed for BIM data correction and implicit BIM data inference (Bloch 2022), they lack any automation for identifying incorrect and missing data and for selecting and developing appropriate enrichment tools.

Thus it is apparent that the complex and diverse nature of building compliance checking presents significant challenges in developing fully automated and generic solutions using traditional automation methods that rely on static and predetermined logic. Instead, an intelligent, autonomous system is needed—one that can independently plan, make decisions, utilize tools to execute tasks, and adapt to new situations.

Recent advances in AI, particularly with Large Language Models (LLMs) such as GPT-4 (OpenAI 2024) and Llama 3 (Meta 2024), offer significant potential for developing a generic compliance checking system. LLMs exhibit broad reasoning capabilities and encompass vast human knowledge, including insights into building design and BIM, due to their extensive training data. They excel at understanding and generating human-like text. Recent advances have made LLMs multimodal, enabling them to process not only text but also images. Furthermore, LLMs have been equipped with the ability to utilize tools and possess coding capabilities for developing new tools. Lastly, LLM-based agent techniques synthesize these capabilities to autonomously solve complex tasks (Wang et al 2024). These features make LLMs highly promising for addressing the significant challenges in automatically interpreting rules and preparing BIM data.

In this paper, we propose an autonomous approach using LLM agents, designed to create a generic, scalable, and fully automatic building compliance checking system. This approach can handle requirements in various formats, including text, tables, and figures, thanks to the multimodal capabilities of LLMs. It eliminates the need for pre-programmed rule checking entirely. Instead, for each target requirement, it autonomously plans and executes the rule-checking process, making it adaptable to various types of rules with varying complexity. The approach includes two complementary modules, both of which use natural language: one for querying design requirements to prepare for design checking, and another for improving building design based on rule-checking results.

2 Background

2.1 ACC and its current challenges

According to Eastman et al. (2009), a BIM-based ACC process generally consists of four stages: (1) rule interpretation, which translates checking rules from normative provisions into a computable format; (2) BIM data preparation, which ensures high-quality and complete design data for rule execution; (3) rule checking execution, which combines computable rules and required data for checking; and (4) result reporting, which presents results in a user-friendly manner. Despite extensive research and development efforts, significant technical challenges persist across all four stages, particularly in the first two.

Automated rule interpretation is challenged by the vast number of potential rules with diverse rule logic and complexities (Amor & Dimyadi 2021). Various efforts have aimed to improve automation, scalability, and versatility. SQL-like and visual programming-based rule languages (Lee et al 2015; Preidel & Borrmann 2016) have been developed to allow users to define customized checking rules without direct programming, providing a semi-automatic but flexible solution. However, the ability to support customized rules is restricted by the capabilities of these rule languages.

Semantic and ontology-based approaches (Jiang et al 2022; Pauwels et al 2024) aim to provide specialized BIM data representation formats to facilitate concept mapping between narrative rules and BIM data. This simplifies BIM data queries and semantic reasoning for rule checking (Solihin et al 2020). However, constructing comprehensive and useful ontologies

requires extensive domain knowledge and effort, limiting their scalability. These approaches are often confined to specific domains, such as fire safety (Fitkau & Hartmann 2024) or underground utilities (Xu & Cai 2020), and typically support only simple quantity rules.

Traditional and deep learning-based NLP technologies have been applied to parse design requirements automatically (Zhang & El-Gohary 2016; Wang & El-Gohary 2023). However, these efforts focus on recognizing entities, attributes, constraints, and relations from natural language requirements and mapping them to BIM models. They fall short of achieving automatic checking systems and struggle with rules that have complex logic, performance-related rules, cross-referenced rules, and rules represented in tables and figures.

In the BIM data preparation stage, issues such as missing or inaccurate data are common (Amor & Dimyadi 2021; Ying & Lee 2021). These issues are often resolved manually through multiple iterations, making the process laborious, time-consuming, and costly. Recent advances in semantic enrichment methods using AI tools offer promising solutions for correcting and inferring BIM data (Wang et al 2024). However, automating the recognition of quality issues and selecting the appropriate enrichment tools during a rule checking process remain unexplored.

Current rule checking execution methods are straightforward, launching the checking system once rules are translated and BIM data are prepared. This static approach does not allow for human intervention, making it unsuitable for a large amount of ambiguous, subjective, or qualitative rules (Zhang et al 2023). A human-in-the-loop mode is needed to incorporate domain knowledge and local preferences when executing these rules.

Result reporting in current ACC systems typically indicates a simple "pass" or "fail" with references to the original textual rules (Eastman et al 2009). While sufficient for simple rules, this approach lacks the detail necessary for improving designs for complex rules. Building permitting processes often require experts to manually investigate and report the causes of failures, which is time-consuming and error-prone.

In summary, achieving a generic and fully automatic building design checking system remains elusive. Current efforts are limited in scalability and adaptability to new rules due to their prep-programmed automation nature. They also struggle to handle complex, ambiguous, and multimodal rules and cannot address BIM data quality issues effectively.

2.2 LLMs in the AEC industry

The development of LLMs, particularly Generative Pre-trained Transformer (GPT) models, has revolutionized natural language understanding and generation. LLM capabilities have evolved rapidly, especially in tool utilization. They have developed an agentic technique, enabling them to solve a wide range of engineering tasks beyond NLP across various industries.

LLMs are now being explored for engineering tasks within the AEC industry. For instance, Zheng & Fischer (2023) developed a dynamic prompt-based approach using GPT models to support natural language-based BIM data queries. Saka et al (2023) demonstrated LLMs' remarkable few-shot learning capability as a valuable tool for selecting suitable building materials. Jiang et al (2024) developed an LLM-based platform called EPlus-LLM, which can automatically generate an EnergyPlus building energy model from a natural language description of a building with high accuracy, significantly reducing the effort required for energy modeling. Similarly, Jang et al (2024) proposed an LLM-driven system that enables automated detailing of exterior walls using natural language with promising accuracy.

Despite these very promising applications, the potential of LLMs for ACC remains unexplored. How LLMs can address the limitations of current ACC systems, as well as the specific challenges and limitations of LLMs in this context, remain unclear. Inspired by the capabilities of LLMs—such as embedded domain knowledge, generic reasoning, human language understanding and generation, and tool utilization and development—this study proposes an LLM approach for autonomous design compliance checking. The approach seeks to resolve the key issues we have identified, including the scalability and generalization limitations of current ACC systems.

3 LLM-based approach for autonomous checking

3.1 Overview

To integrate design requirement checking into design processes seamlessly, the proposed approach includes not only the core requirement checking component but also two complementary modules: design requirement querying and building design revision. Both modules also use a LLM. Figure 1 illustrates the workflow of the approach.

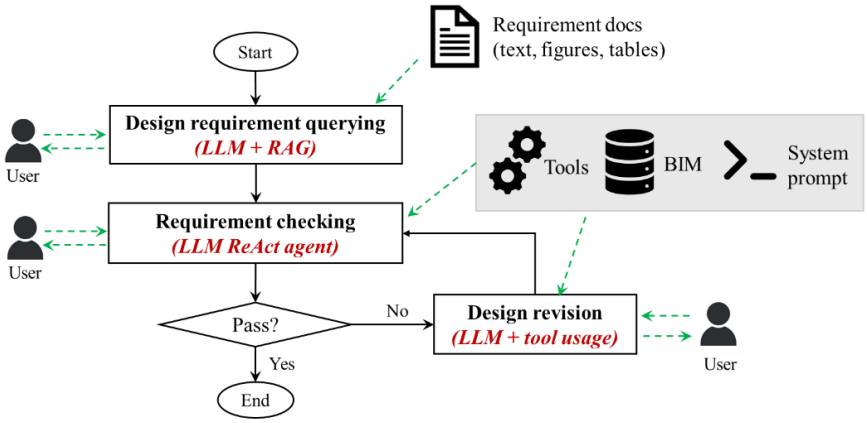


Figure 1. Workflow of the approach

The design requirement querying module enables designers to quickly obtain specific requirements using natural language, eliminating the need to search through extensive documents manually. This is very efficient because it leverages the Retrieval-Augmented Generation (RAG) technique (Lewis et al 2020), which allows LLMs to improve the accuracy of their answers and to respond using content not seen during training.

Once designers have obtained the specific design requirements, they can launch the autonomous requirement checking module to verify whether a BIM model meets these requirements. This module functions as an LLM agent, autonomously planning and executing the traditional BIM-based rule checking process, including rule interpretation, BIM data querying, checking, and reporting results.

The design revision module can manipulate and improve the BIM design directly according to the requirement checking results or instructions from designers. This module leverages the tool-using capabilities of LLMs to execute these tasks.

3.2 Design requirement query

LLMs often struggle with domain-specific tasks and queries beyond their training data. Even when trained on relevant data, LLMs may still produce inaccurate answers due to a lack of precise context for specific queries. The RAG technique addresses this issue by first retrieving relevant context from an external knowledge base and then sending it, along with the queries, to LLMs to reduce the risk of generating incorrect content (Gao et al 2022).

Therefore, we propose a RAG-based design requirement query method, consisting of two steps, illustrated as `Algorithm 1-1` and `Algorithm 1-2` in Figure 2. In the first step, the target design requirement document is chunked and embedded to enable relevant content retrieval. Since design requirement documents often contain text, tables, and figures, these different modalities are treated separately. The long text is chunked into smaller pieces (`Algorithm 1-1`, line 1), and each text chunk is embedded into a fixed-length vector using a large embedding model (LEM) (`Algorithm 1-1`, line 2). Tables and figures are preprocessed by generating textual descriptions and captions using LLMs, which are then embedded (`Algorithm 1-1`, lines 3-6). This document embedding process is a one-time task, and the stored embeddings can be reused as needed.

In the second step, the RAG process that uses the embeddings is implemented. First, the user's query is embedded using the same LEM (`Algorithm 1-2`, line 1). An embedding-based semantic

search then retrieves relevant content from the design requirement document (Algorithm 1-2, lines 2-4). This is achieved by computing the cosine similarity between the query and each document chunk, subsequently returning the top five highest-scored text chunks, table summaries, and image captions. These retrieved pieces are aggregated to form the query's context (Algorithm 1-2, line 5). Finally, the LLM uses this context to generate the desired design requirements (Algorithm 1-2, line 6).

Algorithm 1-1: Building code document embedding

Input: Digital document of building code: doc
Large language model: LLM
Large embedding model: LEM

Output: Text chunks and embeddings: $text\_embed$
Tables, summaries, and embeddings: tab_sum_embed
Figures, captions, and embeddings: fig_cap_embed

1: $textChunks, tables, images \leftarrow \text{PartitionDoc}(doc)$
2: $text\_embed \leftarrow \text{CreateEmbeddings}(textChunks, LEM)$
3: $tab\_sum \leftarrow \text{SummarizeTable}(tables, LLM)$
4: $tab\_sum\_embed \leftarrow \text{CreateEmbeddings}(tab\_sum, LEM)$
5: $fig\_sum \leftarrow \text{FigureCaption}(figures, LLM)$
6: $fig\_sum\_embed \leftarrow \text{CreateEmbeddings}(fig\_sum, LEM)$
7: **return** $text\_embed, tab\_sum\_embed, fig\_cap\_embed$

(a)

Algorithm 1-2: Embedding-based design requirement query

Input: Text chunks and their embeddings: $text\_embed$
Tables, summaries, and their embeddings: tab_sum_embed
Figures, captions, and their embeddings: fig_cap_embed
Large embedding model: LEM
Large language model: LLM
User query: q

Output: Design requirements: $reqs$

1: $q\_embed \leftarrow \text{CreateEmbeddings}(q, LEM)$
2: $text \leftarrow \text{EmbeddingSearch}(q\_embed, text\_embed)$
3: $tab\_sum \leftarrow \text{EmbeddingSearch}(q\_embed, tab\_sum\_embed)$
4: $fig\_cap \leftarrow \text{EmbeddingSearch}(q\_embed, fig\_cap\_embed)$
5: $relevant\_content \leftarrow \text{Aggregrate}(test, tab\_sum, fig\_cap)$
6: $reqs \leftarrow \text{GetAnswer}([relevant\_content, \text{query}], LLM)$
7: **return** $reqs$

(b)

Figure 2. Pseudocodes of the RAG approach for design requirement query

3.3 Requirement compliance checking

Once design requirements are obtained, the next step is to implement the compliance checking process. As explained earlier, a generic, scalable, and fully automated building design checking requires an autonomous solution that can independently plan and execute the checks for any given requirement, rather than relying on pre-programmed automation.

Here, we apply an LLM agent-based solution for autonomous design checking using the ReAct (Reason and Act) framework (Yao et al 2022). ReAct can significantly enhance the problem-solving capabilities of LLMs by generating reasoning traces and actionable steps in an interleaved manner. This dynamic approach enables the agent to solve complex problems by simultaneously proposing and executing concrete actions. The proposed ReAct agent has three key components:

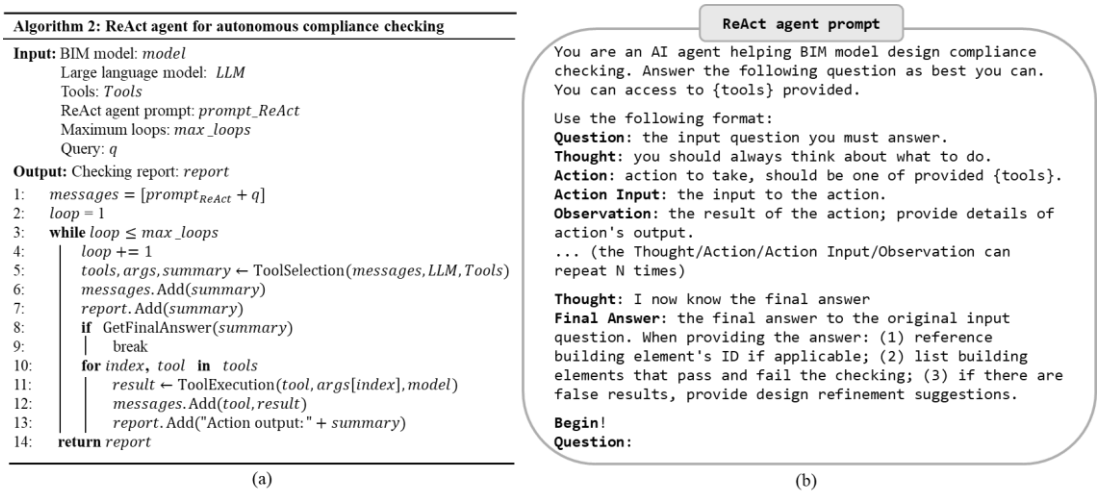
System prompt: The system prompt instructs the LLM to follow the reason-and-act paradigm for executing the rule checking process, as shown in Figure 3(b), through a loop consisting of Thought (think what to do), Action (determine specific actions), Action Input (identify action inputs), and Observation (analyze the action results). This loop repeats until the checking task is completed. Additionally, the system prompt includes instructions to ensure the final checking report is well-organized and provides design revision suggestions for any failed checks, as shown in the "Final Answer" section of Figure 3(b).

Tools: The agent can access built-in LLM tools such as file search and code interpreters, along with a wide range of third-party custom tools with APIs for various purposes (e.g., web search). In our context, we have developed additional tools for interacting with BIM models. These tools enable the LLM to query and process BIM data for rule checking.

Agent Memory: This stores the conversation history and state. Given that LLMs lack the ability to recall previous interactions, a dedicated memory system is implemented to collect and retain all conversations that occur throughout the rule-checking process.

Algorithm 2, shown in Figure 3(a), details the workflow of the constructed agent. First, an agent memory is initialized by combining the agent system prompt with user's query (i.e., the textual request for executing a specific design check) (line 1 in Algorithm 2). The memory is then frequently accessed and updated to facilitate the agent workflow (lines 3-13). In each iteration, the agent plans the next steps based on current state and available tools, then determines the actions (i.e., tools) to take and the corresponding input arguments (line 5). Next, it executes the actions and records the results into memory to update the state (lines 10-13). The loop continues until the agent determines that the task is completed (lines 8-9). To ensure

transparency, the entire rule checking process is documented in the final report, including every thought, action, input, output, and the final answer.



Algorithm 2: ReAct agent for autonomous compliance checking

Input: BIM model: *model*
Large language model: *LLM*
Tools: *Tools*
ReAct agent prompt: *prompt_ReAct*
Maximum loops: *max_loops*
Query: *q*

Output: Checking report: *report*

```
messages = [prompt_ReAct + q]
loop = 1
while loop ≤ max_loops
    loop += 1
    tools, args, summary ← ToolSelection(messages, LLM, Tools)
    messages.Add(summary)
    report.Add(summary)
    if GetFinalAnswer(summary)
        break
    for index, tool in tools
        result ← ToolExecution(tool, args[index], model)
        messages.Add(tool, result)
        report.Add("Action output:" + summary)
return report
```

(a)

ReAct agent prompt

```
You are an AI agent helping BIM model design compliance
checking. Answer the following question as best you can.
You can access to {tools} provided.

Use the following format:
Question: the input question you must answer.
Thought: you should always think about what to do.
Action: action to take, should be one of provided {tools}.
Action Input: the input to the action.
Observation: the result of the action; provide details of
action's output.
... (the Thought/Action/Action Input/Observation can
repeat N times)

Thought: I now know the final answer
Final Answer: the final answer to the original input
question. When providing the answer: (1) reference
building element's ID if applicable; (2) list building
elements that pass and fail the checking; (3) if there are
false results, provide design refinement suggestions.

Begin!
Question:
```

(b)

Figure 3. Pseudocode of the ReAct agent for design compliance checking (a) and the prompt designed for the ReAct agent (b).

3.4 BIM design revision

If a BIM model's elements fail to meet design requirements, an LLMs can edit the BIM model using failure information from the rule-checking report or new requests from designers. To enable this functionality, LLMs need to be equipped with relevant BIM model manipulation tools. Assuming a set of BIM manipulation tools has been provided to satisfy revision requests, the process for executing a single BIM design revision is straightforward. The revision query and a description of the available tools are sent to the LLM, which determines which tool to use, when to use it, and what input arguments are needed. The tools are then used to revise the BIM model, and the execution results are sent back to the LLM to prepare the final response. Designers can decide the next steps: confirm the edit or interact with the LLM to make further adjustments. This iterative process continues until the designer confirms that the model meets the design requirements.

4 Preliminary validation

4.1 Proof-of-concept prototyping

To validate the proposed approach, we developed a prototype called RevitCopilot as a Revit plugin, using C# and Python, as shown in Figure 4(a). The multi-modal GPT-4o (OpenAI 2024a) was selected as the LLM for all three modules. For Module 1, we used Unstructured (2024)'s package to partition the design document, and the text-embedding-3-small model (OpenAI 2024b) for text embedding. For Module 2, we developed two tools for Revit model queries to support the rule-checking agent: (1) extractBuildingElement_property, which retrieves all semantic properties of specified building element types (e.g., walls, floors) using the Revit API and outputs the data in JSON format; (2) BIMJSONQuery, which allows natural language queries of building elements' semantic properties stored in JSON format. For Module 3, we developed the adjustBuildingElementThickness tool, which changes the thickness of a building element in Revit given its Revit element ID and the new thickness.

4.2 Experiment

4.2.1 Experimental design

To demonstrate feasibility, we devised an experiment using a Revit model of a two-level duplex apartment, shown in Figure 4(a). This model includes 20 floor elements, four of which are cast directly on the ground, as depicted in Figure 4(b). For testing purposes, the family type name and the thicknesses of two of the elements, Slab 216507 and Slab 216552, were edited to values

different to their original values. Chapter 19 of the International Building Code (IBC) 2009, which focuses on concrete requirements, was selected as the design document for the experiment.

The experiment began with querying relevant design requirements based on user intent (Section 4.2.2). A specific design requirement was then selected and autonomously checked against the BIM model to identify design issues (Section 4.2.3). Finally, the identified incompliances were automatically corrected using natural language prompts (Section 4.2.4).

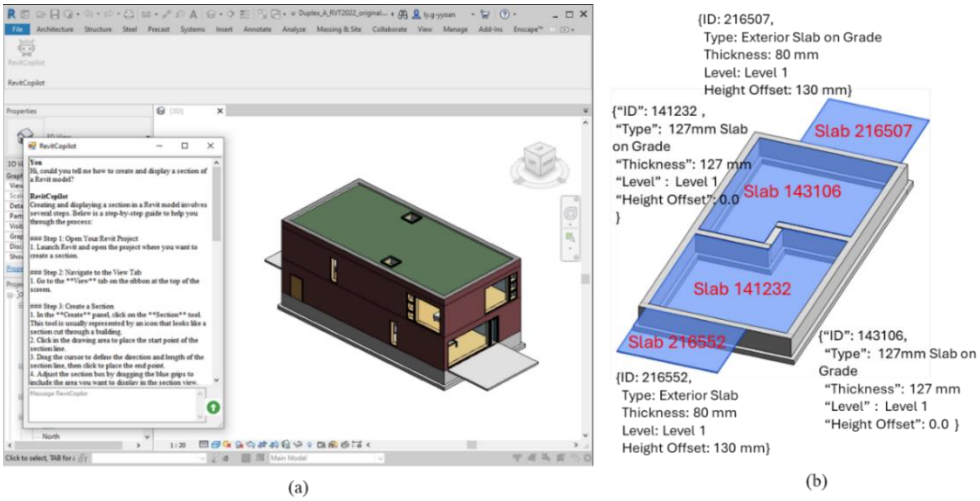


Figure 4. Interface of RevitCopilot (a) and four target ground slabs in the sample BIM model (b).

4.2.2 Design requirement query

According to Section 3.2, IBC 2009 Chapter 19 was preprocessed by partitioning, embedding, and storing it in a local file. RevitCopilot accesses this file to handle natural language-based design requirement queries. In this experiment, we queried specific design requirements for concrete floor slabs in buildings. It took two rounds of conversation with RevitCopilot to obtain a satisfactory answer, as shown in Figure 5. In the first round, RevitCopilot retrieved all the relevant design requirements for concrete floor slabs (highlighted in green in Figure 5a), but the results also included irrelevant items (highlighted in red in Figure 5b), which pertained to general concrete topics such as concrete pouring and inspection. This indicates that the retrieval step of the RAG approach generated irrelevant context for the LLM, leading to an inaccurate answer.

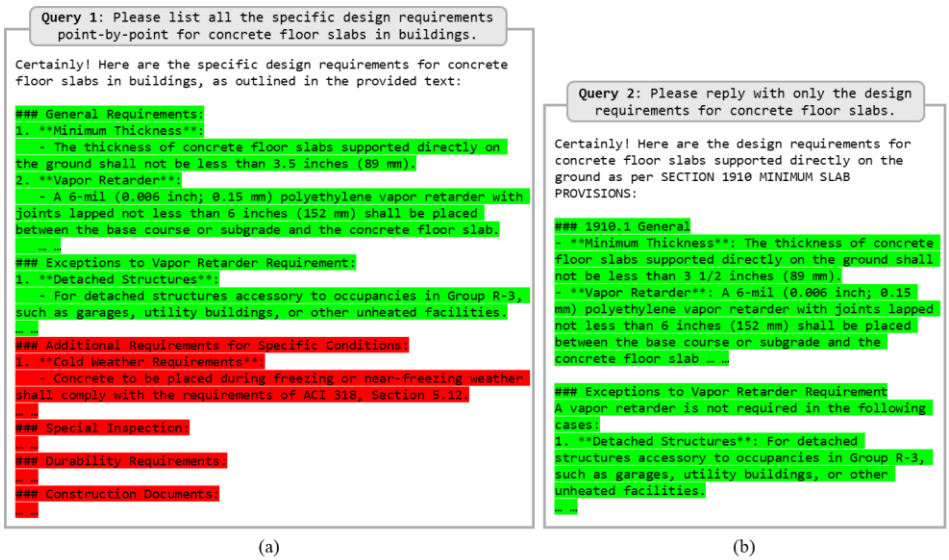


Figure 5. Two rounds of conversation for design requirement query: (a) first round; (b) second round. Note: the requirements colored in green are correct and those in red are irrelevant; "… …" indicates where text has been omitted for brevity.

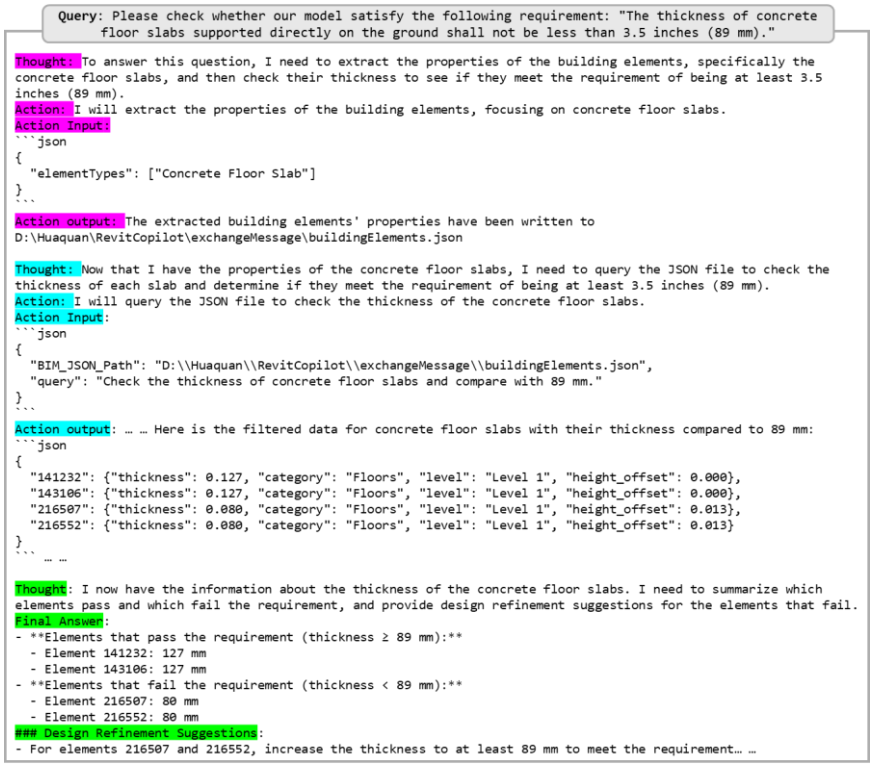
In the second round, by simply emphasizing the design requirements, RevitCopilot generated a correct and satisfactory answer. This highlights the benefit of the conversational task-solving paradigm, which allows for flexibility and dynamic correction of errors during the process. In fact,

the second query acted as a form of self-reflection, albeit facilitated by the user. Self-reflection has been shown to enhance LLMs' reasoning capabilities. Moving forward, we plan to embed a self-reflection mechanism into the RAG approach to enable more accurate, efficient, and robust design requirement querying.

4.2.3 Autonomous design checking

To demonstrate the feasibility of the autonomous design checking capability, we selected the first requirement returned in the previous step: "The thickness of concrete floor slabs supported directly on the ground shall not be less than 3.5 inches (89 mm)". Figure 6 shows the checking process autonomously planned and executed by the LLM agent. The agent completed the checking process through two reason-and-act cycles. In the first cycle (the pink sections in the figure), the agent interpreted the design requirement and broke the task into two simpler ones: extracting properties of concrete floor slabs and comparing their thickness with the specified value of 89 mm. The agent then executed the first task by extracting the semantic properties of all floor slabs from the Revit model and saving the retrieved data in a JSON file. The resulting JSON file contained 28 properties for each of all 20 floor slabs, including the four ground slabs and 16 other floor slabs.

In the second cycle (blue sections in the figure), the agent first planned to query the stored JSON file to extract the thicknesses of the slabs. It then executed the query, retrieving four concrete floor slabs and their respective thicknesses. Interestingly, despite the agent's output descriptions in both cycles seemingly overlooking the constraint that the target slabs should be "directly on the ground", the retrieved slabs were indeed those on the ground. We believe the agent considered this constraint during actions but did not provide a rigorous description. The inference is based on observations of running the agent multiple times: in several instances, the agent explicitly explained its method for locating the four ground floor slabs, based on the "Level" and "Height Offset from Level" properties. In fact, the user's query consistently provided the task context, ensuring the agent had the necessary information to consider this constraint.



Query: Please check whether our model satisfy the following requirement: "The thickness of concrete floor slabs supported directly on the ground shall not be less than 3.5 inches (89 mm)."

Thought: To answer this question, I need to extract the properties of the building elements, specifically the concrete floor slabs, and then check their thickness to see if they meet the requirement of being at least 3.5 inches (89 mm).
Action: I will extract the properties of the building elements, focusing on concrete floor slabs.
Action Input:

```json
{
  "elementTypes": ["Concrete Floor Slab"]
}
```

Action output: The extracted building elements' properties have been written to D:\Huaquan\RevitCopilot\exchangeMessage\buildingElements.json

Thought: Now that I have the properties of the concrete floor slabs, I need to query the JSON file to check the thickness of each slab and determine if they meet the requirement of being at least 3.5 inches (89 mm).
Action: I will query the JSON file to check the thickness of the concrete floor slabs.
Action Input:

```json
{
  "BIM_JSON_Path": "D:\\Huaquan\\RevitCopilot\\exchangeMessage\\buildingElements.json",
  "query": "Check the thickness of concrete floor slabs and compare with 89 mm."
}
```

Action output: … … Here is the filtered data for concrete floor slabs with their thickness compared to 89 mm:

```json
{
  "141232": {"thickness": 0.127, "category": "Floors", "level": "Level 1", "height_offset": 0.000},
  "143106": {"thickness": 0.127, "category": "Floors", "level": "Level 1", "height_offset": 0.000},
  "216507": {"thickness": 0.080, "category": "Floors", "level": "Level 1", "height_offset": 0.013},
  "216552": {"thickness": 0.080, "category": "Floors", "level": "Level 1", "height_offset": 0.013}
}
```

… …

Thought: I now have the information about the thickness of the concrete floor slabs. I need to summarize which elements pass and which fail the requirement, and provide design refinement suggestions for the elements that fail.
Final Answer:
- **Elements that pass the requirement (thickness ≥ 89 mm):**
  - Element 141232: 127 mm
  - Element 143106: 127 mm
- **Elements that fail the requirement (thickness < 89 mm):**
  - Element 216507: 80 mm
  - Element 216552: 80 mm

Design Refinement Suggestions:
- For elements 216507 and 216552, increase the thickness to at least 89 mm to meet the requirement… …

Figure 6. Recorded autonomous requirement checking process of RevitCopilot. Note that "… …" indicates where text has been omitted for brevity; the two reason-and-act cycles are colored in pink and blue, respectively; the items colored with green represent the final answers.

After accurately identifying the ground floor slabs and obtaining their thickness, the agent correctly identified two slabs (IDs 141232 and 143106) that meet the requirement and two slabs (IDs 216507 and 216552) that do not, aligning with the ground truth shown in Figure 4(b).

Note that different runs of the agent can produce varying process descriptions, and some runs result in incorrect results, highlighting the inherent uncertainty of LLMs. To improve stability and robustness, more advanced prompting techniques like few-shot prompts, should be considered.

4.2.4 BIM design revision

For the two slabs that fail to meet the design requirements, designers can instruct RevitCopilot to adjust them accordingly. For instance, after sending the following request to RevitCopilot: "Please change the thickness of concrete floor slabs 216507 and 216552 to 127 mm." RevitCopilot activated the design revision function and successfully executed this request (see Figure 7).

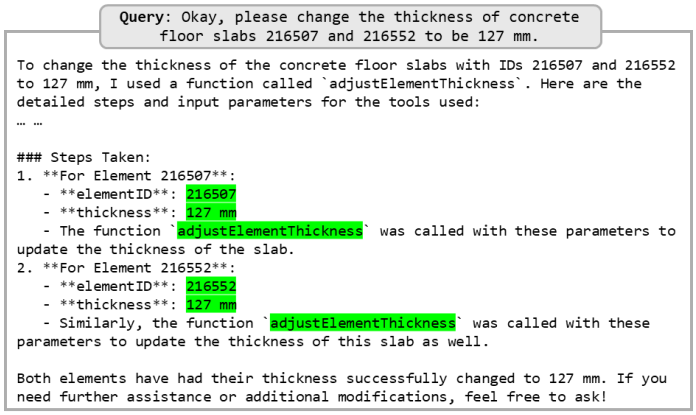


Figure 7. Explanation of the BIM design revision generated by RevitCopilot.

5 Discussion

5.1 Novel autonomous paradigm for building design checking

In this paper, we propose a novel LLM-driven autonomous paradigm for building design compliance checking, demonstrating strong potential to overcome significant limitations of current ACC methods. Our approach features several key characteristics.

First, unlike existing ACC systems that follow a traditional pre-programmed automation paradigm, our approach checks rules autonomously using the provided tools. This autonomy makes our approach generic and scalable, capable of handling new rules simply by expanding the toolbox. Furthermore, our approach processes design requirements directly in various original formats including text, figure, and table, demonstrating the potential for full automation.

Second, our approach supports a natural language-driven, conversational, and dynamic rule checking process. Thus, users can participate actively in the process, which is particularly useful for handling complex and ambiguous rules. According to Zhang et al (2023), around 53% of building requirements contain ambiguities. Our human-in-the-loop mechanism allows users to inject domain knowledge and intents during the process to address these ambiguities. For instance, when checking the design requirement "*Waiting areas should be close to the clinical or work area served and WC facilities*" (Zhang et al., 2023), our approach has the potential to identify the ambiguity in the relationship of "close to" and prompt the user for clarification before proceeding with the checking process.

Third, as demonstrated in Section 4.2, our system is fully transparent, presented in natural language. This allows users to understand the process easily, even without extensive domain knowledge. This builds trust and reliability, which is essential particularly for building permitting.

Fourth, our approach supports natural language-based design requirement query and BIM design revision, seamlessly integrating design checking into the design process. This integration improves design efficiency and quality by allowing designers to conduct compliance checks at any time during the design period. In contrast, current ACC practices typically separate design and checking, mainly due to the separation between ACC systems and BIM modeling environment.

5.2 Challenges and limitations

While the experiment demonstrated the promising potential of using an LLM agent-driven autonomous approach for generic, scalable, and fully automated design checking, several

challenges and limitations persist due to current LLM constraints and the complex nature of design checking. Addressing these challenges is essential for its practical application.

First, LLMs are trained on vast datasets, but it's unclear if they include sufficient domain-specific data for the AEC industry. This affects the LLMs' ability to understand and execute design requirements accurately.

Second, LLMs often produce different responses to the same query across different runs. While this variability can be beneficial for creative applications (e.g., storytelling, art and design generation, and brainstorming), it is undesirable for engineering tasks like ACC, which require consistent and reproducible results. Defining clear instructions in prompts can help, but it does not fully mitigate this issue. In the future, researchers will need to experiment extensively to evaluate the accuracy, consistency and robustness of the proposed approach, as these are crucial to its practical application. Appropriate levels of professional knowledge and suitable training may be required to help ensure the consistency of the queries submitted to the system.

Third, to ensure the system is sufficiently generic to cover all potential design requirements, a comprehensive set of tools must be provided to the LLM agent. It is unreasonable to develop every tool that may be needed for compliance checking in advance, due to the vast number of diverse design requirements. A potential solution includes combining the options of leveraging LLMs' coding capabilities to make necessary tools themselves and enabling web searches for existing functions and algorithms, thereby minimizing manual tool development efforts.

Fourth, the LLM-driven approach can be expensive, as each LLM call incurs fees based on the number of input and output tokens. For design checking, large BIM datasets are frequently used as input tokens, and extensive rule checking descriptions need to be generated. Therefore, it's essential to implement the approach carefully to optimize its inputs and outputs to control costs.

Lastly, the use of private BIM data as contextual information for each query poses a risk of data leakage, as this data must be sent to an LLM server controlled by an external company. Ensuring data privacy and security is crucial when implementing this LLM-based approach.

6 Conclusions

In this paper, we propose an LLM agent-driven approach for generic, scalable, and fully automatic building compliance checking. Our autonomous approach offers promising solutions to significant and longstanding limitations of current automated methods. Key theoretical advantages include handling diverse requirements in various formats, supporting conversational and dynamic checking that allows incorporating human input for ambiguous requirements, and ensuring full transparency with all checks documented in natural language. The approach has been implemented as a Revit Plugin, supplemented with two additional LLM-driven modules for better integration of the compliance checking function into design processes. One module allows natural language-based design requirement queries while the other enables natural language-based BIM design revisions based on the results of compliance checks. A preliminary experiment demonstrates the feasibility of our approach.

Our future work will focus on enhancing each module of the LLM-based autonomous compliance checking approach and conducting extensive experiments with varied and complex design requirements for comprehensive evaluation. Specifically, for the design requirement query module, we aim to improve query precision and enable processing requirements with cross-references using agent technologies. The compliance checking module will be upgraded to a multi-agent system to address complex requirements and enhance reporting for 3D visualization of the checking results. For the design revision module, we will develop an agent capable of more intricate revisions.

References

Amor, R. & Dimyadi, J. (2021). The promise of automated compliance checking. *Developments in the built environment*, 5, p.100039.

Bloch, T. (2022). Connecting research on semantic enrichment of BIM-review of approaches, methods and possible applications. *Journal of Information Technology in Construction*, 27, pp.416-440.

Eastman, C., Lee, J.M., Jeong, Y.S. & Lee, J.K. (2009.) Automatic rule-based checking of building designs. *Automation in construction*, 18(8), pp.1011-1033.

Fitkau, I. & Hartmann, T. (2024). An ontology-based approach of automatic compliance checking for structural fire safety requirements. *Advanced Engineering Informatics*, 59, p.102314

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J. & Wang, H. (2023). Retrieval-augmented generation for large language models: A survey. arXiv preprint arXiv:2312.10997

Jang, S., Lee, G., Oh, J., Lee, J. & Koo, B. (2024). Automated detailing of exterior walls using NADIA: Natural-language-based architectural detailing through interaction with AI. *Advanced Engineering Informatics*, 61, p.102532.

Jiang, G., Ma, Z., Zhang, L. & Chen, J. (2024). EPlus-LLM: A large language model-based computing platform for automated building energy modeling. *Applied Energy*, 367, p.123431.

Jiang, L., Shi, J. & Wang, C. (2022). Multi-ontology fusion and rule development to facilitate automated code compliance checking using BIM and rule-based reasoning. *Advanced Engineering Informatics*, 51, p.101449.

Lee, J.K., Eastman, C.M. & Lee, Y.C. (2015). Implementation of a BIM domain-specific language for the building environment rule and analysis. *Journal of Intelligent & Robotic Systems*, 79, pp.507-522.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.T., Rocktäschel, T. & Riedel, S. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33, pp.9459-9474.

Meta (2024). Build the future of AI with Meta Llama 3. https://llama.meta.com/llama3/

OpenAI (2024a). Hello GPT-4o. https://openai.com/index/hello-gpt-4o/

OpenAI (2024b). New embedding models and API updates. https://openai.com/index/new-embedding-models-and-api-updates/

Pauwels, P., van den Bersselaar, E. & Verhelst, L. (2024). Validation of technical requirements for a BIM model using semantic web technologies. *Advanced Engineering Informatics*, 60, p.102426

Preidel, C. & Borrmann, A. (2016). Towards code compliance checking on the basis of a visual programming language. *Journal of Information Technology in Construction*. 21(25), pp.402-421.

Saka, A., Taiwo, R., Saka, N., Salami, B.A., Ajayi, S., Akande, K. & Kazemi, H. (2023). GPT models in construction industry: Opportunities, limitations, and a use case validation. *Developments in the Built Environment*, p.100300.

Solibri, 2024. https://www.solibri.com/

Solihin, W., Dimyadi, J., Lee, Y.C., Eastman, C. & Amor, R. (2020). Simplified schema queries for supporting BIM-based rule-checking applications. *Automation in construction*, 117, p.103248.

Solihin, W. & Eastman, C. (2015). Classification of rules for automated BIM rule checking development. *Automation in construction*, 53, pp.69-82.

Unstructured (2024). https://unstructured.io/

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y. & Zhao, W.X. (2024). A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6), pp.1-26.

Wang, X. & El-Gohary, N. (2023). Deep learning-based relation extraction and knowledge graph-based representation of construction safety requirements. *Automation in Construction*, 147, p.104696.

Wang, Z, Sacks, R, Ouyang, B, Ying, H, & Borrmann, A. (2024). A Framework for Generic Semantic Enrichment of BIM Models. *Journal of Computing in Civil Engineering*, 38(1), p.04023038.

Xu, X. & Cai, H. (2020). Semantic approach to compliance checking of underground utilities. *Automation in Construction*, 109 (2020): 103006.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. & Cao, Y. (2022). React: Synergizing reasoning and acting in language models. arXiv preprint arXiv:2210.03629.

Ying, H. & Lee, S. (2021). Generating second-level space boundaries from large-scale IFC-compliant building information models using multiple geometry representations. *Automation in Construction*, 126, p.103659.

Zhang, J. & 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), p.04015014.

Zheng, J. & Fischer, M. (2023). Dynamic prompt-based virtual assistant framework for BIM information search. *Automation in Construction*, 155, p.105067.

Zhang, Z., Ma, L. & Nisbet, N. (2023). Unpacking ambiguity in building requirements to support automated compliance checking. *Journal of Management in Engineering*, 39(5), p.04023033.