electronics

MDPI

Article

Large Language Model-Driven Code Compliance Checking in Building Information Modeling

Soumya Madireddy [1,†], Lu Gao [1,*,†], Zia Ud Din [1,†], Kinam Kim [1,†], Ahmed Senouci [1,†], Zhe Han [2,†] and Yunpeng Zhang [3,†]

[1] Department of Construction Management, University of Houston, Houston, TX 77004, USA; smadire3@cougarnet.uh.edu (S.M.); uziauddi@central.uh.edu (Z.U.D.); kkim48@central.uh.edu (K.K.); asenouci@central.uh.edu (A.S.)
[2] Center for Transportation Research, The University of Texas at Austin, Austin, TX 78712, USA; hanzhe@utexas.edu
[3] Department of Information Science Technology, University of Houston, Houston, TX 77004, USA; yzhan226@central.uh.edu
* Correspondence: lgao5@central.uh.edu
† These authors contributed equally to this work.

Abstract: This research addresses the time-consuming and error-prone nature of manual code compliance checking in Building Information Modeling (BIM) by introducing a Large Language Model (LLM)-driven approach to semi-automate this critical process. The developed system integrates LLMs such as ChatGPT, Claude, Gemini, and Llama, with Revit software to interpret building codes, generate Python scripts, and perform semi-automated compliance checks within the BIM environment. Case studies on a single-family residential project and an office building project demonstrated the system's ability to reduce the time and effort required for compliance checks while improving accuracy. It streamlined the identification of violations, such as non-compliant room dimensions, material usage, and object placements, by automatically assessing relationships and generating actionable reports. Compared to manual methods, the system eliminated repetitive tasks, simplified complex regulations, and ensured reliable adherence to standards. By offering a comprehensive, adaptable, and cost-effective solution, this proposed approach offers a promising advancement in BIM-based compliance checking, with potential applications across diverse regulatory documents in construction projects.

Keywords: BIM; semi-automated compliance checking; code compliance checking; AI; LLM

Academic Editor: Domenico Rosaci

Received: 7 April 2025
Revised: 14 May 2025
Accepted: 21 May 2025
Published: 24 May 2025

Citation: Madireddy, S.; Gao, L.; Din, Z.; Kim, K.; Senouci, A.; Han, Z.; Zhang, Y. Large Language Model-Driven Code Compliance Checking in Building Information Modeling. *Electronics* **2025**, *14*, 2146. https://doi.org/10.3390/electronics14112146

1. Introduction

1.1. Code Compliance

Ensuring that construction projects adhere to building codes and regulations is a critical aspect of the architecture, engineering, and construction (AEC) industry [1]. Traditionally, this process has been manual, relying on senior professionals to review designs using Computer-Aided Design (CAD) drawings and specifications, which is time-consuming, inefficient, and prone to errors [2]. Automated compliance checking (ACC) has emerged as a solution to improve accuracy and efficiency by using computer programs to check building projects against codes. The use of Building Information Modeling (BIM) has enhanced ACC by providing a digital representation of a building, enabling better collaboration and information exchange [3].

The core of ACC involves translating building codes and regulations into a machine-interpretable format [4]. This process includes several key steps: rule interpretation, building model preparation, rule execution, and result reporting [5]. Rule interpretation is considered the most vital and complex stage, often involving techniques like using existing software, creating plug-in applications, or adopting object-based, logical, or ontological approaches [3]. The use of natural language processing (NLP) is also important for interpreting regulatory texts and converting them into computable rules [6]. The building model is prepared by extracting the necessary information from BIM models, including geometric and property data. During rule execution, the interpreted rules are applied to the prepared model, and any violations are recorded [7].

These steps are increasingly supported by advanced digital technologies, such as BIM, NLP, and machine learning, which significantly improve the accuracy and speed of compliance checks [8]. For example, Li et al. [9] proposed an integrative framework for automated compliance checking in BIM models, utilizing knowledge graphs and NLP to identify errors based on building standards, specifically in architectural and fire safety contexts. Lange et al. [10] developed a machine learning system to automate accessibility compliance checking in BIM designs; the system utilized a Convolutional Neural Network (CNN) to analyze BIM models and identify accessibility issues, such as urban surfaces or excessive ramp slopes with an accuracy of 95%.

1.2. Challenges in Current ACC Systems

The current ACC systems face several persistent challenges that hinder their widespread implementation and effectiveness. A significant issue lies in translating the complex and often ambiguous language of building codes into a machine-interpretable format [11]. Building codes frequently change, with new requirements added regularly, making it difficult to maintain and update these systems, especially when they rely on hard-coded rules [12]. Also, the interpretation of regulatory texts using NLP tools is complicated by the legal and technical nuances of the text [13].

Another critical challenge is the reliance on detailed, accurate, and complete building information, as deficiencies in BIM models, such as missing or incorrect data, can significantly hinder the automated checking process [14]. The lack of standardization in BIM data further exacerbates these problems, often necessitating extensive manual preprocessing to correct inconsistencies and omissions [15]. Furthermore, existing ACC systems often operate as "black boxes", lacking transparency and flexibility, which makes them difficult to understand or modify for specific user needs [4]. These systems also struggle with addressing qualitative aspects of building codes, such as aesthetics and spatial functionality, and face scalability issues as building designs become increasingly complex [16].

Despite these challenges, ACC systems have demonstrated significant potential to improve the speed, accuracy, and consistency of code compliance checks. By leveraging advanced digital technologies such as BIM, NLP, and machine learning, these systems can save time and resources while reducing the likelihood of human error [8].

1.3. Emergence of LLMs and Its Transformative Potential

The emergence of LLMs has introduced a transformative potential for ACC by offering advanced capabilities in natural language processing [5]. LLMs, pre-trained on vast amounts of data, demonstrate a strong ability to understand and generate human language with minimal task-specific training. This enables them to interpret complex regulatory texts and potentially convert them into computable rules more efficiently than previous methods [17]. LLMs can also adapt to new regulations and extract structured information from regulatory texts. LLMs also show promise in generating formal representations of

regulations, potentially replacing the need for manual rule creation and improving the overall efficiency and effectiveness of checking processes [18]. However, despite their great potential, there are still challenges to be addressed, such as ensuring the accuracy of the generated outputs and the dependence on prompt engineering to guide their responses effectively. These challenges show that more research is needed to make full use of LLMs for automated compliance checking.

1.4. Objectives of This Research

While previous research has explored the application of LLMs to ACC, there are still challenges that remain unresolved. Existing compliance-checking models lack integration with BIM environments. Many platforms are restricted to specific regulations and difficult to adapt to changes. To address this, we propose an LLM-based approach that converts regulations into executable Python 3.13.2 scripts for real-time compliance checking in Revit 2024.

2. Literature Review

2.1. Importance of Code Compliance in Construction

Code compliance in construction is crucial for building safety, legal adherence, and efficiency through automation [19]. Building codes are legal documents designed to protect public safety by establishing minimum standards for construction [20]. They specify requirements for various aspects of building design and construction, such as fire safety, accessibility, and structural integrity [21]. By adhering to these codes, construction projects can avoid safety hazards that may lead to accidents, injuries, or even fatalities [22]. ACC systems play a crucial role in enhancing safety by reducing human error during the design, review, and construction planning phases [1]. These systems can identify potential safety issues early in the design process, allowing for timely modifications and preventing costly and potentially dangerous rework. Moreover, automated systems can check for specific safety concerns like fall hazards and spatial relationships, ensuring a safer environment for both construction workers and future building occupants [22].

Code compliance also significantly impacts the efficiency of construction projects [4]. Manual code checking is a time-consuming and error-prone process, often requiring extensive resources and leading to project delays and increased costs [3]. However, ACC systems offer a solution by streamlining the process, saving time, money, and labor [22]. BIM technology facilitates automated checking by providing a digital representation of building designs and related data, making it possible to perform checks more quickly and accurately [13]. By detecting and addressing potential issues during the design phase, ACC helps to prevent costly rework [20]. Furthermore, ACC can streamline the building permit process, promote better collaboration among project participants, and produce inspection reports quickly [21].

Furthermore, code compliance is critical for legal adherence [2]. Building codes are legally binding and failure to comply can result in significant legal disputes, project delays, and financial penalties. Standardized codes and automated systems also promote greater consistency across jurisdictions, reducing confusion for designers and builders [22]. A system that can perform checks with integrity and credibility is essential for maintaining trust and preventing legal problems in construction projects [16].

2.2. Methods for Ensuring Compliance

The development of methods for ensuring compliance in the AEC industry has progressed from manual, error-prone processes to sophisticated automated systems, driven by technological advancements, the increasing complexity of building codes, and the need for

efficiency and accuracy [22]. Initially, compliance relied on manual interpretation and review of design drawings and specifications by experienced professionals [7]. This involved a time-consuming and costly process where senior personnel examined drawings, often repeating similar checks, and was highly susceptible to errors [13]. These manual checks were limited by the fact that professionals could not memorize all the codes, which were often contradictory [1].

The introduction of computer-aided design (CAD) in the 1980s brought a transition to digital methods, but early CAD systems still relied on manual checking of drawings and textual descriptions, simply digitizing existing workflows. These systems lacked the ability to perform intelligent rule-based checks and were essentially digital versions of the manual process [1]. BIM marked a significant shift in compliance checking by integrating building information with rule-based checking [5]. BIM is a digital model that captures the physical and functional characteristics of a facility. It acts as a shared knowledge resource to aid decision-making throughout a facility's life [23]. BIM technology has revolutionized the construction industry by improving collaboration, accuracy, and project management [24]. Initially focused on 3D modeling, BIM has expanded to incorporate 4D (time), 5D (cost), and 6D (sustainability) [25]. BIM allowed for the creation of digital models containing not only the geometry of a building but also extensive data about its components and systems. Early BIM-based approaches used hard-coded rules directly into software, where compliance rules were embedded within the software's code [26]. These early systems were inflexible, difficult to modify, and often lacked transparency, acting as "black boxes" [19].

To address the limitations of inflexibility and lack of transparency, semi-automated methods were introduced [5]. These methods aimed to translate regulatory text into machine-processable formats using logical operators, while still requiring some manual effort. The use of predicate logic allowed for the validation of checking methods, calculations, and conditions in the rules, expressing rules as logical conditions that could be evaluated. The deontological approach, using deontic logic, was introduced for more complex knowledge representation and reasoning [27].

Further advancements included the use of NLP techniques to automate the extraction of information from regulatory texts [5]. These techniques converted unstructured text into structured information for automated reasoning. Early NLP approaches included both rule-based and statistical methods, with rule-based methods generally offering better accuracy but requiring more human labor. Researchers explored methods to extract semantic and syntactic information and to categorize text in regulatory documents to improve efficiency [28]. The development of ontologies also played a key role, enabling the representation of domain knowledge and supporting semantic reasoning [29].

More recently, there has been a focus on utilizing visual programming languages to make rule-making and compliance checking more accessible to non-programmers. The Visual Code Checking Language (VCCL) allows users to visually translate codes and formalize them [19]. Domain-specific languages, such as the Building Environment and Analysis Language (BERA), provide a mean to encode complex rules regarding spatial and circulation requirements [4].

The emergence of LLMs like ChatGPT has demonstrated the potential to automate the translation of natural language requirements into computable representations [17]. LLMs can address the limitations of deep learning by providing robust language understanding with minimal labeled data, adapting to evolving regulations, and accurately extracting structured information from regulatory texts [5].

Knowledge graphs are also being explored to structure and store knowledge from BIM standards, enabling rule-based systems and machine learning to be used for compliance checking [26]. These methods correlate information, allow for information reuse, and

fully express the constraints between building entities. The use of deep learning for pre-classifying regulatory texts can improve the accuracy of structured information extraction by LLMs [5].

Current research also includes addressing the complexity of translating natural language rules, dealing with spatial and geometric relationships, and improving the transparency and usability of these systems [13]. There is also a push to create digital libraries of rule sets that can be shared online, subdivided by geographical location, which would unite the controls that must comply with a specific regulation [30]. The development of an open format for these rule sets could guarantee interoperability between model-checking software. Many studies are now focusing on the integration of LLMs, deep learning models, and ontology knowledge models to improve the efficiency and accuracy of compliance checks [5]. Also, there is a growing interest in making these systems more user-friendly and capable of enhancing efficiency and compliance in BIM modeling and procurement processes [15].

2.3. Current Automated Tools and Their Limitations

Current ACC systems face several challenges, primarily stemming from the complexities of translating natural language regulations into machine-readable rules and the difficulties in ensuring the completeness and accuracy of BIM [16]. One significant challenge is the difficulty in translating building code text into machine-readable rules [11]. Building codes are written in natural language, which is not easily interpreted by computers. The language used in building codes can be ambiguous and complex, with cross-references and inconsistent relationship displays, which increases the complexity of interpreting the sentences and creates discrepancies [4]. Moreover, building codes often have national, regional, and cultural variations in wording and application, making it difficult to develop a universally applicable system [16]. This issue is further compounded by the fact that building codes are updated frequently, requiring constant modifications to the automated systems [22]. The lack of a standardized format for building codes and the absence of a unified conclusion in the ACC domain also contribute to interoperability issues between BIM and ACC systems [31].

Another major challenge is the need for extensive preprocessing and preparation of BIM models for checking [14]. Existing platforms often require users to manually supplement missing information, correct inaccuracies, or address incomplete data in the model before checking can begin. This process, known as normalization, is labor-intensive, time-consuming, and prone to errors. Furthermore, the complexity of building designs increases the difficulty of ensuring that all necessary information is included and accurate in the model [3]. The lack of clearly defined information requirements and variations in modeling practices also contribute to inconsistencies and errors in BIM data [15]. Moreover, many systems are too focused on specific domains, such as particular building codes or safety regulations, which limits their general applicability and integration with other systems [32].

To address these challenges, several software tools have been developed to automate compliance checking directly within BIM environments, particularly Revit. For instance, UpCodes AI provides real-time in-model checking using AI, flagging violations related to stairs, doors, ramps, and clearances while linking them to the relevant code sections [33]. SMARTreview offers in-depth analysis of the International Building Code (IBC) through its Revit plugin and generates formal compliance reports accepted by some city permitting departments [34]. Solibri, while external, allows for extensive rule-based checking, including accessibility and egress, by importing Revit models through IFC [35]. Autodesk's own Model Checker is a free, configurable tool within Revit, enabling users to create custom

code rules [36]. icheck focuses on California's accessibility code (CBC Chapter 11B) [37], and EvolveLAB's Revit Code Tools automate occupancy, egress, and plumbing calculations using Revit schedules and tags [38].

Although these tools offer varying levels of automation and sophistication, they are not without limitations. Many of them rely on hard-coded rules, which restrict flexibility, making it difficult to adapt to new or changing regulations. Others may struggle with complex geometric relationships, require extensive manual configuration, or are constrained by their focus on specific code standards. These limitations reduce their ability to dynamically adapt to diverse regulatory requirements and complex BIM models.

Furthermore, many existing ACC systems also suffer from a lack of transparency and flexibility [4]. Hard-coded rules, while enabling the checking of specific provisions, are difficult to maintain, modify, and scale [22]. Users are often limited to predefined checking capabilities and cannot customize or adjust the rules to meet their specific project needs [39]. This lack of user involvement and transparency reduces the acceptance of these systems among domain experts [19]. On top of that, current systems may lack the ability to check complex, geometric relationships between components [6].

2.4. Applications of Large Language Models in AEC

LLMs represent a major advancement in artificial intelligence (AI), showcasing transformative potential across various sectors, including the AEC industry [22]. These models, built on transformer architectures and trained on extensive datasets, excel at complex language-related tasks such as translation, summarization, and content generation [40]. In construction management, LLMs have been increasingly utilized to automate tasks like translating building regulations into computable formats and integrating regulatory requirements into compliance systems. For example, Fuchs et al. [18] demonstrated how GPT-3.5 and GPT-4 could structure regulatory texts into machine-readable formats, while Zhang [17] showed that ChatGPT could generate Python code from regulations, showcasing the scalability of LLMs in compliance automation.

A key strength of LLMs lies in their robust natural language understanding and generation, enabled by pre-training on diverse and large-scale datasets. This equips them with the ability to identify complex patterns in language and adapt to downstream tasks with minimal fine-tuning through emergent capabilities like in-context and few-shot learning [5]. These features make LLMs particularly suitable for automating resource-intensive tasks in the AEC industry, such as ACC. Traditionally, ACC has been labor-intensive, requiring manual analysis of complex and ambiguous regulatory texts. LLMs streamline this process by extracting structured information, improving accuracy, and reducing errors [18].

Generative AI models like the GPT series have also demonstrated potential in architectural design through integration with BIM tools. Jang and Lee [41] showcased their use as design assistants, interpreting user inputs, suggesting materials, and updating BIM models with AI-driven recommendations. Similarly, He et al. [42] developed a framework using LLMs and a Physics-Based Conditional Diffusion Model (PCDM) to optimize structural designs, such as shear wall layouts, based on real-world conditions like seismic intensities and building heights.

By reducing dependence on large, annotated datasets and extensive manual feature engineering, LLMs overcome limitations of traditional methods, offering a more efficient and scalable approach to automation [18]. They also enhance the compliance process by employing pre-classification techniques and leveraging deep learning models to handle nested conditional statements and ambiguous regulatory language. These advancements

highlight their potential to improve productivity, accuracy, and the overall quality of construction processes [5].

However, the application of LLMs in ACC is not without challenges. LLMs can still have difficulty handling highly complex regulatory texts with intricate structures, nested clauses, and conditional statements [5]. Furthermore, some studies have primarily focused on simpler regulatory texts, suggesting that more advanced strategies may be necessary to effectively process more complex information. Despite these limitations, LLMs offer promising solutions for automating the interpretation of regulatory texts, enhancing the efficiency and accuracy of compliance checking in the construction sector [17]. The ability to translate natural language into computable formats is a key advantage that LLMs bring to the AEC-FM field.

2.5. Research Gap

Despite previous studies' efforts to apply LLMs to ACC, several challenges persist in the field. There is still a lack of integration of compliance-checking models directly within the BIM environment, making it difficult to visually recognize compliance and its implications. Besides, there are limitations in utilizing certain types of regulatory documents, as many commercial rule-checking platforms are restricted to specific country- or state-based regulations and are not customizable to meet diverse needs. Existing systems often struggle to dynamically update compliance-checking processes when regulations change. To address these challenges, we investigated an LLM-based approach that can automatically convert regulations into executable Python scripts within a BIM environment.

3. Methodology

Figure 1 illustrates an LLM-based framework designed to streamline and enhance compliance verification within BIM platforms by leveraging the capabilities of Large Language Models (LLMs), such as GPT-3.5 and GPT-4. The semi-automated framework is divided into four interconnected components: input data, AI-based interpretation, rule-checking algorithm, and output.

The process begins with the input data phase, which integrates two primary sources of information: the BIM model and regulatory documents. The BIM model provides a detailed digital representation of the building/structure's design, including geometry, spatial relationships, construction elements, and associated metadata. Regulatory documents, on the other hand, include the legal and technical compliance standards that buildings must adhere to. These documents define rules, such as fire safety, accessibility, and structural integrity requirements, which the framework must validate against the BIM model.

The AI-based interpretation phase is the core of the framework, utilizing the NLP capabilities of LLMs such as GPT, Claude, Gemini, and Llama. During this phase, the LLMs analyze and interpret the regulatory documents. These models are capable of processing complex and unstructured textual information, converting human-readable regulatory standards into machine-readable logic. Based on the interpreted regulations, the LLMs generate Python scripts tailored for automated rule-checking. These scripts are executed within the Revit environment. If errors are produced in executing the generated scripts, the framework sends the error details back to the LLM for refinement. This feedback mechanism ensures the final scripts are functional and reduce the need for manual intervention.

The rule-checking algorithm phase executes the Python scripts generated in the previous phase to evaluate compliance. The scripts are executed within the Python shell of Revit. If the script execution encounters errors, such as syntax issues, missing data, or logical inconsistencies, these errors are extracted and sent back to the AI-based interpretation

phase for refinement. When the scripts execute successfully, the framework generates detailed compliance or non-compliance reports. These reports highlight areas where the BIM model meets the regulations and identify deviations or violations.

The output phase consolidates and presents the results of the rule-checking process. For compliant models, the framework generates a comprehensive report confirming adherence to all relevant regulations. For non-compliant models, it provides targeted recommendations to address the identified issues. These recommendations are actionable and guide users toward necessary modifications. Beyond basic compliance, the framework also offers additional insights, such as optimization suggestions for improving the BIM model's design and risk assessments to warn about potential issues related to non-compliance.

The framework incorporates a feedback that connects the rule-checking algorithm and AI-based interpretation phases. This loop ensures continuous improvement of the Python scripts. By combining the interpretive power of LLMs with the computational capabilities of BIM tools like Revit, the framework will reduce the time and effort required for compliance checks while enhancing accuracy and reliability.

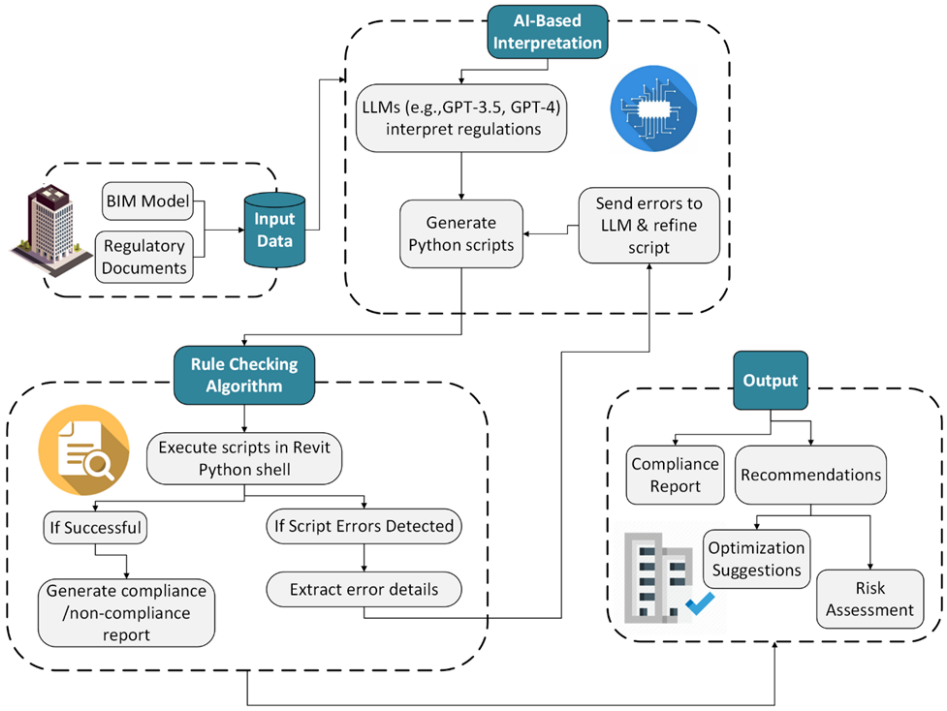

Figure 1. LLM-based framework for rule-checking in BIM.

4. Case Study

In this research, we conducted two case studies: one focused on a single-family residential project and the other on an office building.

4.1. Sample Rules

In the case studies, 12 sample rules were selected from sources such as the International Residential Code (IRC) [43], International Mechanical Code (IMC) [44], and other sources. These rules were converted into Python scripts, as shown in Table 1. These rules address dimensions, object relationships, materials, specific features and fixtures, structural elements, and mechanical components.

Table 1. Building code rules and references.

Rule No.	Rule Description	Existing Revit Data	Reference
1	The minimum width of the required exit is 36 inches (914 mm), with a net clear width of 32 inches (813 mm). The minimum height of a required exit is 6 feet 8 inches (2032 mm).	Properties of doors	IRC Section R311.2.1, Page 66 [43]
2	The minimum clear width of stairways shall be 36 inches.	Stair families	IRC Section R311.7.1, Page 68 [43]
3	Porches, balconies, ramps, or raised floor surfaces located more than 30 inches above the floor or grade shall have guards not less than 36 inches in height.	Properties of the guardrail families	IRC Section R312.1.1, Page 72 [43]
4	Habitable spaces, hallways, and portions of basements containing these spaces shall have a ceiling height of not less than 7 feet.	Floor levels	IRC Section R305.1, Page 56 [43]
5	The window-to-wall ratio in buildings shall not exceed 25% as stipulated by building code regulations. The ratio is influenced by energy efficiency standards, which might be covered under different codes or local amendments.	IRC Wall area, Window area	IRC Section R303 [43]
6	Every dwelling unit shall have at least one habitable room with not less than 120 square feet of gross floor area. Each additional habitable room, except kitchens, shall have a floor area of not less than 70 square feet.	Room tag	IRC Section R304.1 [43]
7	The IRC 2021 Section R307.2 requires a minimum clear space of 21 inches (533 mm) in front of water closets, lavatories, and bidets.	Bathroom fixtures	IRC Section R307.2 [43]
8	Toilet Facilities: Every dwelling unit must have a water closet, lavatory, bathtub, or shower.	Plumbing fixture schedules	IRC Section R306.1 [43]
9	Kitchen Requirements: Each dwelling unit must have a kitchen area with a sink.	Room tag & fixtures	IRC Section R306.2 [43]
10	Requirements for wood structural panels used in floor construction. It details material specifications and installation guidelines to ensure floor assemblies meet structural and fire safety requirements.	Material specifications	IRC Section R503.2.4 [43]
11	The edge-to-edge distance between any two footings must be at least equal to the width of the larger footing between them.	Footings	[45]
12	Minimum outdoor air ventilation rate required for occupied indoor spaces. For Office Spaces (Business Occupancy), the code states: Each room must receive outdoor air at a rate of: 5 CFM per person (people-based) 0.06 CFM per ft^2 of floor area (area-based). Total Minimum Ventilation = (5 × occupants) + (0.06 × floor area)	Mechanical Ducts	IMC 2021, Table 403.3.1.1 [44]

4.2. Prompt Engineering

As part of this research, we developed prompt engineering techniques to enhance the effectiveness of prompts used with LLMs like GPT-4. Figure 2 shows an example of the results of two different prompts using Rule 1 in Table 1 as an example. The initial prompt tested, shown as "Prompt A" in Figure 2, when given to an LLM model, generated a Python script. That script, when executed in Revit's PythonShell, returned an output with an error message, as shown in Result A in Figure 2. To generate a Python script that works in Revit, we tried a number of approaches, including sending the error message back to the LLM, using different phrasing for the prompts, and instructing the LLM to avoid certain types of errors. After several trial-and-error attempts, we were able to identify the optimized prompt structure that is guaranteed to generate an error-free Python script in the Revit PythonShell environment. The optimized prompt, Prompt B in Figure 2, was then applied to Rule 1 again, and the correct result was returned, as shown in Result B in Figure 2.

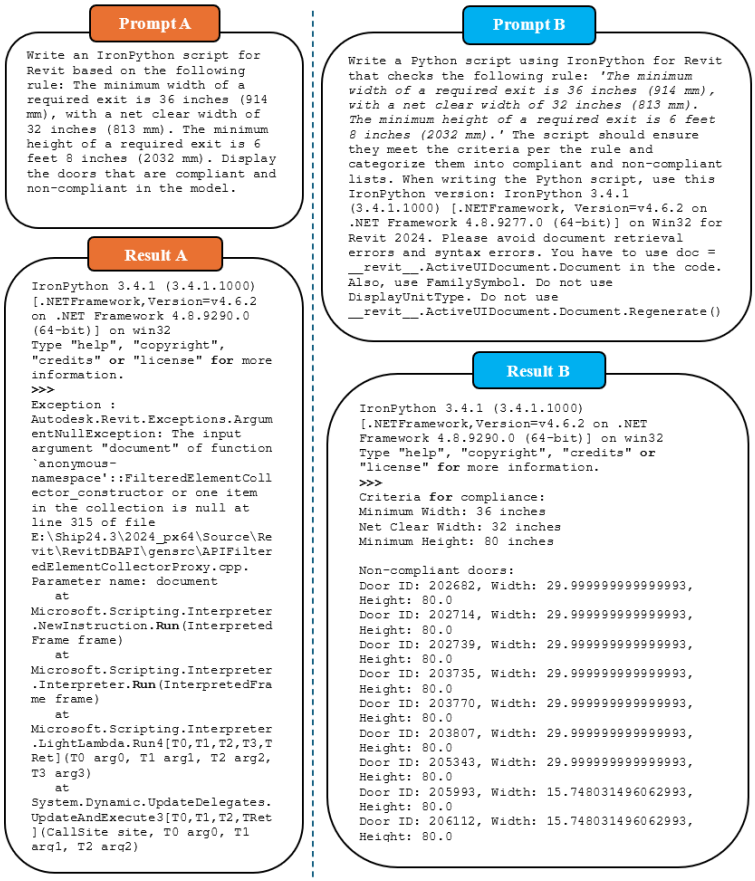

Figure 2. Example showcasing the result of Prompt A & Prompt B on LLM-generated Python script.

The optimized prompt's general structure is shown in Table 2. It consists of four components: the basic prompt, rule description, general instructions, and rule-specific instructions. The first component is the basic prompt, where we define the task that needs to be performed by the LLM. The second component is the rule description, where we provide the actual rule from the regulatory documents that needs to be verified. The third component is the general instructions, which ensure compatibility in the generated script. These instructions specify the versions of PythonShell and Revit being used to provide detailed technical context for the task. Also, since IronPython does not support f-strings, the instructions state that no f-strings should be used in the script to avoid compatibility issues. To further minimize errors, the general instructions emphasize avoiding syntax mistakes and ensuring the script adheres to Revit's requirements. They also address specific elements such as FamilySymbol and DisplayUnitType, and they guide the LLM to generate code that properly interacts with these components within the Revit environment. The last component, rule-specific instructions, focuses on those rules that need to access specific Revit elements, categories, or data attributes, which differ from rule to rule.

Table 3 below provides a set of rule-specific instructions that enable LLMs to generate context-appropriate outputs by focusing on relevant Revit elements (e.g., doors, stairs, windows), categories (e.g., ost_stairs, ost_plumbingfixtures), and data attributes (e.g., width, height, material properties). These instructions serve as examples for 10 IRC rules derived from Table 1. These rule-specific instructions are needed primarily because objects in a BIM model are defined differently by various users, and LLMs require instructions to generate scripts that can accurately extract information from the correct object.

Table 2. Components of the prompt for rule-based Python script generation.

Components of the Prompt	Description
Persona	"You are an expert Python developer with experience in Revit API."
Basic Prompt	"Write a Python script in IronPython for Revit to check the following rule."
Rule Description	Write the rule as follows. For example, for Rule 1, this part should be: "The minimum width of a required exit is 36 inches (914 mm), with a net clear width of 32 inches (813 mm). The minimum height of a required exit is 6 feet 8 inches (2032 mm)."
Context	"This script will be used within Revit 2024, leveraging IronPython 3.4.1 (.NET Framework 4.6.2 on .NET Framework 4.8.9277.0, 64-bit)."
General Instructions	The prompt must also include the following instructions: • Use the following IronPython version: ironpython 3.4.1 (3.4.1.1000), [.netframework, version=v4.6.2 on .net framework 4.8.9277.0 (64-bit)] on win32. • Use the 2024 version of Revit. • Avoid syntax errors and do not use f-strings. • Avoid document retrieval errors. Use doc = __revit__.activeuidocument.document in the code. • Include the familysymbol. • Do not use displayunittype. • Categorize elements into compliant and non-compliant lists.
Rule-Specific Instructions	See the details in Table 3. These instructions relate to the elements from the rule that need to be checked.
Format	• The Python script should be clean, well-structured, and follow Pythonic conventions. • Add comments to explain each section of the code.
Audience	"This script is intended for use by construction professionals with basic Python knowledge, working in Revit environments."

Table 3. Rule-specific instructions for LLMs.

Rule No.	Rule-Specific Instructions for AI
1	Not Applicable
2	stairs_collector = FilteredElementCollector(doc).OfClass(Stairs)
3	Use OST_StairsRailing
4	Use elevation levels as reference
5	Use OST_Windows and OST_Walls
6	Use OST_Floors
7	Not Applicable
8	Use OST_PlumbingFixtures
9	plumbing_fixtures_collector = FilteredElementCollector(doc).OfCategory(BuiltInCategory.OST_PlumbingFixtures)
10	[param.Definition.Name for param in element.Parameters]

4.3. Case Study 1

A case study of a single-family residential project, shown in Figure 3, was conducted to demonstrate the effectiveness of the proposed rule-checking framework. The project includes detailed architectural layouts, room dimensions, and refuge areas, providing a suitable dataset for evaluation. Although a small-scale model was used for testing, the compliance-checking process remains consistent across models of any scale.

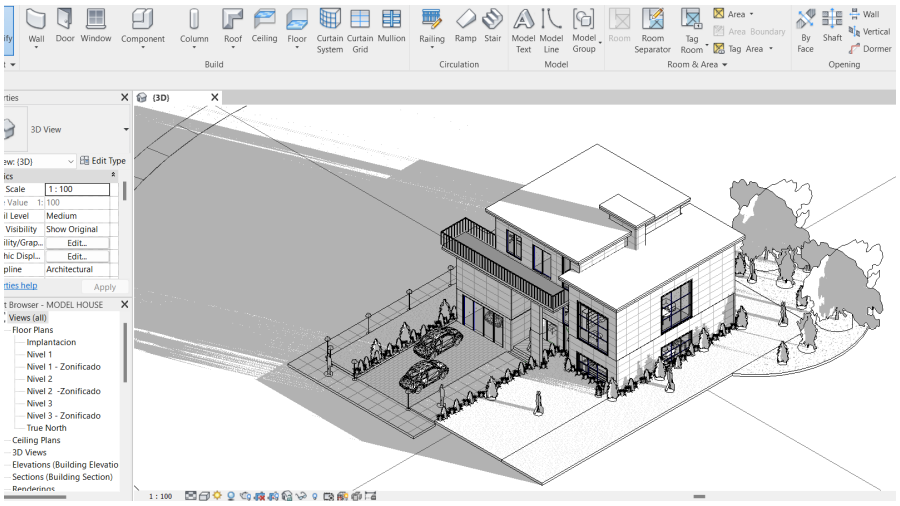

Figure 3. BIM model of a single-family house.

Using the optimized prompt structure discussed above, we generated Python scripts for the rules listed in Table 1. The scripts successfully identified both compliant and non-compliant aspects of the BIM model, and the details of the results are discussed in the section below. Figure 4 displays a list of compliant and non-compliant doors for Rule 1 from Table 1. It includes information about the Door ID along with its width and height dimensions. Doors that meet the minimum width of 36 inches and height of 80 inches are marked as compliant, while those with dimensions below these thresholds, such as a width of 30 inches or less, are classified as non-compliant. Users can utilize the Element ID search in Revit to locate and address the non-compliant elements.

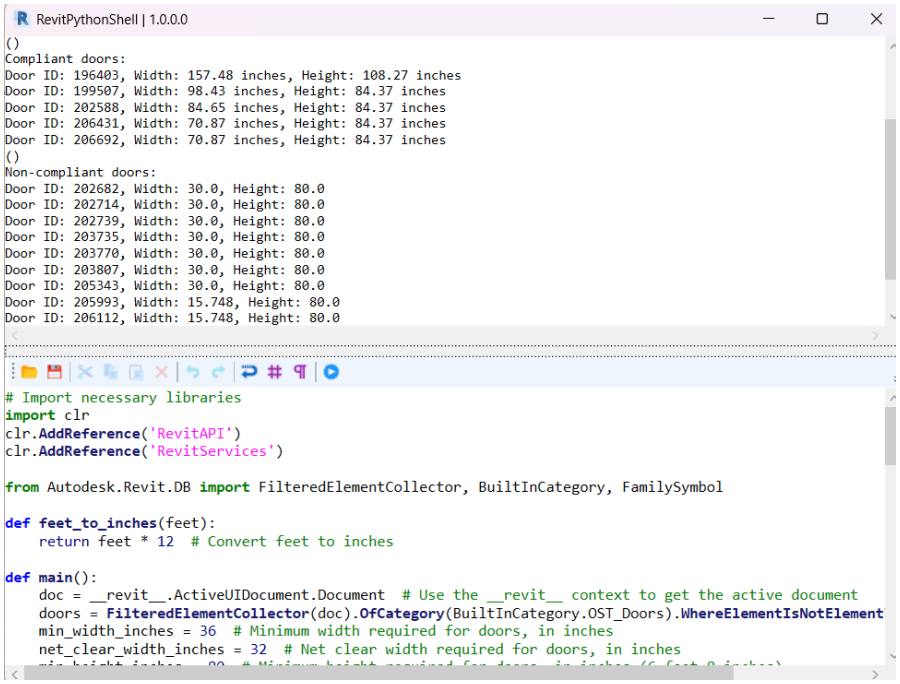

Figure 4. Script displaying both compliant and non-compliant doors.

Figure 5 measures the clear width of each stairway to verify if it meets the minimum width requirement of 36 inches (914 mm), as specified in Rule 2 from Table 1. The script extracts the relevant dimensions from the BIM model, converts the units if necessary, and compares them against the IRC requirements. The result demonstrates that the staircase is compliant.

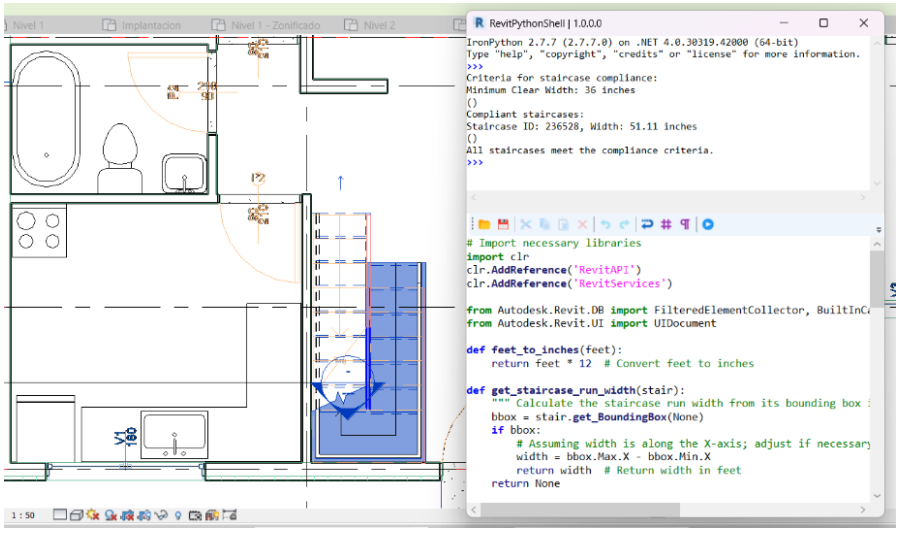

Figure 5. Verifying stairway width compliance with IRC 2021 in Revit using Python script.

Figure 6 is based on Rule 6 from Table 1. The Python script iterates through the habitable rooms, referencing the room tags, and checks whether the rooms have less than 120 square feet of gross floor area. It flags the floor IDs that fall below the minimum standard as non-compliant. For rules like these, the parameters within Revit play a major role. The BIM model must be well-developed with accurate room tags, names, and boundaries to ensure that the correct elements or areas are verified, rather than all rooms.

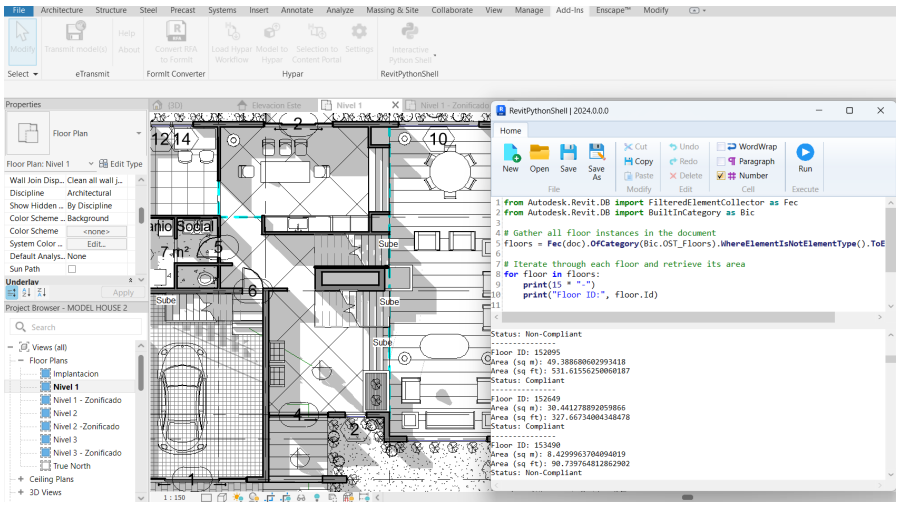

Figure 6. Compliance verification of room size standards using Python integration in Revit.

Figure 7 demonstrates the results for Rule 3, which requires guardrails on elevated surfaces to be at least 36 inches (914 mm) in height. The script in Figure 7 evaluates each guardrail in the model against this height requirement and flags any non-compliant elements. For instance, a non-compliant guardrail is flagged as Element ID: 654321, with a height of 35 inches and a guard height of 32 inches. Other examples of evaluating relationships between different objects include checking window-to-wall ratio compliance and clear space in front of bathroom fixtures.

Figure 8 verifies Rule 4 from Table 1. This rule checks whether all habitable spaces have a ceiling height of at least 7 feet. The output relies on the elevation-level parameters to verify the level difference between the floor and ceiling for these rooms. The image below shows that the ceiling height is 9.84 feet and also provides the floor level. The final output confirms that the building is compliant with this rule.

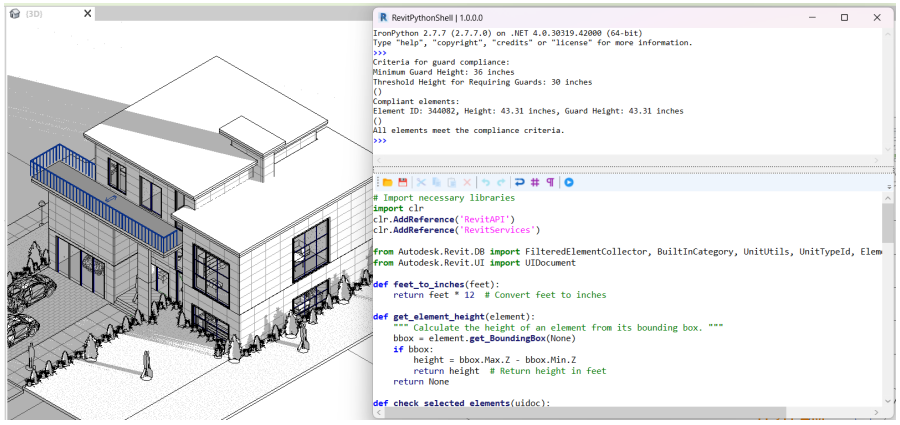

Figure 7. Python script in Revit verifying guardrail height compliance with IRC 2021.

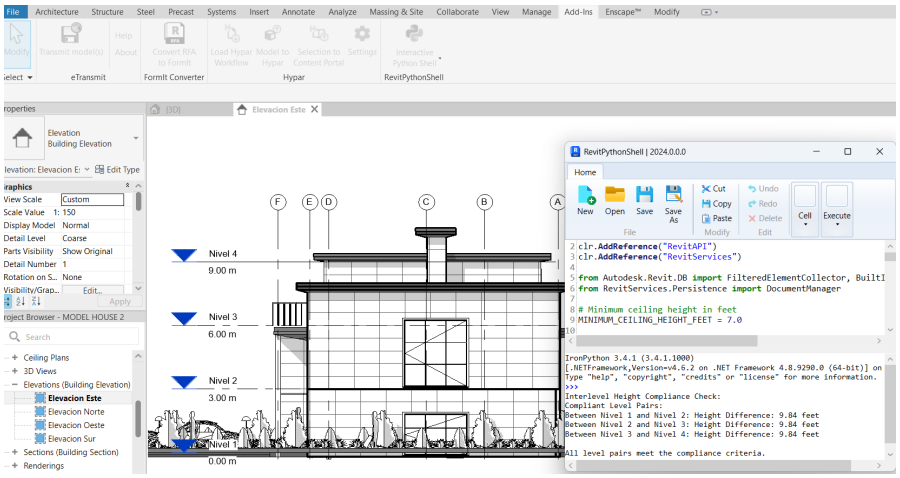

Figure 8. Python script validating ceiling height compliance in Revit.

Figure 9 shows the results for Rule 10, which outlines material specifications and specifies the required thickness and span rating for these panels. It identifies the room boundary, flooring material, floor level, Element ID, and other details essential for ensuring compliance with floor specifications.

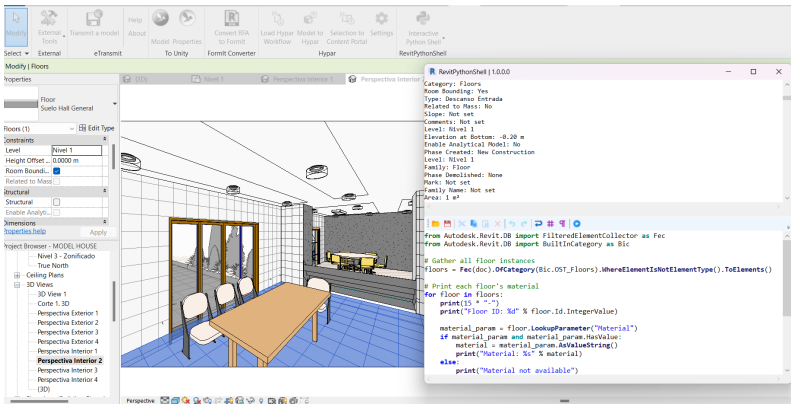

Figure 9. Process of performing compliance check for the flooring material in Revit.

Figure 10 shows the compliance output for Rule 8. The Python script interacts with Revit to detect all toilet fixtures and their properties. It verifies the fixture type, placement coordinates, and spatial relationships with nearby items. The output displays toilet fixtures and their components, including water closets (W.C.) and washbasins, along with their design specifications.

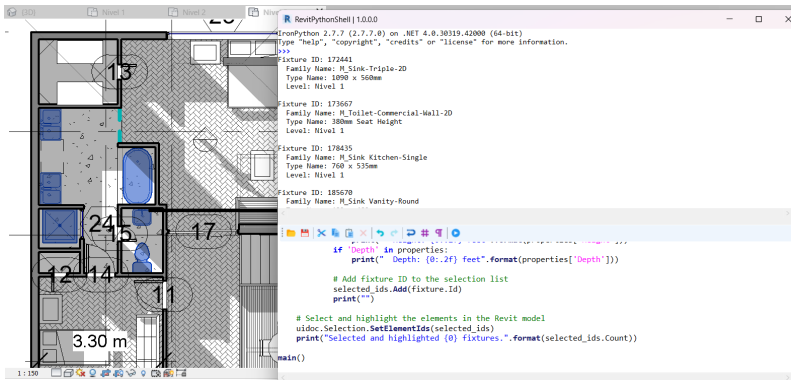

Figure 10. Python script in Revit identifying specific toilet fixtures, detailing their attributes.

Figure 11 shows the compliance verification output based on Rule 9. In compliance verification, it is critical to check for the presence of required elements in designated spaces. In compliance verification, it is critical to check for the presence of required elements in designated spaces. This process facilitates such checks. The results indicate that the sink is missing from the kitchen, resulting in a non-compliant status. Moreover, the output provides details such as the room number or name, depending on the parameters available in Revit.

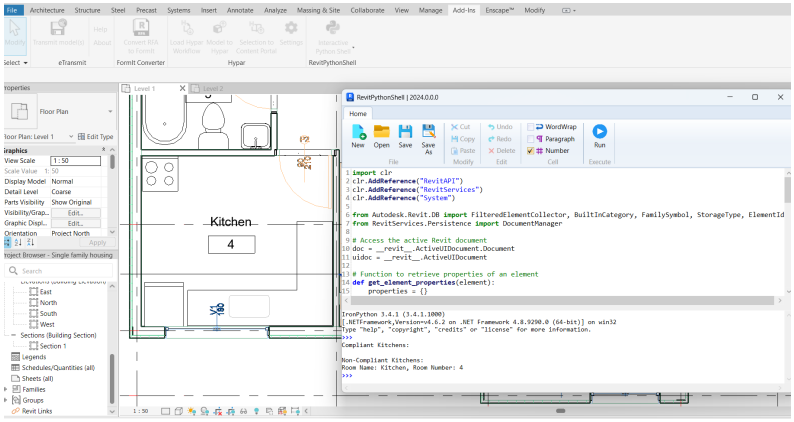

Figure 11. Kitchen compliance check in Revit for required components.

4.4. Case Study 2

In the second case study, we apply the proposed methodology to the office building model. Figure 12 illustrates the office building model with the selected eight rules from Table 1. While the other rules remain consistent with those in Case Study 1, Rule 11 assesses compliance with the requirement that the edge-to-edge distance between any two footings must be at least equal to the width of the larger footing. Rule 12 defines the minimum outdoor air ventilation rate required for occupied indoor spaces, specifically for office spaces (business occupancy) in accordance with the International Mechanical Code (IMC). The script first identifies all rooms in the model and then collects all air terminals (such as diffusers). It iterates through each diffuser, checking if it has a flow parameter (CFM). If the flow data are available, the script assigns that diffuser's flow to the corresponding room. After calculating the airflow for each room, the script verifies if it meets the area-based requirement (0.06 CFM per square foot of floor area). If a room's airflow is insufficient, it is marked as non-compliant.

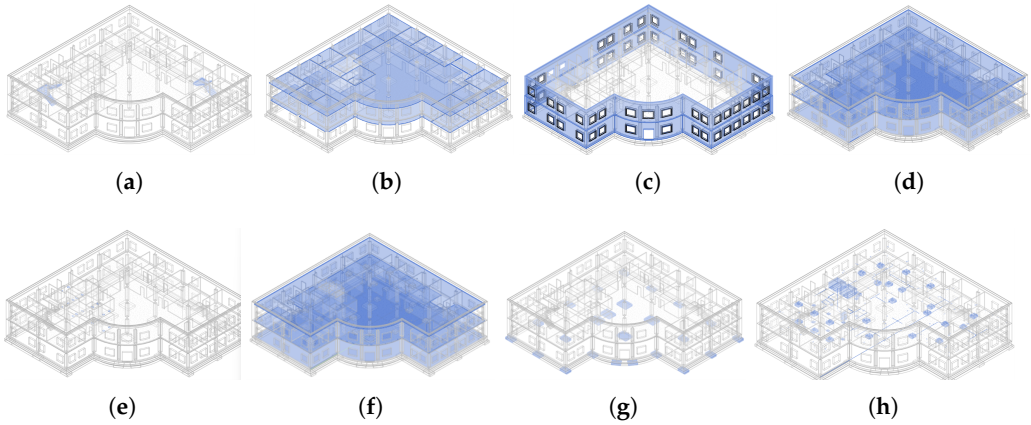

Figure 12. Case Study 2 model. (**a**) Rule 2: stairways; (**b**) Rule 4: habitable spaces; (**c**) Rule 5: windows-to-wall ratio; (**d**) Rule 6: floor area; (**e**) Rule 8: toilet; (**f**) Rule 10: floor; (**g**) Rule 11: edge-to-edge distance; (**h**) Rule 12: ventilation.

4.5. Generating Compliance Reports Using LLMs' API

This study also utilized LLMs' API to generate compliance reports, as shown in Figure 13, for BIM models by analyzing the output from Revit and providing recommendations for elements that did not meet the standards. The API key was obtained, and the necessary libraries were installed to facilitate the setup. A Python script was developed to extract BIM data. These reports include summaries of key issues, descriptions of non-compliant elements, corrective recommendations, and their impacts on safety and functionality. Figure 13 shows a sample prompt and the report generated for Rule 1.

Prompt: *Analyze the door widths for compliance with building standards and provide recommendations for any non-compliant elements.*

Report: *Based on the data provided on door widths, most of the doors have a width of 0.90 m, meeting standard building requirements. However, there are three doors with a width of 0.70 m, which do not meet the typical minimum standard for door widths in buildings. To improve compliance with building standards, the doors with a width of 0.78m should be replaced or widened to meet the requirements. Ensuring that all doors in a building meet standard width requirements is essential for accessibility and safety reasons. Regular audits and measurements can help to identify and address any non-compliant doors in a timely manner.*

Figure 13. Sample prompt and compliance report for Rule 1.

4.6. Comparison Between Different Large Language Models

Table 4 presents the performance of various LLMs using three evaluation metrics:

- Processing Time: The duration (in seconds) required by each LLM to generate Python scripts, as reported by the models themselves.
- Correction Attempts: The number of attempts needed for each LLM to produce a functional script without errors. This metric indicates the level of iterative adjustments required.
- Status: A binary indicator of whether the LLM successfully generated a functional script on any of its attempts.

Moreover, we introduced a quantitative metric, success rate, calculated using the following formula:

$$\text{Success Rate (\%)} = \frac{\text{Number of Correctly Executed Scripts}}{\text{Number of Attempts}} \times 100 \quad (1)$$

This formula measures the LLM's efficiency in producing functional, error-free Python scripts relative to the number of correction attempts made. A higher success rate indicates that the model required fewer attempts to generate a working script.

Eight LLMs were evaluated: ChatGPT 4.0 ($30/$60 per 1M tokens), Claude Sonnet 3.5 ($3/$15 per 1M tokens), Meta LLaMA 3.1-405B ($3.50/$3.50 per 1M tokens), Microsoft Copilot (subscription-based, API price not public), Gemini ($1.25/$10 per 1M tokens), Perplexity.AI ($3/$15 per 1M tokens), Grok ($3/$15 per 1M tokens), and Deepseek ($0.55/$2.19 per 1M tokens). Meta LLaMA 3.1-405B and Microsoft Copilot failed to generate any working scripts. Gemini and Perplexity.AI were relatively fast but produced incorrect scripts for several rules, with success rates below 15%. Deepseek, the lowest-cost option, achieved a moderate 33% average success rate with few correction attempts. ChatGPT 4.0 and Claude Sonnet 3.5 completed all rule checks; Claude required fewer iterations (3.4 versus 6.8) and delivered a higher average success rate (23.7%) at roughly one-tenth of ChatGPT's token cost. Grok delivered the best overall performance, reaching a 76.7% success rate with virtually no retries while maintaining a mid-range price point.

Table 4. Performance capabilities of various LLMs. Success rate is calculated using Equation (1).

Model/LLM (API Price)	Metric	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Avg Metrics
ChatGPT 4.0 (USD 30/USD 60 per 1M tokens)	Processing Time (s)	2.0	2.0	2.0	2.0	2.0	2.0
	Correction Attempts	6	7	9	7	5	6.8
	Status	✓	✓	✓	✓	✓	
	Success Rate (%)	14.3	12.5	10.0	12.5	16.7	13.2
Claude Sonnet 3.5 (USD 3/USD 15 per 1M tokens)	Processing Time (s)	1.5	1.5	1.5	1.5	1.5	1.5
	Correction Attempts	4	2	3	4	4	3.4
	Status	✓	✓	✓	✓	✓	
	Success Rate (%)	20.0	33.3	25.0	20.0	20.0	23.7
Meta LLaMA 3.1-405B (USD 3.50/USD 3.50 per 1M tokens)	Processing Time (s)	-	-	-	-	-	Unknown
	Correction Attempts	-	-	-	-	-	-
	Status	✗	✗	✗	✗	✗	
	Success Rate (%)	0	0	0	0	0	0
Microsoft Copilot (N/A)	Processing Time (s)	-	-	-	-	-	Unknown
	Correction Attempts	-	-	-	-	-	-
	Status	✗	✗	✗	✗	✗	
	Success Rate (%)	0	0	0	0	0	0
Gemini (USD 1.25/USD 10 per 1M tokens)	Processing Time (s)	1.5	1.5	-	1.5	-	1.5
	Correction Attempts	3	7	-	8	-	6
	Status	✓	✓	✗	✓	✗	
	Success Rate (%)	25.0	12.5	0	11.1	0	9.7
Perplexity.AI (USD 3/USD 15 per 1M tokens)	Processing Time (s)	3.0	-	3.0	3.0	3.0	3.0
	Correction Attempts	3	-	4	7	6	5.0
	Status	✓	✗	✓	✓	✓	
	Success Rate (%)	25.0	0	20.0	12.5	14.3	14.4
Grok (USD 3/USD 15 per 1M tokens)	Processing Time (s)	1.5	1.5	1.5	1.5	1.5	1.5
	Correction Attempts	0	0	2	0	1	0.6
	Status	✓	✓	✓	✓	✓	
	Success Rate (%)	100	100	33.3	100	50	76.7
Deepseek (USD 0.55/USD 2.19 per 1M tokens)	Processing Time (s)	3.0	6.0	3.0	3.0	4.0	3.8
	Correction Attempts	1	3	2	2	3	2.2
	Status	✓	✓	✓	✓	✓	
	Success Rate (%)	50.0	25.0	33.3	33.3	25.0	33.3

5. Conclusions

In this study, a generative AI-based framework was developed to assist construction professionals in evaluating projects for compliance with regulations. The framework leverages Large Language Models (LLMs) to generate Python scripts that can be executed within a Building Information Modeling (BIM) environment. Its effectiveness and reliability were demonstrated through a case study involving a residential building assessed against

the IRC building regulations. The framework successfully identified non-compliance issues by flagging problems in the BIM model.

The performance of various LLMs was evaluated in terms of efficiency, speed, and accuracy. Among the models tested, ChatGPT 4.0 and Claude Sonnet 3.5 emerged as the most effective, while Meta Llama 3.1-405B and Microsoft Copilot were unable to generate successful Python scripts, despite multiple attempts.

This study highlights the importance of carefully designing prompts to enable LLMs to generate Python scripts that can execute in Revit without errors. An optimized prompt structure was identified, which includes components such as a basic prompt, rule descriptions, general instructions, and rule-specific instructions. In addition, incorporating a functional script for one specific rule in the prompt was found to aid in generating scripts for other rules.

It was found that the framework was most effective when applied to well-structured BIM models with accurate room tags, boundaries, and naming conventions, as it relies heavily on Revit model properties for verification. Compatibility between software versions also proved crucial; for instance, ensuring that Revit 2024 and PythonShell 2024 are used together is necessary for proper functionality.

The proposed tool offers significant scientific and technical contributions to the field of construction compliance checking. First, it introduces a novel framework that can be effectively utilized during the pre-construction phase to identify and resolve compliance issues early, reducing risks and costs. Second, the tool's design allows for scalability, making it adaptable to a wide range of construction projects, which demonstrates its versatility and broad applicability. Third, the framework is extendable to incorporate additional structural design regulations, showcasing its flexibility and potential for future enhancements. A key technical innovation is its adaptability to changes in building regulations. Finally, the tool significantly reduces the time required for compliance checking.

This research has a few limitations that were identified during the case study: (1) it was tested only on two case studies and evaluated by only 12 rules. The optimized prompt structure may require further improvements when tested on a larger set of rules. (2) The testing was conducted on a limited number of Revit versions, which may not fully represent all possible scenarios. (3) The field of LLMs is rapidly evolving, and newer models in the future might perform better than those tested in this study. (4) Although the framework utilizes LLMs to generate Python scripts, the current workflow requires manual intervention for script execution within Revit, as well as for correcting any errors that occur. One potential approach to achieve full automation is the integration of an LLM agent capable of directly interacting with the Revit API. Such an agent could generate, execute, and debug Python scripts in real-time, eliminating the need for manual script transfer. The LLM agent could be designed to receive feedback from Revit, refine the code, and ensure that all rules are correctly applied without user intervention. Future research should explore this approach to further enhance the framework's automation capabilities. (5) Another limitation of the proposed framework is its reliance on well-structured BIM models with precise metadata. In real-world collaborative environments, BIM models may often contain incomplete, inconsistent, or outdated information, which can negatively impact the framework's ability to generate accurate compliance checking scripts. Errors or missing metadata can lead to inaccurate interpretations of the model, reducing the effectiveness of automated compliance verification. This dependency on high-quality BIM data may limit the framework's applicability in projects where model quality is inconsistent. Future research should focus on enhancing the framework's robustness to handle incomplete or inconsistent BIM data. One potential approach is to integrate data validation techniques that automatically identify and correct data gaps within BIM models before initiating compliance checking.

Moreover, using LLMs to detect and suggest corrections for missing or erroneous metadata can further improve the framework's resilience. (6) One limitation of the proposed framework is its dependency on the Revit platform, specifically leveraging the Revit API and its unique BIM data structure to automate compliance checking. While this approach is highly effective within Revit, its applicability is currently restricted to this platform. However, the method has the potential to be adapted for other design and modeling tools that support Python scripting and API calls. For example, with appropriate modifications to the script generation component, it could be extended to AutoCAD (for 2D/3D design), ArchiCAD (for BIM), Navisworks (for model coordination), or Rhino/Grasshopper (for parametric design). Future research will explore the applicability of this approach to other platforms to enhance its versatility and practicality.

Author Contributions: Conceptualization, S.M. and L.G.; methodology, S.M.; software, S.M.; validation, S.M., L.G. and Z.U.D.; formal analysis, S.M.; investigation, S.M.; resources, K.K.; data curation, S.M.; writing, L.G., Z.U.D., K.K., A.S., Z.H. and Y.Z.; visualization, S.M.; supervision, L.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available upon request from the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Zhang, J.; Cui, B.; Gao, Y.; Zhang, D. Factors Influencing the Acceptance of BIM-Based Automated Code Compliance Checking in the AEC Industry in China. *J. Manag. Eng.* **2023**, *39*, 04023036. [CrossRef]
2. Aydın, M. Building Information Modeling Based Automated Building Regulation Compliance Checking Asp. net Web Software. *Intell. Autom. Soft Comput.* **2021**, *28*, 11–25. [CrossRef]
3. Ismail, A.S.; Ali, K.N.; Iahad, N.A. A review on BIM-based automated code compliance checking system. In Proceedings of the 2017 International Conference on Research and Innovation in Information Systems (ICRIIS), Langkawi, Malaysia, 16–17 July 2017; pp. 1–6.
4. Preidel, C.; Borrmann, A. Automated code compliance checking based on a visual language and building information modeling. In Proceedings of the 32nd ISARC 2015, Oulu, Finland, 15–18 June 2015.
5. Chen, N.; Lin, X.; Jiang, H.; An, Y. Automated building information modeling compliance check through a large language model combined with deep learning and ontology. *Buildings* **2024**, *14*, 1983. [CrossRef]
6. Wang, Y.; Liu, Y.; Cai, H.; Wang, J.; Zhou, X. An automated fire code compliance checking jointly using building information models and natural language processing. *Fire* **2023**, *6*, 358. [CrossRef]
7. Yogana, E.; Latief, Y. Development of system information of building code checking in planning and permitting phase to improve building code compliance based on work breakdown structure (WBS) using building information modeling (BIM). *J. Phys. Conf. Ser.* **2021**, *1858*, 012091. [CrossRef]
8. Bus, N.; Roxin, A.; Picinbono, G.; Fahad, M. Towards french smart building code: Compliance checking based on semantic rules. *arXiv* **2019**, arXiv:1910.00334.
9. Li, S.; Wang, J.; Xu, Z. Automated compliance checking for BIM models based on Chinese-NLP and knowledge graph: An integrative conceptual framework. *Eng. Constr. Archit. Manag.* 2024, *ahead of print*. [CrossRef]
10. Lange, M.; Kirkham, R.; Tannert, B. Strategically Using Applied Machine Learning for Accessibility Documentation in the Built Environment. In Proceedings of the Human-Computer Interaction–INTERACT 2021: 18th IFIP TC 13 International Conference, Bari, Italy, 30 August–3 September 2021; Proceedings, Part II 18; Springer: Berlin/Heidelberg, Germany, 2021; pp. 426–448.
11. Nawari, N.O.; Ravindran, S. Blockchain and building information modeling (BIM): Review and applications in post-disaster recovery. *Buildings* **2019**, *9*, 149. [CrossRef]
12. Preidel, C.; Borrmann, A. Refinement of the visual code checking language for an automated checking of building information models regarding applicable regulations. In Proceedings of the ASCE International Workshop on Computing in Civil Engineering 2017, Seattle, WA, USA, 25–27 June 2017; pp. 157–165.
13. Zhao, X.; Huang, L.; Sun, Z.; Fan, X.; Zhang, M. Compliance Checking on Topological Spatial Relationships of Building Elements Based on Building Information Models and Ontology. *Sustainability* **2023**, *15*, 10901. [CrossRef]

14. Bloch, T.; Sacks, R. Clustering information types for semantic enrichment of building information models to support automated code compliance checking. *J. Comput. Civ. Eng.* **2020**, *34*, 04020040. [CrossRef]
15. de Marco, G.; Slongo, C.; Siegele, D. Enriching Building Information Modeling Models through Information Delivery Specification. *Buildings* **2024**, *14*, 2206. [CrossRef]
16. Lee, J.K.; Cho, K.; Choi, H.; Choi, S.; Kim, S.; Cha, S.H. High-level implementable methods for automated building code compliance checking. *Dev. Built Environ.* **2023**, *15*, 100174. [CrossRef]
17. Zhang, J. How can ChatGPT help in automated building code compliance checking? In Proceedings of the 40th ISARC, Chennai, India, 5–7 July 2023; pp. 63–70.
18. Fuchs, S.; Witbrock, M.; Dimyadi, J.; Amor, R. Using large language models for the interpretation of building regulations. *arXiv* **2024**, arXiv:2407.21060.
19. Borrmann, A.; König, M.; Koch, C.; Beetz, J. *Building Information Modeling: Why? What? How?*; Springer: Berlin/Heidelberg, Germany, 2018.
20. Nguyen, T.H.; Kim, J.L. Building code compliance checking using BIM technology. In Proceedings of the 2011 Winter Simulation Conference (WSC), Phoenix, AZ, USA, 11–14 December 2011; pp. 3395–3400.
21. Lee, Y.; Eastman, C.; Lee, J. Automated rule-based checking for the validation of accessibility and visibility of a building information model. In Proceedings of the 2015 International Workshop on Computing in Civil Engineering 2015, Austin, TX, USA, 21–23 June 2015; pp. 572–579.
22. Anderson, A. Using Building Information Modeling to Transform the Building Codes Compliance Process. In Proceedings of the Construction Research Congress 2020, Tempe, AZ, USA, 8–10 March 2020; American Society of Civil Engineers: Reston, VA, USA, 2020; pp. 1048–1056.
23. Pezeshki, Z.; Ivari, S.A.S. Applications of BIM: A brief review and future outline. *Arch. Comput. Methods Eng.* **2018**, *25*, 273–312. [CrossRef]
24. Azhar, S. Building information modeling (BIM): Trends, benefits, risks, and challenges for the AEC industry. *Leadersh. Manag. Eng.* **2011**, *11*, 241–252. [CrossRef]
25. Succar, B. Building information modelling maturity matrix. In *Handbook of Research on Building Information Modeling and Construction Informatics: Concepts and Technologies*; IGI Global: Hershey, PA, USA, 2010; pp. 65–103.
26. Zhu, S.; Zhou, J.; Cheng, L.; Fu, X.; Wang, Y.; Dai, K. Research on a BIM Model Quality Compliance Checking Method Based on a Knowledge Graph. *J. Comput. Civ. Eng.* **2025**, *39*, 04024049. [CrossRef]
27. Salama, D.; El-Gohary, N. Semantic modeling for automated compliance checking. In Proceedings of the 2011 ASCE International Workshop on Computing in Civil Engineering (2011), Miami, FL, USA, 19–22 June 2011; pp. 641–648.
28. Zhang, J.; El-Gohary, N.M. Automated information transformation for automated regulatory compliance checking in construction. *J. Comput. Civ. Eng.* **2015**, *29*, B4015001. [CrossRef]
29. Zhong, B.; Gan, C.; Luo, H.; Xing, X. Ontology-based framework for building environmental monitoring and compliance checking under BIM environment. *Build. Environ.* **2018**, *141*, 127–142. [CrossRef]
30. Andrich, W.; Daniotti, B.; Pavan, A.; Mirarchi, C. Check and validation of building information models in detailed design phase: A check flow to pave the way for BIM based renovation and construction processes. *Buildings* **2022**, *12*, 154. [CrossRef]
31. Altıntaş, Y.D.; İlal, M.E. Integrating building and context information for automated zoning code checking: A review. *J. Inf. Technol. Constr.* **2022**, *27*, 548–570.
32. Fan, S.L.; Chi, H.L.; Pan, P.Q. Rule checking Interface development between building information model and end user. *Autom. Constr.* **2019**, *105*, 102842. [CrossRef]
33. UpCodes. UpCodes Official Website. 2025. Available online: https://up.codes (accessed on 6 May 2025).
34. SMARTreview. SMARTreview APR–Automated Plan Review for Revit. 2025. Available online: https://smartreview.biz/apr_learn_more (accessed on 13 May 2025).
35. Solibri. Solibri–Intelligent Model Checking for BIM. 2025. Available online: https://www.solibri.com/intelligent-model-checking (accessed on 13 May 2025).
36. Autodesk. Autodesk Model Checker for Revit. 2025. Available online: https://interoperability.autodesk.com/modelchecker.php (accessed on 13 May 2025).
37. Bureau Veritas. icheck for Building–Accessibility [CBC]. 2025. Available online: https://apps.autodesk.com/RVT/en/Detail/Index?appLang=en&id=1041774428548349665&os=Win64 (accessed on 13 May 2025).
38. EvolveLAB. Revit Code Tools. 2025. Available online: https://www.evolvelab.io/product-page/revit-code-tools (accessed on 13 May 2025).
39. Lee, Y.C.; Eastman, C.M.; Lee, J.K. Validations for ensuring the interoperability of data exchange of a building information model. *Autom. Constr.* **2015**, *58*, 176–195. [CrossRef]
40. Minaee, S.; Mikolov, T.; Nikzad, N.; Chenaghlu, M.; Socher, R.; Amatriain, X.; Gao, J. Large language models: A survey. *arXiv* **2024**, arXiv:2402.06196.

41. Jang, S.; Lee, G. Interactive Design by Integrating a Large Pre-Trained Language Model and Building Information Modeling. In Proceedings of the ASCE International Conference on Computing in Civil Engineering 2023, Corvallis, OG, USA, 25–28 June 2022; pp. 291–299.
42. He, Z.; Wang, Y.H.; Zhang, J. Generative AIBIM: An automatic and intelligent structural design pipeline integrating BIM and generative AI. *Inf. Fusion* **2025**, *114*, 102654. [CrossRef]
43. International Code Council (ICC). *International Residential Code (IRC) 2021*; International Code Council: Washington, DC, USA, 2021.
44. International Code Council (ICC). *International Mechanical Code (IMC) 2021*; International Code Council: Washington, DC, USA, 2021.
45. Bowles, J.E.; Guo, Y. *Foundation Analysis and Design*; McGraw-Hill: New York, NY, USA, 1996; Volume 5.