CCC 2025 **Proceedings of the Creative Construction Conference** (2025)
Edited by: Miroslaw J. Skibniewski & Miklós Hajdu & Žiga Turk
https://doi.org/10.22260/CCC2025/0036

RETRIEVAL-AUGMENTED CODE EXECUTION FRAMEWORK FOR AUTOMATED STRUCTURAL CALCULATION OF PRECAST CONCRETE FLOOR PANELS

Minwoo Jeong, Taegeon Kim, Seokhwan Kim, Kichang Choi, Seungwon Baek,
Hongjo Kim
Yonsei University, Seoul, South Korea, minuj9855@yonsei.ac.kr

Abstract

Large language models (LLMs) have recently gained attention in construction engineering for their ability to interpret technical documents, extract domain-specific information, and support natural language interfaces. To mitigate hallucination and improve factual reliability, retrieval-augmented generation (RAG) techniques have been widely adopted, enabling LLMs to ground their responses in external reference documents such as design codes. However, current RAG-based applications still face challenges in performing accurate structural calculations, particularly when precise mathematical reasoning and strict code compliance are required. To address these limitations, this paper proposes a Retrieval-Augmented Generation-based framework for automated structural calculations, integrating semantic retrieval with code-based reasoning and self-verification. The proposed system consists of a vector database that stores and retrieves relevant design code provisions and structural equations, and a prompt-driven LLM that generates executable Python code along with human-readable reasoning. Furthermore, a self-verification mechanism is incorporated to ensure both logical consistency with design standards and numerical accuracy of the computed results. The framework was evaluated on precast concrete floor slab design scenarios, demonstrating that the integration of code execution (CE) and self-verification (SV) significantly improves the reliability of automated structural verification workflows. These findings highlight the potential of AI-driven approaches in supporting digital construction technologies and enhancing real-time decision-making in structural engineering.

Keywords: large language models, prompt engineering, retrieval-augmented generation, self-verification, structural calculation.

Peer-review under responsibility of the scientific committee of the Creative Construction Conference 2025.

1. Introduction

In construction projects, unforeseen site conditions often necessitate design alterations during the execution phase, leading to workflow disruptions and project delays. These changes are often triggered by incompatibility between drawings and site conditions, or unplanned scope revisions. When such changes occur on-site, the typical industry practice is for the contractor's field team to submit a formal modification request to the structural design office, which then performs structural analysis, checks code compliance, and provides feedback. This manual, iterative process not only slows down decision-making but also increases overall project duration and cost.

Frequent design changes require repeated compliance checks against complex design codes, leading to significant delays and additional costs. Structural engineers must reinterpret drawings, rerun calculations, and verify safety and serviceability requirements, often under time pressure during critical phases such as precast slab installation. The lack of rapid, automated tools for structural verification can further disrupt construction workflows. The automation of structural design review has long been a focus in construction engineering. Traditional rule-based systems encode design criteria into deterministic logic but often lack flexibility in handling real-world complexities [1], [2]. Recent advancements in artificial intelligence, particularly large language models (LLMs) such as generative

Corresponding author email address: hongjo@yonsei.ac.kr

pretrained transformers (GPT) [3], have shown promise in automating document interpretation and code compliance checks [4], [5], [6]. However, standalone LLMs are prone to generating incorrect or unsubstantiated responses, a phenomenon known as “hallucination” [7], which limits their reliability in safety-critical domains like structural engineering.

To address these limitations, the Retrieval-Augmented Generation (RAG) framework has been introduced [8], allowing LLMs to ground their responses in authoritative documents such as building codes and design manuals [9]. While RAG improves factual accuracy, existing RAG-based applications in structural engineering are mostly limited to document retrieval or summarization, lacking the ability to perform precise structural calculations that require mathematical reasoning and strict code compliance. A key challenge is that current LLMs are not inherently equipped to execute the rigorous numerical computations essential for structural design [10]. These tasks demand high precision and contextual awareness of engineering standards. Moreover, there is a lack of real-time automated systems that can both interpret design criteria and perform verifiable structural calculations.

To address this research gap, this study proposes a Retrieval-Augmented Generation-based framework that integrates semantic retrieval with code-based reasoning. The system retrieves relevant design provisions, and a prompt-driven LLM generates executable Python code to perform accurate structural calculations aligned with engineering standards. The proposed method is evaluated using precast concrete floor slab scenarios, demonstrating its effectiveness in automating structural verification tasks and supporting real-time decision-making in construction environments. This approach contributes to advancing AI-driven automation in structural engineering and provides a scalable foundation for integration with Building Information Modeling (BIM) and digital twin technologies, enabling responsive and intelligent design verification.

2. Related works

2.1. Integrating RAG with LLMs in construction

Retrieval-Augmented Generation (RAG) has been developed to address the challenge of hallucination in LLM outputs by anchoring generated content to verified external sources. This technique combines a language model with a retrieval mechanism that pulls relevant information from structured knowledge bases. By integrating retrieved context into the response generation process, RAG enables more factually grounded outputs based on established references, including building codes and safety standards.

Within the construction domain, RAG has been applied to several data-intensive tasks such as document querying, summarizing safety protocols, and analyzing contractual risks. For example, Wu et al. [11] presented the RAG4CM framework, which utilizes hierarchical document parsing and user-preference learning to enhance retrieval accuracy across multiple data sources, yielding improved performance over conventional systems. Uhm et al. [12] demonstrated that RAG-GPT, when evaluated by practitioners and field personnel, outperformed standard LLMs in delivering accurate and context-aware safety information, making it a valuable tool for safety management and training. Additionally, Tran et al. [13] developed a system known as the Construction Safety Query Assistant (CSQA), combining document interpretation with interactive feedback to streamline access to safety regulations.

Beyond safety-related applications, RAG-based systems have shown promise in contract analysis. Shuai et al. [14] introduced C3RAG, which draws from legal precedent databases to offer context-sensitive evaluations of contract risks in construction, demonstrating strong results in terms of citation relevance and clarity. In the field of geotechnical engineering, Qian et al. [15] designed a multi-hop RAG solution that supports retrieval of site-specific regulations and geological data, providing real-time assistance for planning tasks.

These examples reflect the expanding role of RAG in delivering contextually precise and factually grounded outputs for construction-related tasks. However, most applications have been limited to textual understanding and retrieval. There remains a notable gap in leveraging RAG for quantitative tasks such as structural calculations, where both semantic comprehension and mathematical accuracy are critical. Few studies have explored LLM-driven automation in this area, particularly those that involve not only

retrieving and interpreting structural formulas but also executing them within context. This reveals a significant opportunity to extend RAG methodologies into the realm of numerical design verification in civil engineering.

2.2. Code-based approaches for improving LLM mathematical reasoning

Enhancing the mathematical reasoning capabilities of large language models (LLMs) has been a significant area of focus, particularly through the refinement of prompt strategies, the integration of external computational tools, and targeted fine-tuning. One of the most influential techniques in this domain is Chain-of-Thought (CoT) prompting, which promotes a step-by-step breakdown of complex problems, allowing the model to process intermediate reasoning stages more effectively. This method, introduced by Wei et al. [16], has notably improved LLM performance in tasks involving multi-step calculations by making the reasoning process explicit and transparent. Expanding on this concept, Self-Consistency [17] proposes generating multiple reasoning trajectories and selecting the most coherent or consistent output, thereby boosting the robustness of the model's conclusions.

Recognizing the inherent limitations of LLMs in performing precise numerical computations, researchers have explored mechanisms for external code-based verification. Zhou et al. [18] introduced a framework where GPT-4 generates Python code snippets as part of its response, enabling verification of its own outputs through actual execution. This method effectively bridges the gap between natural language reasoning and formal numerical validation. Another promising direction has involved leveraging external computing environments, such as the Python Read-Evaluate-Print Loop (REPL). Yamauchi et al. [19] demonstrated that models like ChatGPT can enhance their accuracy by generating code that is executed externally, reducing reliance on the model’s internal, and sometimes unreliable, arithmetic functions. This approach, akin to the "Toolformer" paradigm, clearly delineates reasoning from computation, minimizing hallucinations and improving factual correctness.

Building further on CoT, Program-of-Thought (PoT) prompting, introduced by Chen et al. [20], explicitly formulates reasoning as a programmatic structure. Instead of relying on internal computations, PoT prompts the model to develop a logical sequence that is then externally executed. This clear division between logic formulation and computational execution has shown substantial promise, especially in tasks requiring formal verification. Collectively, these methodologies have established a strong foundation for advancing LLM-based mathematical reasoning. However, most approaches remain constrained by limitations in consistency, error resilience, and comprehensive validation, particularly when tackling complex multi-step computations that require both symbolic reasoning and precise numerical execution.

2.3. Research Gap

Although numerous studies have demonstrated the capabilities of LLMs in tasks such as document interpretation, safety management, and regulatory text analysis within the construction sector, most of these works have concentrated on symbolic reasoning or information retrieval rather than numerical computation. While RAG-based methods have advanced the factual grounding of language models, their use in structural design verification remains fundamentally limited, particularly when accurate mathematical execution and domain-specific compliance are required.

Most current approaches depend on translating rules into logic-based systems or interfacing with external computational tools, falling short of delivering a fully integrated solution where design interpretation, formula retrieval, and direct computational execution are managed within a unified framework. Furthermore, many of these methods stop at retrieving and summarizing design standards, without extending into automated calculation or result validation based on those standards.

To the best of the authors’ knowledge, this study is among the first to tackle the automation of structural calculation tasks through a Retrieval-Augmented Generation framework combined with code-based reasoning and self-verification mechanisms. By enabling language models to not only understand and retrieve design requirements but also autonomously perform and validate structural computations, this research addresses a critical gap in current AI applications in civil engineering. The proposed approach demonstrates how integrating retrieval with executable code generation can improve both the accuracy

and reliability of automated structural verification, offering a new direction for AI-driven design validation systems.

3. Proposed method

3.1. Overview of the proposed framework

Despite recent advancements in applying LLMs to various engineering problems, structural calculations in practice are still largely conducted through manual, spreadsheet-based processes. Engineers are required to interpret complex design codes, identify and apply appropriate formulas, and repeatedly validate their results for compliance, often under tight deadlines and with high demands for accuracy. These procedures are repetitive, prone to error, and mentally taxing, especially when dealing with voluminous and intricate regulatory documents. While LLMs have shown strong capabilities in processing natural language, their application to tasks involving precise numerical validation and strict adherence to engineering standards remains limited.

This study introduces an AI-driven framework intended to automate structural calculations, focusing on the design and verification of precast concrete floor panels. To replicate real-world workflows, we structured the system around typical spreadsheet-based calculation sequences, using multi-turn queries that mirror standard verification procedures. As depicted in Fig. 1, the framework consists of two primary modules: (1) a RAG-based retrieval system that builds a vectorized database from design standards and structural formulas, enabling the retrieval of relevant provisions; and (2) a code-generating LLM component that outputs executable Python code capable of performing the required calculations. The system is designed to provide rapid, regulation-compliant structural analysis, and while this study focuses on precast panels, the architecture is flexible enough to accommodate other structural applications.

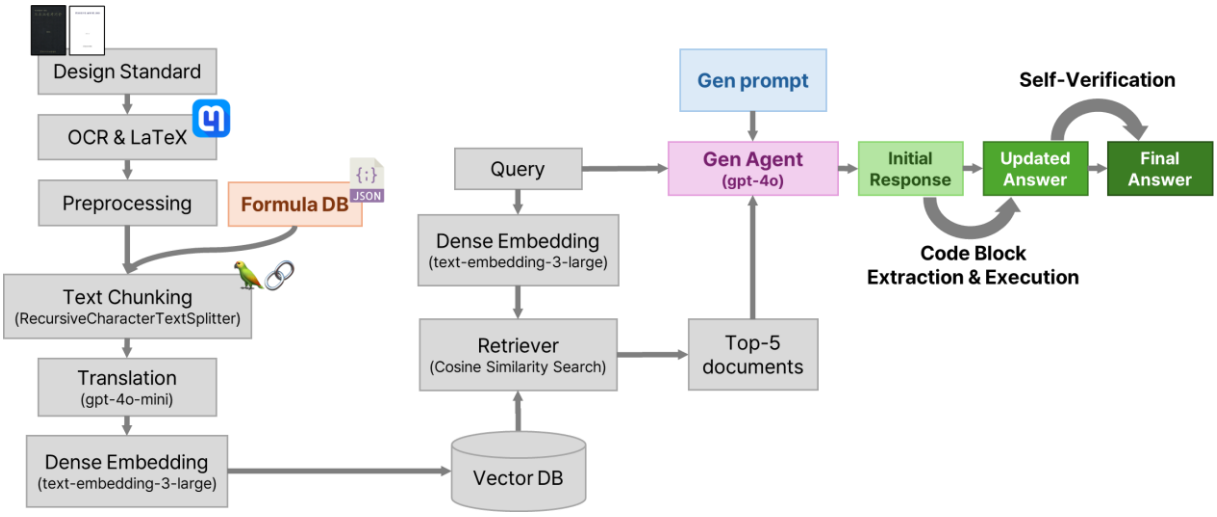

Fig. 1. System architecture of the proposed method.

3.2. RAG-based structural calculation framework

The foundation of the proposed method lies in a semantic retrieval pipeline, which extracts pertinent design provisions and associated formulas from standard code documents. This enables LLM to access precise engineering references necessary for accurate calculations. Domain-specific documents, including the Highway Bridge Design Standard (2010) and the Korea Structural Concrete Design Code (2003), were digitized through Mathpix OCR [21], producing LaTeX-encoded content suitable for downstream processing.

To enhance retrieval efficiency, preprocessing steps were applied to ensure that formulas and their variable definitions were co-located within the same document segments. Frequently, variables were defined separately from their equations in the original standards; these were reorganized so that all relevant details—symbols, units, and descriptions—appeared together, reducing ambiguity during retrieval and interpretation.

In addition to the code documents, a custom formula database was constructed in JSON format to capture structural equations not directly stated in the standards. Each entry in this database included not only the mathematical expressions but also comprehensive metadata such as units and explanatory notes. These entries were segmented into coherent text chunks using LangChain utilities [22], translated into English via gpt-4o-mini-2024-07-18 [23], and embedded using OpenAI's text-embedding-3-large model [24]. The final vector database stored both regulatory text and formula data, enabling dense retrieval via cosine similarity.

Upon receiving a structural query, the system retrieves the top-k most semantically relevant document chunks. Unlike traditional keyword-based retrieval, this dense method matches query intent with document meaning, ensuring higher relevance even when terminology varies [25]. The retrieved information, along with the user's question and contextual data from previous steps, forms a structured prompt for the LLM.

3.3. LLM-based code generation and execution

Once the relevant provisions and formulas have been retrieved, the LLM generates a detailed structural calculation response. Each structured prompt includes the user query, the retrieved design code content, and a generation template that specifies initial values, cumulative variables, and contextual assumptions such as loading conditions and material properties.

The prompt instructs the LLM to reference the retrieved data when selecting formulas and to define each variable explicitly, including units. The model is guided to produce a Python code block that performs the complete calculation, ending with a print() statement that outputs the result. The response format is standardized for clarity and to ensure consistency across different queries.

The generated code is then executed outside the model, and the numerical result is appended to the final response. This execution step ensures that the LLM's reasoning aligns with actual computational outputs, improving the accuracy and reliability of the structural calculations. By combining code generation with real-time execution, the framework achieves a robust and repeatable method for verifying complex engineering computations.

3.4. Self-Verification module for intermediate validation

To further improve the trustworthiness of the system, a self-verification (SV) module was implemented. This component enables the LLM to review its own outputs by checking logical consistency and recalculating values where necessary. The SV process involves two stages: first, assessing whether the reasoning and formula selection align with the retrieved standards; and second, verifying the numerical accuracy by re-executing the code and confirming unit consistency.

A specialized verification prompt guides the model through this process, instructing it to preserve correct answers or revise those that deviate from the expected standards. The SV module ensures that only validated and contextually accurate results are presented, enhancing the system's overall reliability in safety-critical applications.

4. Experiments

4.1. Experimental settings

This study aimed to evaluate the effectiveness of the proposed framework in automating structural design calculations, with a primary focus on the accuracy of the generated answers. The evaluation was conducted using a set of 50 structured queries derived from an actual Excel-based structural calculation sheet for precast concrete floor panels. Each query corresponds to a key step in a typical structural verification process, such as minimum thickness checks, load combination evaluations, and strength assessments for moment and shear.

A total of five model configurations were compared in the experiments, consisting of four baseline configurations and the proposed framework. The baseline models were incrementally constructed by adding different functional modules to a basic RAG-based generation pipeline:

- Baseline 1: RAG-based generation prompt only.

- Baseline 2: RAG-based generation + Formula Database.
- Baseline 3: Baseline 2 + SV.
- Baseline 4: Baseline 2 + CE.
- Proposed method: Baseline 2 + SV + CE.

The retrieval system was constructed using OpenAI's text-embedding-3-large model, with dense embeddings applied to both design code provisions and formula data. The vector database was created from two official Korean design code documents, the Highway Bridge Design Standard (2010) and the Korea Structural Concrete Design Code (2003), along with a manually curated formula database in JSON format. The design code documents were digitized using the Mathpix OCR tool and subsequently translated into English via gpt-4o-mini-2024-07-18. The embedding and storage of the entire content into the vector database were performed with a chunk size of 1024 tokens and a chunk overlap of 256 tokens. The top-5 most relevant chunks were retrieved for each query using dense similarity search.

Regarding the generation process, the proposed framework utilized gpt-4o-2024-08-06 with temperature set to 0 to ensure deterministic and consistent behavior. The entire experimental environment was implemented using Python 3.9.19 and LangChain, operating in a local Jupyter Notebook environment. All computations were performed via the OpenAI API on a CPU-only machine without GPU acceleration.

4.2. Experimental results

The experiment compared the accuracy of various model configurations depending on the inclusion of key functional modules. The second experiment examined the robustness of the proposed system across different structural scenarios by varying slab dimensions and material properties. In these experiments, an answer was considered correct if the relative error compared to the ground truth was within a tolerance of 0.1%.

As shown in Table 1, the baseline model without any additional modules (Baseline 1) exhibited an accuracy of 40%. The introduction of a formula database (Baseline 2) improved the accuracy to 50%. When Self-Verification (Baseline 3) and Code Execution (Baseline 4) modules were added individually, the accuracy increased to 54% and 62%, respectively. Notably, the proposed framework, which integrates Formula Database, Self-Verification, and Code Execution, recorded the highest accuracy of 76%.

Table 1. Accuracy Comparison of RAG-Based Structural Calculation Models with Formula Database, Self-Verification, and Code Execution Modules.

Model	Formula DB	Self-Verification	Code Execution	Accuracy
Baseline 1(Prompt only)				40%
Baseline 2(B1 + Formula DB)	✓			50%
Baseline 3(B2 + SV)	✓	✓		54%
Baseline 4(B2 + CE)	✓		✓	62%
Proposed Method(B2 + SV + CE)	✓	✓	✓	**76%**

5. Discussion

This section analyzes the performance differences among the tested model configurations and highlights how the integration of code execution and self-verification modules enhances both the precision and dependability of automated structural computations. The experimental outcomes clearly show that while a basic RAG-based generation framework can effectively retrieve and summarize relevant code provisions, it lacks the capability to conduct accurate structural calculations, which require strict numerical fidelity and compliance with established standards.

The models that relied solely on RAG retrieval or incorporated a formula database exhibited limited success in producing correct results. For instance, Baseline 1, which did not include any additional

modules for formula access or numerical validation, frequently produced flawed outputs due to misinterpretation of design requirements or incorrect formula usage. By adding a Formula Database in Baseline 2, the system gained access to a broader set of equations, which led to moderate improvements. However, the absence of computational validation meant that the generated answers still diverged from expected numerical outcomes.

Introducing Self-Verification in Baseline 3 allowed the system to identify and correct some logical errors and numerical inconsistencies within its responses. Nevertheless, without executing the calculations directly, the scope of correction was limited, particularly in handling precise decimal values. As illustrated in Fig. 2, when responding to the same structural query, Baseline 3 produced an answer that deviated from the exact result due to rounding issues and the lack of numerical validation via code execution. The proposed method, on the other hand, accurately computed and validated the result by executing the embedded Python code, demonstrating a higher level of numerical precision.

Baseline 4, which integrated Code Execution, demonstrated a more significant reduction in numerical inaccuracies by relying on the actual execution of generated code, thereby ensuring the numerical soundness of outputs. The Proposed Method, combining both Self-Verification and Code Execution, achieved the highest accuracy rate of 76%. This synergy leverages the strengths of both modules: code execution guarantees that calculations follow strict computational logic, while self-verification provides an additional safeguard by checking reasoning and unit consistency against the retrieved design provisions. In cases involving multi-step reasoning, such as determining reinforcement requirements or load-induced stresses, this dual-layered approach was effective in detecting intermediate errors and correcting them before presenting the final output.

Query	Baseline 3	Proposed Method

8-2. What is the factored moment capacity based on the calculated depth (a) and provided values?	Revised Answer
	
	### Explanation:
	Upon reviewing the calculation, I noticed that the units for \(f_y \) were incorrectly stated as kN/m² instead of the correct unit, which should be N/m² (or Pa). This affects the calculation of the factored moment capacity. Let's correct this and recalculate:
	
	Given:
	- \(\phi = 0.85 \)
	- \(A_s = 0.00134 \, \text{m}^2 \)
	- \(f_y = 500,000 \, \text{N/m}^2 \) (or 500 MPa)
	- \(d = 0.18 \, \text{m} \)
	- \(a = 0.0218 \, \text{m} \)
	
	Recalculate \(M_u \):
	
	\[M_u = \phi \times A_s \times f_y \times \left(d - \frac{a}{2} \right) \]
	
	\[M_u = 0.85 \times 0.00134 \times 500,000 \times \left(0.18 - \frac{0.0218}{2} \right) \]
	
	\[M_u = 0.85 \times 0.00134 \times 500,000 \times (0.18 - 0.0109) \]
	
	\[M_u = 0.85 \times 0.00134 \times 500,000 \times 0.1691 \]
	
	\[M_u = 96.57 \, \text{kN·m} \]
	
	### Answer: 96.57 kN·m ✗ **Wrong calculation**
		Revised Answer
		
		To calculate the factored moment capacity (\(M_u \)) based on the calculated depth (\(a \)) and provided values, we use the formula:
		
		\[M_u = \phi \times A_s \times f_y \times \left(d - \frac{a}{2} \right) \]
		
		Where:
		- \(\phi \) = Strength reduction factor (dimensionless) = 0.85
		- \(A_s \) = Provided reinforcement area (m²) = 0.0013404798861749217 m²
		- \(f_y \) = Yield strength of reinforcement (kN/m²) = 500,000 kN/m²
		- \(d \) = Effective depth (m) = 0.18 m
		- \(a \) = Depth of equivalent rectangular stress block (m) = 0.02176103711322925 m (from previous calculation)
		
		Let's perform the calculation:
		
		``python
		# Given values
		phi = 0.85
		A_s = 0.0013404798861749217 # m²
		f_y = 500000 # kN/m²
		d = 0.18 # m
		a = 0.02176103711322925 # m
		
		# Calculate factored moment capacity
		M_u = phi * A_s * f_y * (d - a / 2)
		
		print(M_u)
		``
		
		### Explanation:
		The original answer was correct in its calculation and explanation. The formula used to calculate the factored moment capacity is appropriate, and the values substituted into the formula are consistent with the given data. The calculation steps are mathematically correct, and the units are consistent throughout the process.
		
		### Final Answer: 96.34803687495618 (kN·m) ✓

Fig. 2. Comparison of structural calculation results for the same query between Baseline 3 and the proposed method.

Despite these strengths, the system also exhibited some limitations. Issues related to unit handling were observed in a few scenarios, particularly when retrieved equations used varying unit conventions or when the model failed to standardize outputs according to predefined formats. Additionally, the effectiveness of the self-verification process varied depending on the complexity of the query and the clarity of the retrieved context, sometimes resulting in inconsistent revisions.

6. Conclusion

This research introduced a Retrieval-Augmented Generation framework designed to automate structural calculations for precast concrete floor panels, incorporating both code execution and self-verification mechanisms to enhance accuracy and reliability. The proposed system was developed to address the shortcomings of traditional RAG-based approaches, which often lack the numerical precision and compliance enforcement necessary for structural engineering tasks.

Through a series of comparative experiments, the framework demonstrated superior performance over baseline models, achieving an accuracy of 76% across 50 structured queries. While models that relied solely on retrieval or symbolic reasoning struggled with consistency and computational correctness, the integration of executable code generation ensured that numerical outputs were validated against formal calculation logic. The self-verification module further improved the robustness of the system by providing an internal review of reasoning steps and unit alignment, thereby reducing the likelihood of errors.

Looking forward, the system has the potential to be extended and integrated into digital construction platforms such as Building Information Modeling (BIM) and digital twin environments. By enabling real-time updates and verifications as design changes occur, this approach could play a key role in advancing AI-driven workflows within the construction industry. Future developments may include refining unit handling, expanding the formula database, and improving context interpretation for more complex engineering applications.

Acknowledgements

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(Ministry of Science and ICT)(RS-2024-00457308). The authors would also like to thank **BRICON Co., Ltd.** for providing the structural calculation spreadsheets that were utilized in developing and evaluating the proposed framework.

References

[1] C. Eastman, J. Lee, Y. Jeong, and J. Lee, “Automatic rule-based checking of building designs,” *Automation in Construction*, vol. 18, no. 8, pp. 1011–1033, Dec. 2009, doi: 10.1016/j.autcon.2009.07.002.

[2] C. Sydora and E. Stroulia, “Rule-based compliance checking and generative design for building interiors using BIM,” *Automation in Construction*, vol. 120, p. 103368, Dec. 2020, doi: 10.1016/j.autcon.2020.103368.

[3] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, “Improving Language Understanding by Generative Pre-Training”.

[4] R. Hussain *et al.*, “Conversational AI-based VR system to improve construction safety training of migrant workers,” *Automation in Construction*, vol. 160, p. 105315, Apr. 2024, doi: 10.1016/j.autcon.2024.105315.

[5] J. Zhou and Z. Ma, “Named entity recognition for construction documents based on fine-tuning of large language models with low-quality datasets,” *Automation in Construction*, vol. 174, p. 106151, Jun. 2025, doi: 10.1016/j.autcon.2025.106151.

[6] F. Yang and J. Zhang, “Prompt-based automation of building code information transformation for compliance checking,” *Automation in Construction*, vol. 168, p. 105817, Dec. 2024, doi: 10.1016/j.autcon.2024.105817.

[7] Z. Ji *et al.*, “Survey of Hallucination in Natural Language Generation,” *ACM Comput. Surv.*, vol. 55, no. 12, pp. 1–38, Dec. 2023, doi: 10.1145/3571730.

[8] P. Lewis *et al.*, “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” Apr. 12, 2021, *arXiv*: arXiv:2005.11401. doi: 10.48550/arXiv.2005.11401.

[9] L. Siddharth and J. Luo, “Retrieval Augmented Generation using Engineering Design Knowledge,” Aug. 26, 2024, *arXiv*: arXiv:2307.06985. doi: 10.48550/arXiv.2307.06985.

[10] B. Hope, J. Bracey, S. Choukir, and D. Warner, “Assessment of ChatGPT for Engineering Statics Analysis,” Feb. 01, 2025, *arXiv*: arXiv:2502.00562. doi: 10.48550/arXiv.2502.00562.

[11] C. Wu *et al.*, “Retrieval augmented generation-driven information retrieval and question answering in construction management,” *Advanced Engineering Informatics*, vol. 65, p. 103158, May 2025, doi: 10.1016/j.aei.2025.103158.

[12] M. Uhm, J. Kim, S. Ahn, H. Jeong, and H. Kim, “Effectiveness of retrieval augmented generation-based large language models for generating construction safety information,” *Automation in Construction*, vol. 170, p. 105926, Feb. 2025, doi: 10.1016/j.autcon.2024.105926.

[13] S. V.-T. Tran *et al.*, “Leveraging large language models for enhanced construction safety regulation extraction,” *Journal of Information Technology in Construction (ITcon)*, vol. 29, no. 45, pp. 1026–1038, Dec. 2024, doi: 10.36680/j.itcon.2024.045.

[14] B. Shuai and C. H. Caldas, “A Case-Based Rag Methodology to Analyze Contract Risks for Construction Projects,” Sep. 04, 2024, *Social Science Research Network, Rochester, NY*: 4946907. doi: 10.2139/ssrn.4946907.

[15] Z. Qian and C. Shi, “Large language model-empowered paradigm for automated geotechnical site planning and geological characterization,” *Automation in Construction*, vol. 173, p. 106103, May 2025, doi: 10.1016/j.autcon.2025.106103.

[16] J. Wei *et al.*, “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models,” Jan. 10, 2023, *arXiv*: arXiv:2201.11903. doi: 10.48550/arXiv.2201.11903.

[17] X. Wang *et al.*, “Self-Consistency Improves Chain of Thought Reasoning in Language Models,” Mar. 07, 2023, *arXiv*: arXiv:2203.11171. doi: 10.48550/arXiv.2203.11171.
[18] A. Zhou *et al.*, “Solving Challenging Math Word Problems Using GPT-4 Code Interpreter with Code-based Self-Verification,” Aug. 15, 2023, *arXiv*: arXiv:2308.07921. doi: 10.48550/arXiv.2308.07921.
[19] R. Yamauchi, S. Sonoda, A. Sannai, and W. Kumagai, “LPML: LLM-Prompting Markup Language for Mathematical Reasoning,” Oct. 11, 2023, *arXiv*: arXiv:2309.13078. doi: 10.48550/arXiv.2309.13078.
[20] W. Chen, X. Ma, X. Wang, and W. W. Cohen, “Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks,” Oct. 23, 2023, *arXiv*: arXiv:2211.12588. doi: 10.48550/arXiv.2211.12588.
[21] “Mathpix: document conversion done right.” Accessed: Apr. 07, 2025. [Online]. Available: https://mathpix.com
[22] “LangChain.” Accessed: Apr. 07, 2025. [Online]. Available: https://www.langchain.com/
[23] “Model - OpenAI API.” Accessed: Apr. 07, 2025. [Online]. Available: https://platform.openai.com
[24] “Vector embeddings - OpenAI API.” Accessed: Apr. 07, 2025. [Online]. Available: https://platform.openai.com
[25] V. Karpukhin *et al.*, “Dense Passage Retrieval for Open-Domain Question Answering,” Sep. 30, 2020, *arXiv*: arXiv:2004.04906. doi: 10.48550/arXiv.2004.04906.