Automation in Construction 192 (2026) 107215

Contents lists available at ScienceDirect

Automation in Construction

journal homepage: www.elsevier.com/locate/autcon

Integrating large language models for automated structural analysis

Haoran Liang, Mohammad Talebi-Kalaleh, Qipei Mei *

Department of Civil and Environmental Engineering University of Alberta Edmonton, AB, T6G 2R3, Canada

ARTICLE INFO

Keywords:
Structural analysis
Finite element modeling
Large language models
LLM-based structural analysis

ABSTRACT

Automated analysis for engineering structures offers considerable potential for boosting efficiency by minimizing repetitive tasks. Although AI-driven methods are increasingly common, no systematic framework yet leverages Large Language Models (LLMs) for automatic structural analysis. This paper proposes a framework that employs domain-specific prompt design and in-context learning strategies to enhance LLM problem-solving capabilities and generative stability, enabling fully automated structural analysis from descriptive text to model outputs. A small-scale benchmark dataset consisting of 20 structural analysis word problems (SAWPs) is also introduced to evaluate the performance of different LLMs within the proposed framework. The results demonstrate that the proposed approach can increase the level of automation in solving SAWPs compared with traditional methods. Quantitatively, the framework built on GPT-5.4 and GPT-4o both achieved 100% accuracy, outperforming GPT-4 (85%), Gemini 1.5 Pro (80%), and Llama-3.3 (30%) on the test examples. Furthermore, integrating domain-specific instructions enhanced performance by 30% on problems with asymmetrical structural configurations.

1. Introduction

Structural engineering depends on thorough analysis to ensure accurate design under various loading conditions. While this analysis can be conducted through hand calculations, finite element modeling (FEM) software, or a combination of both, computational approaches are increasingly favored for their efficiency and ability to handle complex structural systems. These tools range from open-source platforms like OpenSees [1], Code_Aster [2], and Calculix [3], to commercial programs such as SAP2000 [4], Abaqus/CAE [5], and ANSYS [6]. They support a wide spectrum of structures, from 2D frames [7] and trusses [8] to high-rise buildings [9] and bridges [10]. However, traditional FEM-based methods still require substantial manual effort and specialized knowledge, highlighting the need for further automation and optimization in structural analysis.

Large Language Models (LLMs) offer a promising opportunity to revolutionize different sectors by automating laborious tasks while preserving engineering rigor. Recent advances in transformer architectures [11] have demonstrated remarkable capabilities not only in natural language processing (NLP) [12] but also in computer vision (CV) [13]. Moreover, these models have broadened their impact by addressing diverse language-related challenges in scientific fields [14]. However, beyond these task-specific evaluations, several studies have explored strategies to optimize LLMs for broader applicability and efficiency. For instance, pruning techniques [15,16] have been investigated to improve inference speed and reduce memory usage, while differential privacy methods [17] have been applied to enhance data security and protect user confidentiality without compromising model utility.

Despite the significant advances in LLM-driven applications across various practical domains, such as healthcare [18], finance [19], and data science [20], civil engineering has received comparatively less attention. Recently, researchers have begun exploring their applications in civil engineering, including LLM-based architectural flaw detection [21] and LLM-assisted technical writing for urban construction [22]. Fan et al. [23] introduced an interactive visual query system designed to assess the ergonomic postural risks of construction workers. This system incorporates visual question answering (VQA) to respond to visual queries regarding workers' exposure to ergonomic risks and image captioning (IC) to generate textual descriptions of these risks from images. Furthermore, Joffe et al. [24] proposed an LLM-based tool that enables engineers to ask code-related questions in natural language and receive accurate answers with citations, demonstrating its effectiveness using the 2020 National Building Code of Canada and highlighting its potential to improve design efficiency.

More recently, there has been some initial work exploring LLM's application in structural engineering, particularly structural analysis and design. Further work examined interactions between multiple LLM-based agents for programming tasks, utilizing FEniCS for FEA and GPT-3-turbo for code generation [25]. With only one basic example,

* Corresponding author.
E-mail addresses: hliang7@ualberta.ca (H. Liang), talebika@ualberta.ca (M. Talebi-Kalaleh), qipei.mei@ualberta.ca (Q. Mei).

https://doi.org/10.1016/j.autcon.2026.107215
Received 15 April 2025; Received in revised form 10 August 2026; Accepted 14 August 2026
Available online 1 September 2026

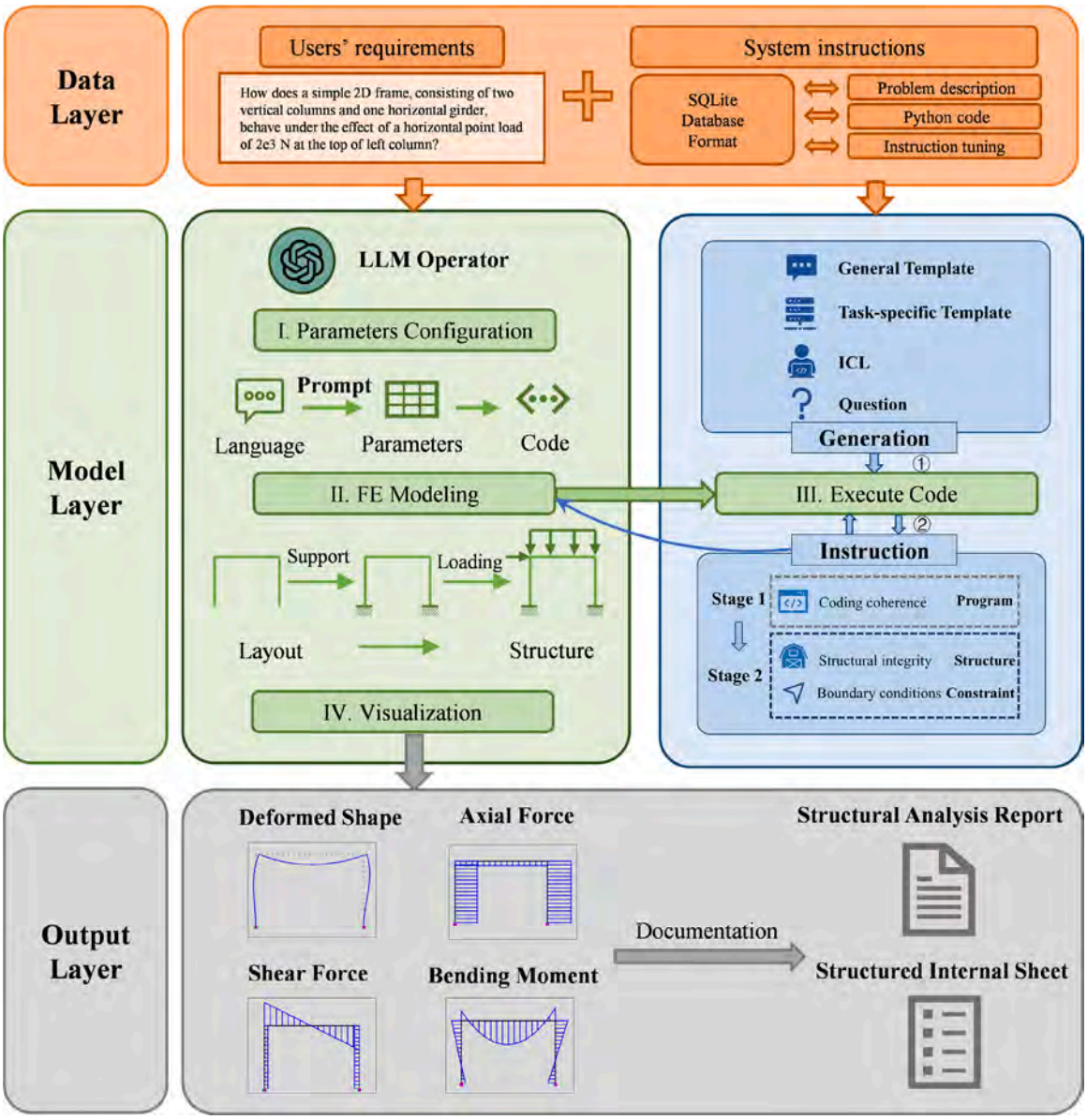


Fig. 1. LLM-Driven finite element analysis framework for 2D frame structures.

comparing different agents' performances becomes less persuasive and informative. Similarly, research on AI-driven design optimization has gained attention. Qin et al. [26] proposes an intelligent design and optimization system for shear wall structures, leveraging LLMs and generative artificial intelligence. This system employs an LLM as the central controller, interpreting engineers' language descriptions and converting them into executable code. Building on these developments, this paper is driven by two primary research questions: (1) how can the capabilities of modern LLMs be systematically leveraged to enable automated structural analysis from natural language descriptions, and (2) to what extent does the choice of underlying LLM influence the overall performance, reliability, and stability of the proposed framework?

Despite these promising efforts, existing approaches largely stop at proof-of-concept demonstrations and do not fundamentally address the persistent bottlenecks in current structural-analysis workflows. Traditional processes still demand extensive human intervention in model definition, parameter extraction, and verification across different finite element platforms, which hinders reproducibility and scalability. These steps are also highly repetitive and prone to human error, further reducing productivity and making consistent analysis difficult to achieve in practice. Prior LLM-based attempts either focus on single-step code generation or small, hand-crafted examples, lacking the systematic integration of reasoning, domain knowledge, and automated validation required for practical engineering use. Consequently, there remains a clear gap between isolated demonstrations of LLM competence and a fully automated, reproducible pipeline for structural analysis. In this paper, the gap is bridged by establishing an end-to-end framework that links natural language descriptions, multi-stage reasoning, and automated finite element analysis through OpenSeesPy. The proposed design explicitly targets workflow-level automation rather than convenience-driven code generation, enabling reproducible benchmarking and a deeper understanding of how LLMs can augment structural engineering practice.

To address these gaps, a novel framework is proposed that combines the generative capabilities of LLMs with the OpenSeesPy package [27], and its performance is assessed on a curated dataset of 20 structural analysis word problems (SAWPs). Specifically, multiple base models are employed within the framework, including GPT-4 [28], GPT-4o [29], GPT-5.4 [30], Llama 3 [31], and Gemini 1.5 [32], and compare their baseline performance with versions enhanced by techniques such as few-shot learning [33] and in-context learning (ICL) [34].

Question	Extracted information	Python scripts

How does a simple 2D frame, consisting of two vertical columns (4e0 meters in height) and one horizontal girder (6e0 meters in length), behave under the combined effects of a horizontal point load of 2e3 N at the top of one column and a uniform vertical distributed load of 1e4 N/m along the girder, considering elastic material properties with a Young's modulus E of 2e11 Pa, column cross-sectional area of 2e-3 m^2, girder cross-sectional area of 6e-3 m^2, column moment of inertia Ic of 1.6e-5 m^4, and girder moment of inertia Ig of 5.4e-5 m^4, all supports are fixed and what are the resulting deformations and internal forces (axial, shear, and moment) in the frame?	**Structural Layout**
	A simple 2D frame consists of:
	• **Two vertical columns**, each **4e0 meters** in height.
	• **One horizontal girder**, **6e0 meters** in length.
	
	Loading
	The frame is subjected to the combined effects of:
	• **A horizontal point load of 2e3 N** applied at the top of one column.
	• **A uniform vertical distributed load** of 1e4 N/m along the girder.
	
	Boundary Conditions
	• All supports are **fixed**.
	
	Visualization Request
	Analyze the frame's **deformation** and **internal forces** (axial, shear, and moment) through:
	• **Deformed Shape**
	• **Axial Force Diagram**
	• **Shear Force Diagram**
	• **Bending Moment Diagram**
		# Define the node coordinates
		ops.node(1, 0, 0) # Node 1 at (0, 0)
		ops.node(2, 0, 4e0) # Node 2 at (0, colL)
		ops.node(3, 6e0, 0) # Node 3 at (girL, 0)
		ops.node(4, 6e0, 4e0) # Node 4 at (girL, colL)
		
		# Define external loads
		Px = 2e3 # Point load in the x-direction
		Wy = -10e3 # Uniform load in the y-direction
		Wx = 0.0 # Uniform load in the x-direction
		# Applying point loads
		ops.load(2, Px, 0.0, 0.0) # Px applied in x-direction, no load in y and rotation
		# Applying distributed loads
		for etag in Ew:
		ops.eleLoad('-ele', etag, '-type', Ew[etag][0], Ew[etag][1], Ew[etag]
		
		# Define boundary conditions (supports)
		ops.fix(1, 1, 1, 1) # Fix all 3 DOFs (x, y, rotation) for node 1
		ops.fix(3, 1, 1, 1) # Fix all 3 DOFs (x, y, rotation) for node 3
		
		# Plot deformations (scaled) after analysis
		opsv.plot_defo()
		# Plot axial force distribution
		opsv.section_force_diagram_2d('N', sfacN)
		plt.title('Axial force distribution')
		# Plot shear force distribution
		opsv.section_force_diagram_2d('T', sfacV)
		plt.title('Shear force distribution')
		# Plot bending moment distribution
		opsv.section_force_diagram_2d('M', sfacM)
		plt.title('Bending moment distribution')

Fig. 2. Partial process illustrating how LLMs extract information from the problem description and convert it into Python scripts.

LLMs are capable of extracting critical information from textual problem descriptions and generating Finite Element Analysis (FEA) Python scripts for static 2-D frame structures, which serve as a controlled and interpretable setting for validating the core mechanisms of the framework. This controlled validation setting is intended primarily as a methodological proof-of-concept rather than a demonstration of immediate applicability to full-scale real-world structural engineering problems. This scope reflects a deliberate first step: although 3-D and dynamic analyses involve more degrees of freedom, more variables, and substantially higher computational demands, the proposed multi-stage pipeline (parameter extraction → FE modeling → visualization) remains directly applicable. Establishing correctness and stability in 2-D is therefore essential before generalizing to more complex structural systems. This enables engineers to provide concise natural language prompts rather than manually debugging code or interacting directly with analysis software, substantially reducing the time required for structural modeling and results visualization.

The remainder of this paper is structured as follows. In Section 2, the core methodology is detailed, including the tools and techniques employed by LLMs to solve SAWPs. Section 3 outlines the computational setup and the design of the 20-problem benchmark. Section 4 presents key findings on comparative performance analysis, generative stability, and the influence of system instructions on output quality. In Section 5, a focused discussion is provided on automating template and instruction generation as a pathway toward scalable prompt engineering and improved framework adaptability. Section 6 presents a case study applying the proposed method to the structural analysis of a practical steel racking system. Finally, Section 7 summarizes the main conclusions and outlines potential directions for future research.

The contributions of the paper are threefold. First, a unified end-to-end framework is introduced that links natural-language structural descriptions to parameter extraction, finite element model construction, and execution in OpenSeesPy, providing full validation through numerical and visual outputs. Second, a staged code-generation design is developed that decomposes the task into parameter extraction, FE modeling, and visualization, improving reliability and fault isolation during generation. Third, a 20-problem benchmark (SAWP-20) is curated with ground truth schematics, numerical references, and standardized descriptions to enable systematic evaluation of LLMs for structural analysis. Together, these contributions establish one of the first workflow-level assessment of LLM capabilities in automated structural analysis.

2. Methodology

This section presents the methodology of the proposed framework, including its general workflow and the data, model, and output layers. The key components and procedures within each layer are described to explain how natural language structural analysis problems are transformed into executable analyses and structured outputs.

2.1. General workflow

The workflow of the proposed model is illustrated in Fig. 1. In Section 2.2, the user requirements, the construction of system instructions, and the data structures used to store them are described. Section 2.3 details how structural analysis problems are decomposed, how the LLM processes each component, and how system instructions enhance the model's problem-solving capabilities within the proposed framework. Finally, in Section 2.4, the types of visualizations generated by the proposed framework and the output format are discussed.

2.2. Data layer

At the data layer, the input to the LLM comprises two components: (1) the user's requirements, provided as a problem description, and (2) system instructions, which integrate standard few-shot prompting, as introduced by Brown et al. [33]. And the proposed framework also introduces system instructions to enhance the LLM's ability to understand and solve SAWPs. Specifically, SQLite is used to structure the system instructions, which include: the problem description (formatted similarly to the user's requirements), the corresponding Python code for performing structural analysis and visualization as requested in the problem description (crucial for in-context learning, allowing the LLM to learn and replicate effective problem-solving formats), and instruction tuning elements, such as intermediate reasoning steps, to guide the LLM in solving specific sub-tasks.

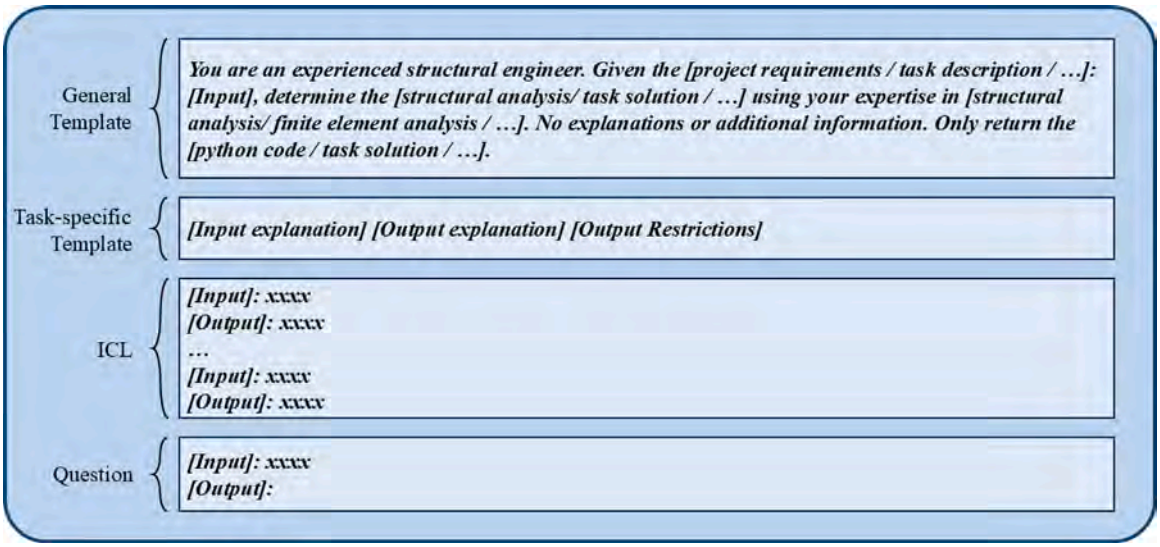


Fig. 3. ICL prompt template for all structural analysis problems.

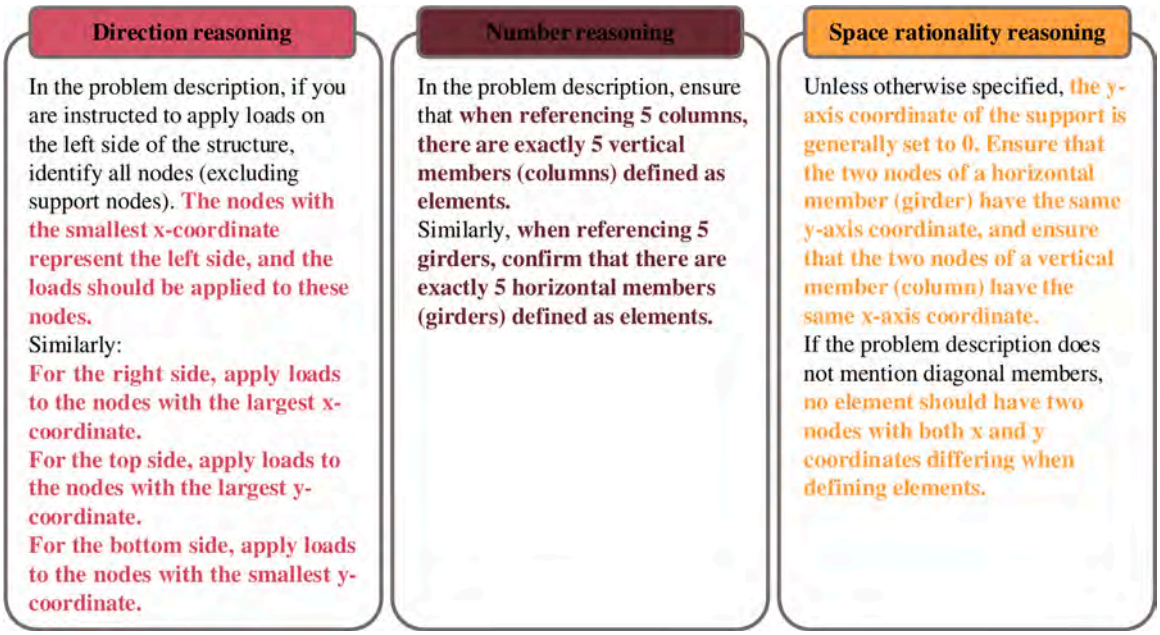


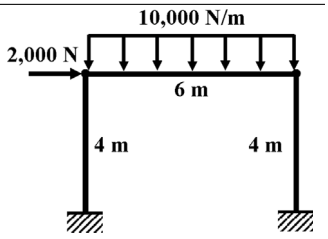
Fig. 4. Examples of commonsense reasoning for system instructions.

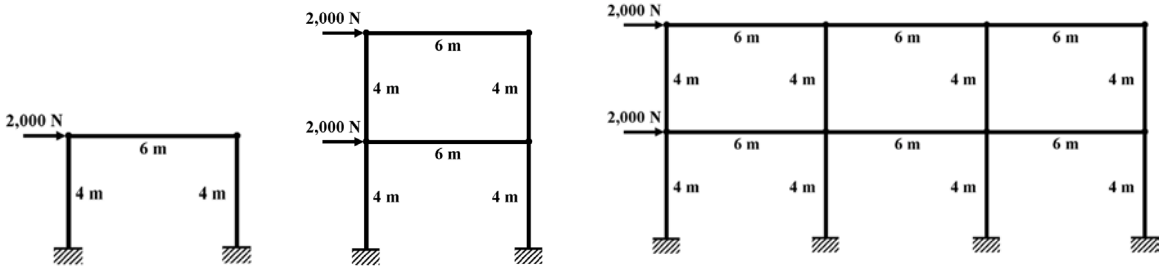
2.3. Model layer

At the model layer, the LLM serves as the primary operator. The OpenSeesPy package in Python is employed for structural analysis, which interfaces with OpenSees—an open-source finite element platform widely used in structural and earthquake engineering. Through Python scripting, OpenSeesPy allows the LLM to automatically generate and execute complete analysis scripts that yield numerical results and internal force diagrams. Given the computational nature of structural analysis, the proposed framework leverages the LLM's tool-use capability to generate executable Python code rather than relying on probabilistic recall from training data. To improve reliability and debugging efficiency, the code generation process is divided into three consecutive stages executed via API calls: (i) extraction of key geometric and material parameters from the problem description, (ii) definition of structural layout, supports, and loading conditions, and (iii) visualization of deformed shapes and internal force diagrams. This staged workflow mitigates long-context errors, enhances execution success rates, and isolates potential failures within specific code segments. The staged code-generation strategy constitutes a core methodological contribution of this work, as it improves robustness under long-context generation and significantly increases the stability of LLM-driven structural-analysis pipelines. All word problems are standardized in format to ensure consistency across the benchmark, enabling fair comparison and reproducible evaluation. The system instructions are refined based on expert feedback to further improve accuracy and reliability. Fig. 2 demonstrates how the LLM extracts essential information from problem descriptions and translates it into executable Python scripts.

Toolkit. LLMs themselves cannot perform precise structural analysis. However, their ability to generate code, particularly in Python, has significantly improved [35]. To leverage this capability, LLMs are employed to interface with two Python libraries: OpenSeesPy, a popular open-source Python package for finite element structural analysis, and OpsVis, a specialized library for visualizing results obtained by

Table 1
Partial problem descriptions and ground truth schematics.

Problem description	Ground truth
1. How does a simple 2D frame, consisting of two vertical columns (4×10^0 meters in height) and one horizontal girder (6×10^0 meters in length), behave under the combined effects of a horizontal point load of 2×10^3 N at the top of one column and a uniform vertical distributed load of 1×10^4 N/m along the girder, considering elastic material properties with a Young's modulus $E = 2 \times 10^{11}$ Pa, column cross-sectional area $A_c = 2 \times 10^{-3}$ m^2, girder cross-sectional area $A_g = 6 \times 10^{-3}$ m^2, column moment of inertia $I_c = 1.6 \times 10^{-5}$ m^4, and girder moment of inertia $I_g = 5.4 \times 10^{-5}$ m^4. All supports are fixed. What are the resulting deformations and internal forces (axial, shear, and moment) in the frame?	



(a) Example 3 (b) Example 5 (c) Example 9

Fig. 5. Pattern 1 for generating new example problems.

OpenSeesPy. The proposed instruction set for the LLM includes a step-by-step guide on using these libraries to solve a basic 2D frame structural analysis problem. The LLM is responsible for both performing the coding for structural analysis and accomplish visualizations using these tools. Traditionally, solving such problems with these libraries requires expertise in structural mechanics and Python programming. But utilizing the LLM as a compiler enables a direct translation of natural language problem descriptions into desired results, eliminating the need for extensive domain-specific knowledge and coding skills.

ICL Prompt Template. To enhance the LLM's understanding of structural engineering problems, supplementary information is provided alongside the problem description. The input includes an example problem, its corresponding code, and specific constraints to help the LLM generate runnable and stable Python scripts. To structure the system instructions effectively, an ICL prompt template is introduced in Fig. 3. ICL has been shown to improve LLM performance in coding and reasoning [34], and the style and format of the template are adapted from Ref. [36]. **General Template** includes preliminary settings to help the LLM understand the problem in a structural engineering context. **Task-Specific Template** provides explanations and constraints for both input and output. **ICL** contains a reference problem, using Example 1 from Table 1, along with detailed instructions on writing Python scripts using OpenSeesPy and OpsVis. Finally, **Question** presents the problem that the LLM needs to solve. The complete ICL template is shown in Appendix B.

Commonsense Reasoning. Instruction tuning has been shown to improve LLM performance across various reasoning tasks [37]. Since structural analysis problems require LLMs to interpret problem descriptions and reason about structural layouts, essential commonsense reasoning is incorporated into the system instructions to enhance problem-solving capabilities. Examples of commonsense reasoning used in the proposed approach are shown in Fig. 4. These examples are for illustration purposes only; the distributed loading direction reasoning is presented in Table 3.

2.4. Output layer

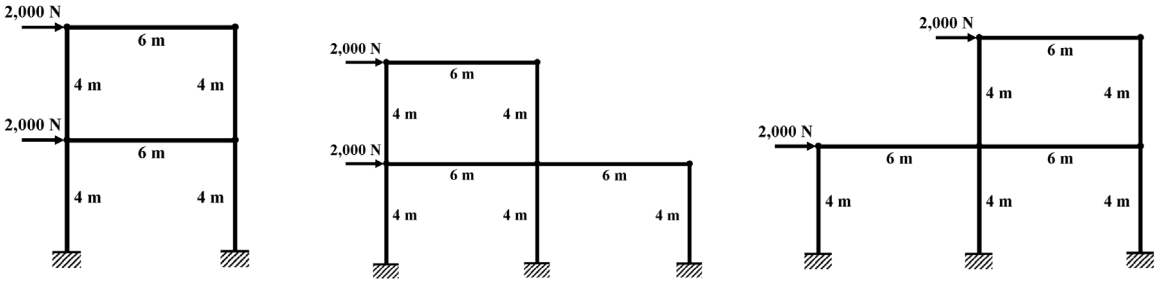
At the output layer, the results generated by the model layer are collected and formatted into structured outputs, such as reports or spreadsheets, depending on the user's requirements. To be more specific, with just a few lines of prompt, the LLM is able to automatically generate a structured report that includes the problem description, the generated code, explanations of the results, and spreadsheets containing internal forces at each node, along with their maximum and minimum values. By systematically processing inputs, integrating structured instructions, and producing clear, visualized outputs, the proposed framework significantly enhances the capability of LLMs to solve SAWPs, while maintaining transparency and interpretability throughout the generative process.

3. Experiment setup

This section describes the experimental setup used to evaluate the proposed framework, including the computational environment and the construction of the structural analysis test examples. The evaluation settings and benchmark design are presented to support consistent and reproducible comparison across different LLMs.

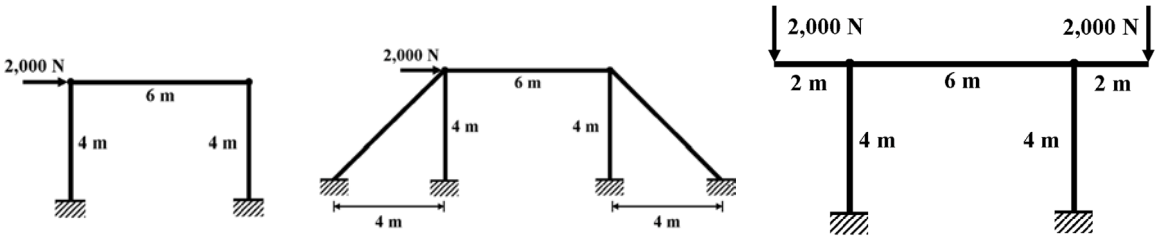
3.1. Computational setup

Several LLMs are used in this work, including gpt-5.4, gpt-4o-2024-08-06 and gpt-4-turbo from OpenAI, Gemini 1.5 Pro from Google, and llama-3.3-70b-versatile from Meta. These models were accessed via API calls, minimizing the need for local computational resources. Most API-based tasks completed within a few seconds per request. All scripts were written in Python 3.12 and executed on a Lenovo Legion 5 (2022) laptop running Windows 11. The machine is equipped with a Ryzen 7 6800H CPU (8 cores, up to 4.7 GHz), 64 GB RAM, 1 TB SSD, and an NVIDIA GeForce RTX 3060 GPU. The key Python libraries used to call LLMs API include `openai`, `google-generativeai`, and `groq`. API keys were securely stored using environment variables. For all API calls, the sampling parameters were set to `temperature = 1.0` and `top_p = 1.0` to ensure consistent randomness across models during generation.



(a) Example 5 (b) Example 11 (c) Example 13

Fig. 6. Pattern 2 for generating new example problems.



(a) Example 3 (b) Example 17 (c) Example 19

Fig. 7. Pattern 3 for generating new example problems.

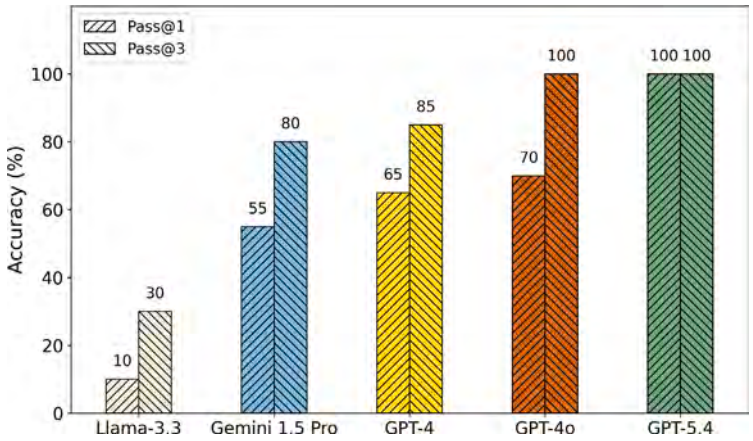


Fig. 8. Pass@1 and Pass@3 performance of five LLMs on the twenty manually designed examples.

3.2. Test examples

The paper focuses on a foundational structural pattern: simple 2D frames. These structures are straightforward to describe within a single dialogue turn. Currently, there is no publicly available dataset that systematically includes structural descriptions, clear layouts, and corresponding ground truth. To fill this gap, a benchmark comprising 20 different structural analysis problems has been manually designed. For each structure, a detailed word problem description is provided along with ground truth solutions to validate the outputs generated by LLMs. A key problem appears in Table 1, while the full set of problems is included in Appendix A.

To be more specific, each problem was presented in a standardized format, consisting of five main components: geometrical properties, material properties, loading conditions, boundary conditions, and requirements for result visualization. Corresponding schematics were prepared as ground truth for validating the accuracy of the model's outputs. In the creation of the dataset, LLMs are used to generate code based on manually defined problem descriptions, and the generated results are reviewed. This approach significantly reduced the time required compared to manually creating the entire dataset. Three patterns were adapted to generate new example problems. In Pattern 1 (Fig. 5), the number of stories and bays are modified to create new structures. For instance, transitioning from Fig. 5(a) to Fig. 5(b), the structure is extended from one to two stories, while from Fig. 5(a) to Fig. 5(c), a single-bay, one-story structure is expanded to a three-bay, two-story structure. This pattern requires LLMs to accurately define nodes and elements. In Pattern 2 (Fig. 6), asymmetry is introduced. For example, in Fig. 6(b), the first bay consists of two stories, while the second bay has only one floor, requiring LLMs to infer and handle structural asymmetry. In Pattern 3, additional structural features commonly found in engineering are incorporated, such as diagonal members (Fig. 7(b)) and cantilever beams (Fig. 7(c)). This pattern challenges LLMs to interpret and solve problems involving complex structural configurations.

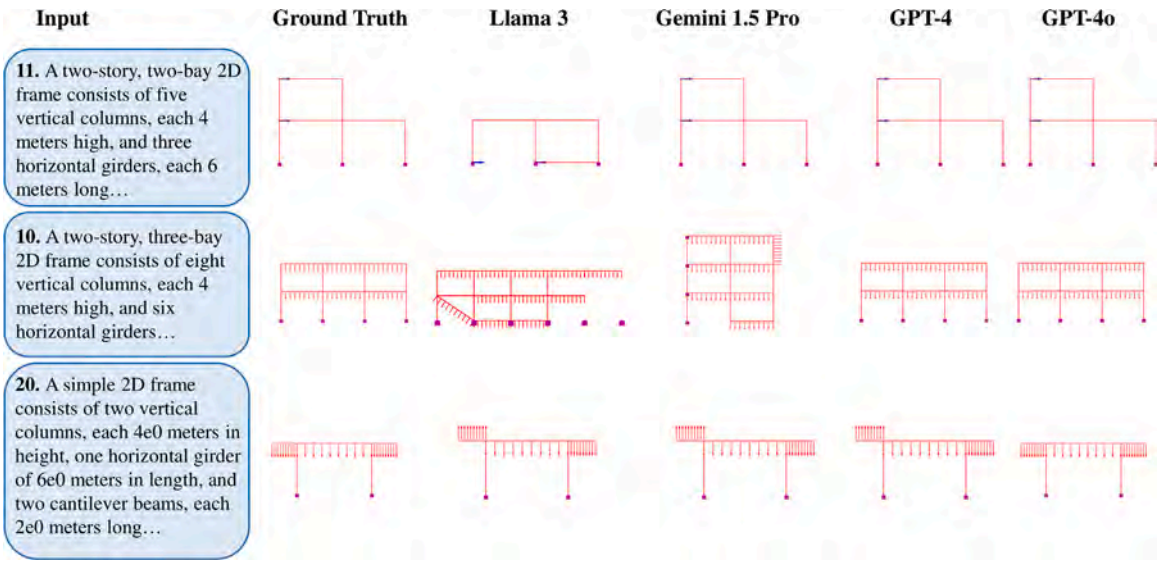


Fig. 9. Performance of four LLMs on 20 manually designed examples (detailed problem descriptions are provided in Appendix A).

4. Results and discussion

This section begins with a comparative analysis of the proposed framework across different LLMs' performance, followed by results on the generative stability of the framework on the benchmark and the impact of instructions on output quality.

4.1. Comparative analysis of LLM-generated structural analysis

It is also observed that the proposed framework demonstrates a remarkable ability to learn and generalize structural patterns from minimal examples. Building on the strong performance shown in Fig. 8, the framework not only solved problems with high accuracy but also exhibited pattern abstraction capabilities from a single reference case. Specifically, after being provided with only one example describing the modeling of a one-bay, one-story frame (Example 1), the framework successfully extended this structural pattern to more complex configurations, such as a one-bay, two-story frame (Example 5), a two-bay, one-story frame (Example 7), and a three-bay, two-story frame (Example 9). This demonstrates the model's ability to extract high-level structural concepts such as *bay* and *story* from natural language and apply them accurately in code generation for structural modeling. Furthermore, the framework exhibited strong extrapolation capabilities beyond the initial instruction. While the reference example only illustrated the modeling of columns and girders, the model autonomously incorporated additional structural components, including diagonal members (Example 15 and Example 17) and cantilever beams (Example 19). These results suggest that LLMs, when guided by a structured prompting framework, are capable of reasoning from a limited seed input to produce correct and meaningful generalizations, even when faced with novel structural patterns absent from the original example.

Multiple LLMs (Llama-3.3, Gemini 1.5 Pro, GPT-4, GPT-4o, and GPT-5.4) are evaluated on 20 SAWPs. The findings indicate that GPT-4, GPT-4o, and Gemini 1.5 Pro exhibit strong capabilities in generating Python code for structural analysis, achieving pass@3 accuracy rates of 85%, 100%, and 80% on this benchmark, respectively, while GPT-5.4 achieves perfect one-shot performance on SAWP-20 (Pass@1 = 100%), solving all problems correctly under the same experimental setup. A detailed discussion on evaluation metrics, including the distinction between pass@k and pass^k, is provided in Appendix D.2. The proposed framework based on these models effectively extracts information from natural language problem descriptions, constructs finite element models in Python, and visualizes the results. The results suggest that with advancements in state-of-the-art LLMs, the proposed framework holds great potential to assist structural engineers in structural analysis, ranging from 2D to more complex structures. Notably, GPT-5.4 demonstrates further performance improvements beyond prior models, achieving full reliability under the same controlled evaluation setting. This progression highlights the increasing ability of advanced LLMs to automate structural analysis within the controlled context of the SAWP-20 benchmark, while acknowledging that such results are obtained under curated textual inputs and absence of real-world data noise. With rapid advancements in LLM development, performance optimization techniques, and ongoing research into their applications in structural engineering, the role of LLMs in this field is expected to expand further, offering deeper integration and enhanced capabilities. Additionally, techniques such as few-shot learning and in-context learning can enhance the performance of LLMs in structural analysis. Among the evaluated models, GPT-5.4 exhibits the strongest capability in information extraction, code generation, system instruction comprehension, and spatial reasoning, achieving an accuracy of 100% without any fine-tuning. As seen in Fig. 8, the proposed framework based on GPT-5.4 consistently achieves full reliability in a single pass and outperforms the other models in solving the given problems, while GPT-4o also demonstrates strong performance under limited attempts. For detailed per-model results and error classifications, please refer to Appendix D.1.

As shown in Fig. 9, the proposed framework based on different LLMs exhibit distinct strengths and weaknesses. Among the four models, llama 3 fails all three problems, highlighting its relatively weak general capability in solving these tasks. Additionally, compared to GPT-4 and GPT-4o, Gemini 1.5 Pro struggles with scaling simple structural patterns. For instance, in Example 10, when transitioning from a one-bay, one-story frame to a three-bay, two-story frame, it consistently either misdefines the structural layout or attempts to use loop statements to define nodes and elements, but the generated code is never executable. Furthermore, GPT-4o demonstrates a better understanding of system instructions than GPT-4. For example, in Example 20, when both models are given specific instruction to determine the correct direction of distributed loads, GPT-4o successfully improves its performance on this subtask, whereas GPT-4 does not.

4.2. Evaluation of output stability in the proposed framework

All 20 SAWPs from the dataset in Appendix A are analyzed to assess the generative stability of the proposed framework based on GPT-4o.

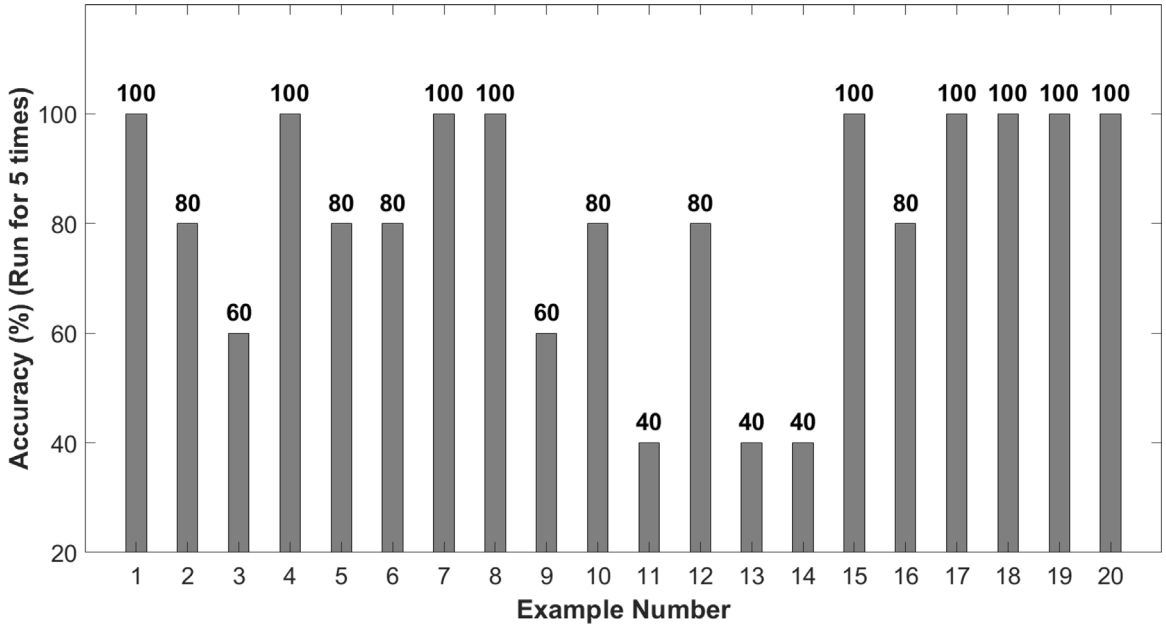


Fig. 10. Stability experiment for GPT-4o (accuracy (%) of 20 SAWPs).

Although these 2D frame SAWPs are relatively simple for structural engineering professionals, language models often struggle with them due to the lack of systematically curated datasets in this field during training stages and inherent limitations in space reasoning [38]. LLMs are proficient in writing code, solving mathematical problems through reasoning, and retrieving information via web searches. However, as demonstrated in Appendix C, LLMs consistently fail to solve SAWPs using standard prompting methods [37]. Standard prompting refers to directly asking the model to solve the problem without providing any additional guidance, such as solution templates or task-specific instructions. Despite the inclusion of structural analysis knowledge within their training data, LLMs struggle to effectively utilize this knowledge to accurately solve SAWPs. Appendix C also provides the ground truth solutions for comparison. Notably, instruction tuning has significantly improved GPT-4o's ability to handle SAWPs. The proposed framework exhibits robust generative stability across most cases. Furthermore, in particularly challenging scenarios, the framework successfully solves the problems within a limited number of iterative attempts.

The experiment was conducted using GPT-4o within the proposed framework, with five runs performed for each benchmark problem. The reported accuracy represents the probability that the proposed framework successfully generates a correct and complete solution within these five attempts. The key findings from the stability experiment are summarized in Fig. 10. First, with only one reference example and limited attempts, the proposed framework successfully generates complete and executable Python code using OpenSeesPy to perform structural analysis for all 20 SAWPs.

However, the experiment also reveals that the proposed framework's performance declines when handling asymmetrical frames, as seen in Examples 11 and 13. This indicates that while LLMs can effectively learn and generalize structural patterns, they may struggle with asymmetric configurations. The detailed experimental results are further summarized in Table 2.

As illustrated in Fig. 11, when GPT-4o generates runnable code, some scripts fail due to coding coherence issues and other inconsistencies, primarily making two types of mistakes: Error Type 1, where the framework fails to sketch the structural layout as required. For instance, in Example 9, the frame should be a three-bay, two-story frame, but GPT-4o generates a two-bay, two-story frame; in Example 11, it fails to define a required node, resulting in an incomplete second-story frame; and in Example 13, it incorrectly defines an element, leading to an inaccurate second story. Error Type 2 involves incorrect definitions of boundary conditions, where the framework misassigns loads. In the three given examples, GPT-4o was instructed to assign point loads to the left side of the frame but instead assigned them to all nodes on the first floor; in other cases, it may also misdefine structural supports. Two main causes of these errors are identified: first, at this stage, both input and output are limited to text (including code), meaning GPT-4o cannot "see" the results it generates, preventing it from recognizing misassigned elements or loads. In future work, multimodal capabilities may be incorporated into the framework by integrating a validation layer that uploads graphical results during the initial generation phase. If the LLM detects an error pattern, it can provide modification suggestions to update the generated output. Second, LLMs are currently provided only with correct structural patterns, without exposure to incorrect ones. Due to the inherent randomness of LLM-generated outputs, a single problem may yield multiple structural layouts, some correct and others incorrect. To mitigate this, future work can incorporate negative sampling techniques to reduce the likelihood of generating incorrect structural patterns when modeling frames from natural language input.

Table 2
Summary of stability experiment.

Accuracy	Example Number	Total
40%	11, 13, 14	3
60%–80%	2, 3, 5, 6, 9, 10, 12, 16	8
100%	1, 4, 7, 8, 15, 17, 18, 19, 20	9

Considering the inherent randomness in the generative process of LLMs, each experiment is run three consecutive times for each problem. A problem is considered solved if the proposed framework produces a correct response in at least one of the three attempts. A Best-of-N sampling strategy [39] is adopted because it better reflects the model's potential and allows for a fair comparison of the upper-bound performance across different models. The choice of three attempts was determined empirically, balancing computational cost and output variability; additional tests with five attempts confirmed

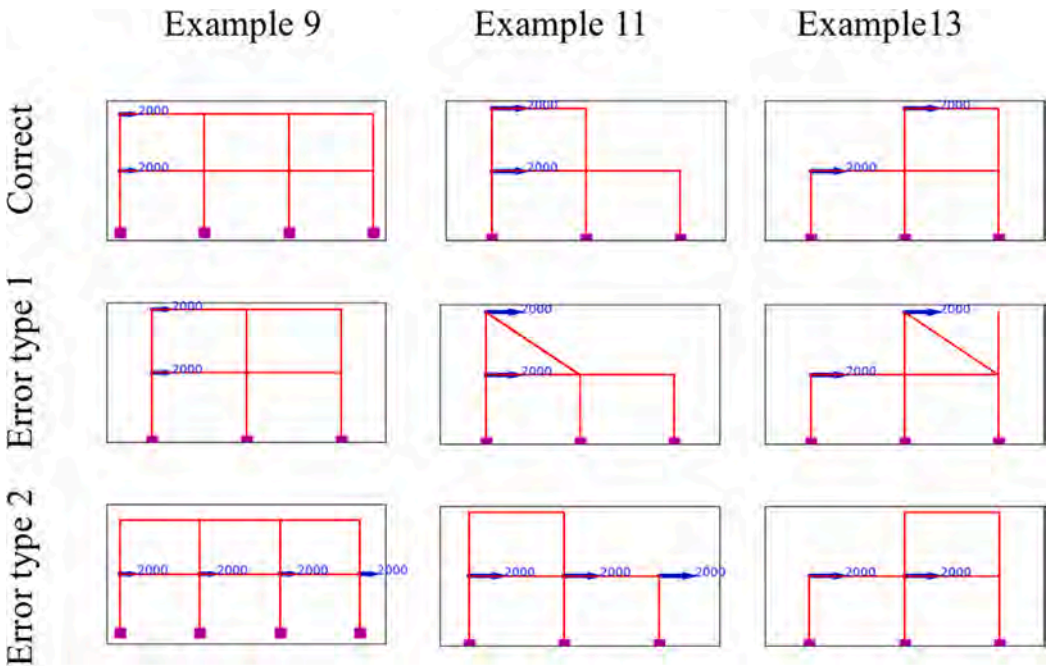


Fig. 11. GPT-4o utilizes OpenSeesPy and OpsVis to generate results based on user input.

Table 3
Structural reasoning instructions.

Category	Description
Direction reasoning	To correctly apply loads to specific regions of the structure, follow these steps: 1. Identify all nodes in the structure, excluding support nodes. 2. Determine the location where the load should be applied based on coordinate values: • Left side: Nodes with the smallest x-coordinate. • Right side: Nodes with the largest x-coordinate. • Top side: Nodes with the largest y-coordinate. • Bottom side: Nodes with the smallest y-coordinate. 3. Assign the load to the identified nodes accordingly.
Number reasoning	To ensure the structure contains the correct number of members: 1. Identify all vertical members (columns) in the structure. 2. Verify that the number of defined columns matches the stated count in the problem. 3. Similarly, count all horizontal members (girders) and ensure their number aligns with the problem description.
Space rationality reasoning	To maintain spatial consistency in structural elements: 1. By default, set the y-coordinate of support nodes to zero unless specified otherwise. 2. Ensure horizontal members (girders) have two nodes with identical y-coordinates. 3. Ensure vertical members (columns) have two nodes with identical x-coordinates. 4. If diagonal members are not mentioned, ensure that no element has nodes with both x and y coordinates differing.
Distributed loading direction reasoning	When applying a distributed load to an element: 1. Check the direction of the load: If the load is inward, apply a negative sign to the load value. 2. Identify the starting and ending nodes of the element. 3. If the x-coordinate of the starting node is smaller than the ending node, assign the load as given. 3. If the x-coordinate of the starting node is greater than the ending node, negate the distributed load value before applying it.

that the overall trend and model ranking remain consistent, indicating limited sensitivity to this sampling parameter. A solution is considered correct when the generated Python code executes successfully in OpenSeesPy and its numerical outputs closely match the predefined ground truth. Specifically, nodal coordinates, nodal displacements, and internal forces (axial, shear, and bending moment) are extracted as reference values. If the differences between the generated and ground-truth results fall within a small tolerance (less than 5%), the output is automatically accepted. When discrepancies exceed this range, manual verification is performed by examining the generated deformed shapes, internal force diagrams, and geometric consistency to determine correctness and categorize failure modes.

4.3. Impact of instructions on LLM output quality

Although the proposed framework can understand SAWPs descriptions and generate Python scripts through referencing only a simple example and its corresponding code, it still makes many basic mistakes due to its lack of space reasoning capabilities for structures in 2D space. To address this, structured system instructions are provided to help the language model align with human understanding of structural layouts in space. The complete set of system instructions can be found in Table 3. To be more specific, Direction reasoning provides instructions on how to judge the direction of the structure. Humans have a unique visual system, allowing them to determine the orientation of a structure

Fig. 12. Positive result.

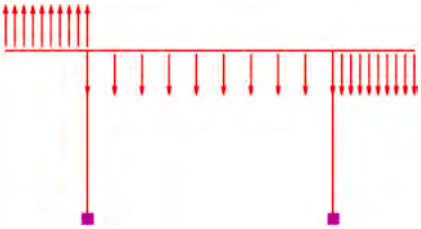


Fig. 13. Negative result.

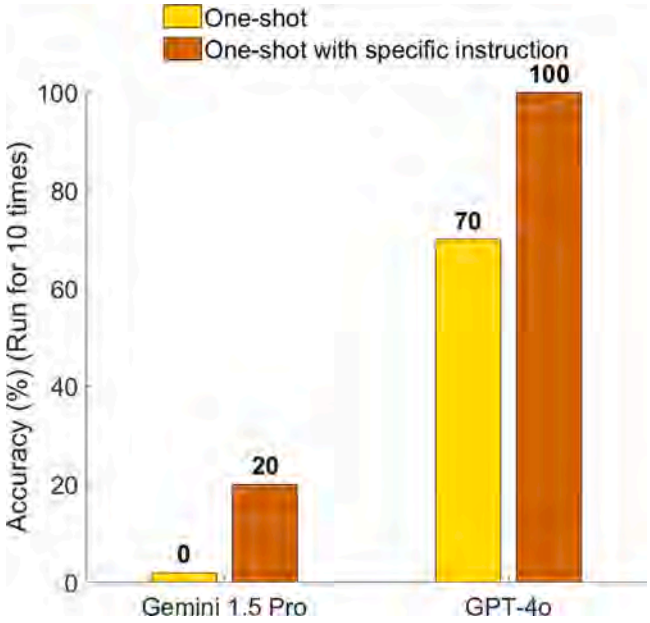


Fig. 14. LLMs, such as Gemini 1.5 Pro and GPT-4o, are optimized with distributed loading direction reasoning instruction to enhance their accuracy in applying distributed loads to the structure in problem 20.

without explicit reasoning. However, the used LLMs lack visual recognition capabilities. Instead, they have access to the coordinates of all structural nodes. This component instructs LLMs on how to judge the direction of the structure using node coordinates to accurately define loads or supports (see Figs. 12–14). Furthermore, number reasoning is introduced in the system instructions. Before adding this instruction, it is found that LLMs often defined elements with a different number than what was mentioned in the problem description. To address this issue, this instruction ensures that LLMs define the correct number of elements in the structures.

For a more complex example like Example 12 in Appendix A, if instructions in Table 3 are not provided for the LLMs, the framework generates a cluster of results, as shown in Fig. 15(a). These results contain various types of errors, such as incorrect layout definitions, incorrect load applications, incorrect support definitions, or a combination of these mistakes. The primary reason for these errors is the increased number of nodes and elements, along with the asymmetry of the structure. To address this, the complete instructions from Table 3 were incorporated into the system instructions to evaluate whether they could enhance the framework's performance on this task. As shown in Fig. 15(b), the problem description from Example 12 was provided to GPT-4o, and the experiment was conducted ten times both with and without the complete system instructions. It is found that including the instructions increases the framework's accuracy on this problem from 50% to 80%. However, continuously expanding system instructions is not always beneficial. Increasing the number of instructions introduces a scalability dilemma: more instructions lead to higher token consumption, increasing computational costs and resource demands for maintaining an efficient generative process. Additionally, excessive instructions can confuse LLMs, as they lack weighted prioritization to distinguish more critical directives from less relevant ones. Ambiguous instructions may even degrade overall performance compared to experiments conducted without any system instructions. Therefore, determining optimal instruction combinations and developing efficient methods for synthesizing and compressing system instructions could be valuable directions for future research.

5. Discussion

Automating Template and Instruction Generation

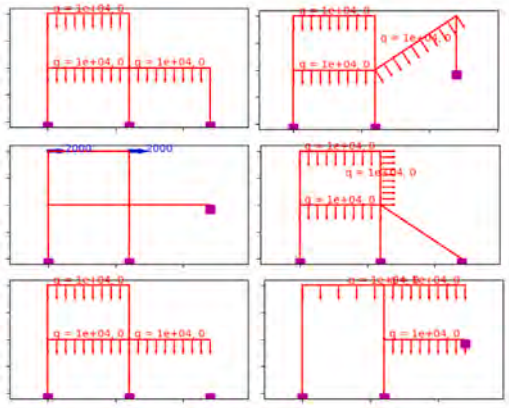
Future extensions could automate parts of the template and instruction creation process. LLMs can be leveraged to extract recurring reasoning patterns from solved examples and generalize them into reusable templates, reducing manual effort. In addition, clustering successful prompts or using reinforcement feedback from execution outcomes can iteratively refine instruction quality. Such automation would transform prompt engineering from a manual, expert-driven task into a self-improving, data-driven process, enhancing scalability and adaptability across diverse structural problems.

Practical validation strategies. In real-world structural engineering applications, where ground truth is often unavailable, human engineers remain essential as verifiers. Instead of relying solely on numerical outputs, LLMs are also prompted to generate interpretable visualizations, such as structural schematics that explicitly illustrate member dimensions, boundary conditions, applied loads, and section assignments. These visual representations enable engineers to efficiently and reliably assess the correctness of the generated solutions. Moreover, in practical deployments, human inspection is indispensable to ensure safety and reliability, as LLMs fundamentally lack direct physical understanding of the world and do not acquire real-world physical grounding during training.

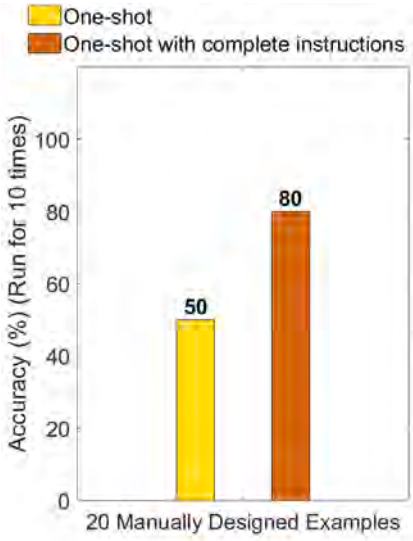
Cost analysis. Using GPT-4o on SAWP-20, the average cost per task is $0.0175 (input) and $0.0159 (completion), totaling approximately $0.033 per task and $0.668 for a full 20-task trial. Input and completion tokens contribute 52.5% and 47.5% of the total cost, respectively.

6. Case study

The racking system structural analysis problem from [40] is introduced to demonstrate the practical applicability of the proposed method. As shown in Fig. 16, the racking elevation defines the front geometry of the system, including the number of pallets at each level, the corresponding pallet loads, the number of beams, and the vertical layout of the structure. The side elevation illustrates the arrangement of columns and bracing members together with their geometric relationships. Combined with the annotated dimensions and cross-sectional details of the beams, columns, and braces, these views constitute a complete structural drawing of the racking system. A problem description, as shown in Fig. 17, is then used as input to the proposed framework for solving the corresponding 2D frame racking system problem. Using the GPT-4o-based framework, a success rate of 80% over 10 independent trials is achieved in generating the structural analysis results directly from the provided problem description.



(a) Cluster generated by GPT-4o for the solution of example 12



(b) Impact of complete instructions

Fig. 15. Comparison of clustering results and instruction tuning.

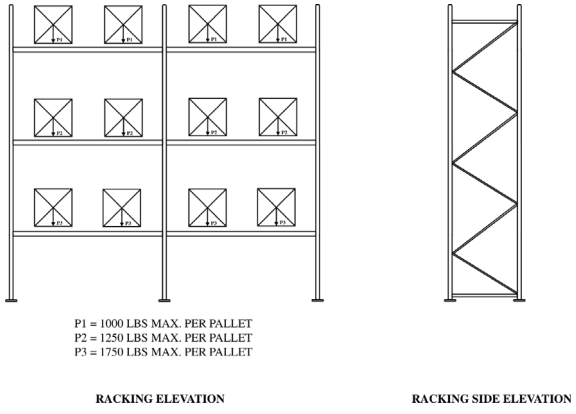


Fig. 16. Racking system example.

7. Conclusions

This paper explored the potential of LLMs to perform structural analysis based solely on natural language input. The contributions are: (i) a three-stage code generation pipeline (parameter extraction → FE modeling → visualization) that mitigates long-context failures and eases fault isolation; (ii) embedded commonsense and spatial reasoning instructions to align LLM behavior with structural reasoning; and (iii) an end-to-end setup that executes in OpenSeesPy and yields interpretable outputs with minimal human intervention. The results also demonstrate generalization from a single reference example to more complex patterns (e.g., more stories/bays, asymmetry), laying groundwork for workflow integration and dataset curation. The framework employs structured prompt design and in-context learning strategies. By integrating LLMs with the finite element analysis tool OpenSeesPy, the framework streamlines model creation and modification in domain-specific software. A dataset of 20 structural analysis word problems, including ground truth solutions, problem descriptions, and corresponding code, was curated to compare LLM performance, assess stability, and test instruction effects. Together, these components demonstrate a practical path to integrating LLM-generated models into engineering workflows.

Major conclusions are as follows:

1. GPT-5.4 demonstrates the best performance among the evaluated models on the benchmark, showing strong capability in solving structural analysis problems based on natural language input. This holds under end-to-end execution in OpenSeesPy with interpretable outputs.
2. Reasoning instructions significantly enhance performance, especially for tasks involving code generation and reasoning in structural engineering contexts, indicating the importance of prompt design in aligning LLM behavior with engineering goals. In particular, commonsense reasoning and spatial instructions (direction, number, consistency, load direction) yield measurable gains, especially on asymmetric layouts.
3. LLMs demonstrate strong potential for integration into structural engineering workflows by improving efficiency and streamlining repetitive modeling tasks, enabling rapid, reliable decision-making from natural-language specifications. Generalization from a single example to more complex patterns is further observed, suggesting scalability with sparse exemplars and providing groundwork for future datasets and workflow integration.

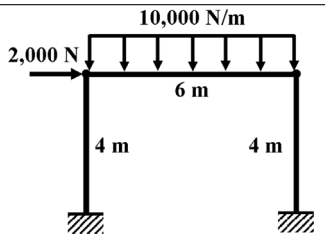
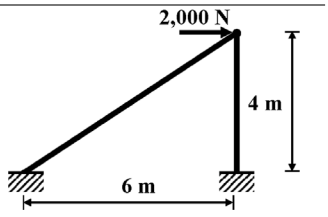
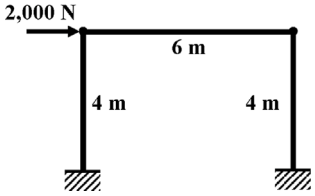
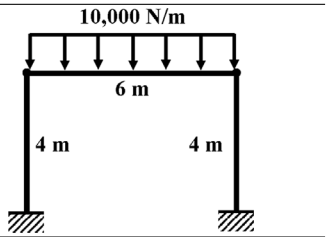
While the results demonstrate the potential of LLM-based structural analysis, several important limitations must be addressed before widespread adoption in practice. The current benchmark focuses on static 2-D frame problems and does not yet cover dynamic or 3-D cases, which limits generalizability. In addition, the current SAWP-20 benchmark is manually curated and conducted in a controlled validation setting; therefore, the reported results should be interpreted primarily as a methodological proof-of-concept rather than evidence of full real-world engineering readiness. First, the dataset size and the number of experimental runs were limited since the dataset was manually designed for this project while also considering the usage of APIs. Scaling to larger, more diverse datasets, including dynamic and 3-D problems, and reducing API costs are key next steps. A key question for future research is how to obtain larger datasets and invoke LLMs' APIs at lower costs in this domain. Future work will further evaluate the framework on more realistic textbook-level and engineering case studies beyond the current SAWP-20 benchmark. In addition, different LLMs exhibit distinct trade-offs in performance, reliability, API cost, and latency. Future work will include more comprehensive benchmarking of these practical trade-offs across models for real-world engineering applications. Second, although the cost of

Problem Description (for structural scheme see Figure 16)

Coordinates are given in feet (1 ft = 12 in); forces are in kip; stiffness is in kip/in^2. The overall layout consists of two longitudinal bays in plan, each beam length being 8.0 ft, with a side elevation frame width of 3.5 ft between posts and a post height of 16.0 ft. Beam elevations are placed at 4.0 ft, 8.5 ft, and 13.0 ft, and each beam carries two pallets. The beams are 4 in Z-sections of steel, the columns are steel U-channels 3.079 in × 2.795 in × 0.0787 in with a height of 16.0 ft, modeled as elastic beamcolumns with $E = 29{,}000$ kip/in^2, and the braces are steel U-channels 1.0 in × 1.0 in × 0.054 in, treated as truss elements (pinpin) with a typical diagonal length of approximately 4.3 ft. The column centerlines are defined from $(0, 0)$ to $(0, 16.0)$ and from $(3.5, 0)$ to $(3.5, 16.0)$. The diagonal truss braces connect successive nodes in the following sequence: $(0, 0.5) \rightarrow (3.5, 0.5)$, $(3.5, 0.5) \rightarrow (0, 3)$, $(0, 3) \rightarrow (3.5, 5.5)$, $(3.5, 5.5) \rightarrow (0, 8)$, $(0, 8) \rightarrow (3.5, 10.5)$, $(3.5, 10.5) \rightarrow (0, 13)$, $(0, 13) \rightarrow (3.5, 15.5)$, and $(3.5, 15.5) \rightarrow (0, 15.5)$. The supports are fixed bases located at $(0, 0)$ and $(3.5, 0)$. The weight on each pallet: $P_{4.0\text{ ft}} = 1.75$ kip (1750 lb), $P_{8.5\text{ ft}} = 1.25$ kip (1250 lb), and $P_{13.0\text{ ft}} = 1.00$ kip (1000 lb). Under this configuration, Please do a structural analysis of this structure.

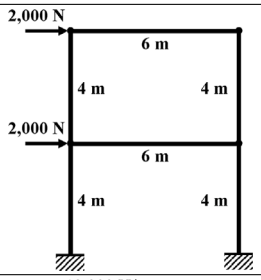
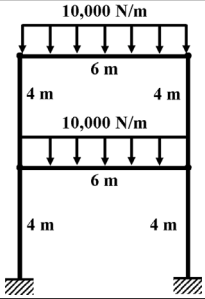
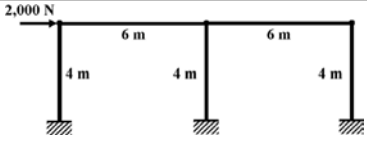
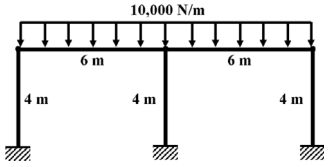
Fig. 17. Racking system problem description.

Table 4
Problem descriptions and ground truth schematics.

Problem description	Ground truth
1. How does a simple 2D frame, consisting of two vertical columns (4×10^0 meters in height) and one horizontal girder (6×10^0 meters in length), behave under the combined effects of a horizontal point load of 2×10^3 N at the top of one column and a uniform vertical distributed load of 1×10^4 N/m along the girder, considering elastic material properties with a Young's modulus $E = 2 \times 10^{11}$ Pa, column cross-sectional area $A_c = 2 \times 10^{-3}$ m^2, girder cross-sectional area $A_g = 6 \times 10^{-3}$ m^2, column moment of inertia $I_c = 1.6 \times 10^{-5}$ m^4, and girder moment of inertia $I_g = 5.4 \times 10^{-5}$ m^4. All supports are fixed. What are the resulting deformations and internal forces (axial, shear, and moment) in the frame?	
2. How does a simple 2D frame, consisting of one vertical column (4×10^0 meters in height), one diagonal member forming the brace in the left side of the column, where the horizontal height from the top of the column to the support of the diagonal member is 6×10^0 meters, behave under the effect of a horizontal point load of 2×10^3 N at the top of the column, considering elastic material properties with a Young's modulus $E = 2 \times 10^{11}$ Pa, column cross-sectional area of 2×10^{-3} m^2, diagonal member cross-sectional area of 6×10^{-3} m^2, column moment of inertia of 1.6×10^{-5} m^4, and diagonal member moment of inertia of 5.4×10^{-5} m^4 and all supports are fixed. What are the resulting deformations and internal forces (axial, shear, and moment) in the frame?	
3. How does a simple 2D frame, consisting of two vertical columns (4×10^0 meters in height) and one horizontal girder (6×10^0 meters in length), behave under the effect of a horizontal point load of 2×10^3 N at the top of one column, considering elastic material properties with a Young's modulus $E = 2 \times 10^{11}$ Pa, column cross-sectional area of 2×10^{-3} m^2, girder cross-sectional area of 6×10^{-3} m^2, column moment of inertia $I_c = 1.6 \times 10^{-5}$ m^4, and girder moment of inertia $I_g = 5.4 \times 10^{-5}$ m^4 and all supports are fixed. What are the resulting deformations and internal forces (axial, shear, and moment) in the frame?	
4. How does a simple 2D frame, consisting of two vertical columns (4×10^0 meters in height) and one horizontal girder (6×10^0 meters in length), behave under the effect of a uniform vertical distributed load of 1×10^4 N/m along the girder, considering elastic material properties with a Young's modulus $E = 2 \times 10^{11}$ Pa, column cross-sectional area of 2×10^{-3} m^2, girder cross-sectional area of 6×10^{-3} m^2, column moment of inertia $I_c = 1.6 \times 10^{-5}$ m^4, and girder moment of inertia $I_g = 5.4 \times 10^{-5}$ m^4 and all supports are fixed. What are the resulting deformations and internal forces (axial, shear, and moment) in the frame?	

manually augmenting exemplars with system instructions is minimal in the few-shot setting, such annotation costs could be prohibitive for fine-tuning, and the cost of expert-level evaluation of LLM-generated results could be very high (though this could potentially be surmounted with synthetic data generation [41] and LLMs as judges [42]). Moreover, the current instruction templates are handcrafted and may not scale or generalize well; future work will explore automatic template induction, instruction distillation, and full automation of verification to reduce expert dependence. In addition, evaluation remains largely manual and subjective; automated numerical validation, repeated trials, statistical testing, and latency evaluation will improve reproducibility and rigor. Finally, the reliance on proprietary LLMs poses cost and reproducibility concerns, and future work will include fine-tuning open-source baselines and benchmarking their cost–performance trade-offs. Improving the alignment of tactical approaches with structural engineering needs remains a promising direction for future research. Finally, this work only investigates LLMs' performance on manually designed tasks; further research could explore how to effectively integrate LLMs into real-world applications in this sector. Future work could involve incorporating advanced techniques such as supervised

Table 5
Additional problem descriptions and ground truth schematics.

Problem description	Ground truth
5. How does a two-story 2D frame, consisting of four vertical columns (4 m in height for each story) and two horizontal girders (6 m in length each), behave under the horizontal point load of 2×10^3 N at each column on the left side? Consider elastic material properties with Young's modulus of 2×10^{11} Pa, column cross-sectional area of 2×10^{-3} m^2, girder cross-sectional area of 6×10^{-3} m^2, column moment of inertia of 1.6×10^{-5} m^4, and girder moment of inertia of 5.4×10^{-5} m^4. All supports are fixed. What are the resulting deformations and internal forces (axial, shear, and bending moment) within the frame?	
6. How does a two-story 2D frame, consisting of four vertical columns (4 m in height for each story) and two horizontal girders (6 m in length each), behave under the uniform vertical distributed load of 1×10^4 N/m along each girder? Consider elastic material properties with Young's modulus of 2×10^{11} Pa, column cross-sectional area of 2×10^{-3} m^2, girder cross-sectional area of 6×10^{-3} m^2, column moment of inertia of 1.6×10^{-5} m^4, and girder moment of inertia of 5.4×10^{-5} m^4. All supports are fixed. What are the resulting deformations and internal forces (axial, shear, and bending moment) within the frame?	
7. How does a one-story two-bay 2D frame, consisting of three vertical columns (4 m in height each) and two horizontal girders (6 m in length each), behave under the horizontal point load of 2×10^3 N at the top of the column on the left side? Consider elastic material properties with Young's modulus of 2×10^{11} Pa, column cross-sectional area of 2×10^{-3} m^2, girder cross-sectional area of 6×10^{-3} m^2, column moment of inertia of 1.6×10^{-5} m^4, and girder moment of inertia of 5.4×10^{-5} m^4. All supports are fixed. What are the resulting deformations and internal forces (axial, shear, and bending moment) within the frame?	
8. How does a one-story two-bay simple 2D frame, consisting of three vertical columns (4 m in height each) and two horizontal girders (6 m in length each), behave under the uniform vertical distributed load of 1×10^4 N/m along each girder? Consider elastic material properties with Young's modulus of 2×10^{11} Pa, column cross-sectional area of 2×10^{-3} m^2, girder cross-sectional area of 6×10^{-3} m^2, column moment of inertia of 1.6×10^{-5} m^4, and girder moment of inertia of 5.4×10^{-5} m^4. All supports are fixed. What are the resulting deformations and internal forces (axial, shear, and bending moment) within the frame?	

fine-tuning (SFT) and reinforcement learning fine-tuning (RLFT) to further enhance LLMs' reasoning and tool-using capabilities for structural analysis. The benchmark will also be extended to dynamic and 3-D analyses, with automated instruction and template generation and integrated statistical verification pipelines to ensure scalability, reproducibility, and reliability. In addition, integrating LLMs specialized in structural analysis with LLMs which are proficient in structural design has the potential to create agentic structural engineers for real-world applications.

CRediT authorship contribution statement

Haoran Liang: Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Investigation, Formal analysis. **Mohammad Talebi-Kalaleh:** Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology, Investigation, Data curation, Conceptualization. **Qipei Mei:** Writing – review & editing, Supervision, Resources, Project administration, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by Natural Sciences and Engineering Research Council of Canada (NSERC) through the Alliance Grant [ALLRP 581074-22]

Appendix A. Dataset

See Tables 4–9.

The full implementation of our LLM-driven structural analysis framework is available at:

https://github.com/DelosLiang/LLM-StructuralAnalysis

The SAWP-20 benchmark dataset introduced in this study can be accessed at:

https://huggingface.co/datasets/NateLiang/SAWP20

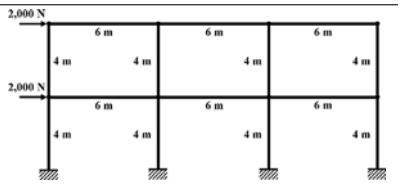
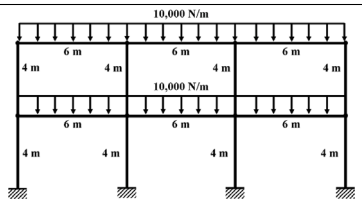
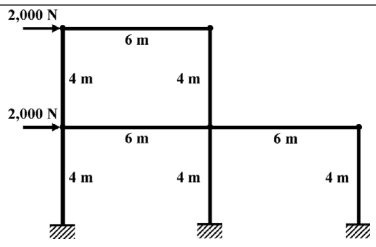
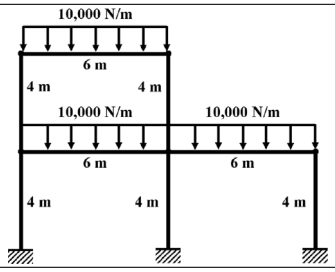
Appendix B. ICL template

See Figs. 18–24.

Appendix C. Baseline

We present detailed traces for representative failure cases to illustrate how the system behaves without in-context learning (ICL) templates and to pinpoint error modes. Each trace is structured as a sequence of sectioned, breakable `tcolorbox` logs (problem, tool/-code, intermediate outputs, and diagnostics) to support debugging and reproducibility.

Table 6
Additional problem descriptions and ground truth schematics.

Problem description	Ground truth
9. How does a two-story three-bay 2D frame, consisting of 8 vertical columns (4 m in height each) and 6 horizontal girders (6 m in length each), behave under the horizontal point load of 2×10^3 N at each column on the left side? Consider elastic material properties with Young's modulus of 2×10^{11} Pa, column cross-sectional area of 2×10^{-3} m^2, girder cross-sectional area of 6×10^{-3} m^2, column moment of inertia of 1.6×10^{-5} m^4, and girder moment of inertia of 5.4×10^{-5} m^4. All supports are fixed. What are the resulting deformations and internal forces (axial force, shear force, and bending moment) within the frame?	
10. How does a two-story three-bay 2D frame, consisting of 8 vertical columns (4 m in height each) and 6 horizontal girders (6 m in length each), behave under the uniform vertical distributed load of 1×10^4 N/m along each girder? Consider elastic material properties with Young's modulus of 2×10^{11} Pa, column cross-sectional area of 2×10^{-3} m^2, girder cross-sectional area of 6×10^{-3} m^2, column moment of inertia of 1.6×10^{-5} m^4, and girder moment of inertia of 5.4×10^{-5} m^4. All supports are fixed. What are the resulting deformations and internal forces (axial force, shear force, and bending moment) within the frame?	
11. How does a two-story two-bay 2D frame, where the first bay has two stories and the second bay has one story, consisting of 5 vertical columns (4 m in height each) and 3 horizontal girders (6 m in length each), behave under the horizontal point load of 2×10^3 N at each column on the left side? Consider elastic material properties with Young's modulus of 2×10^{11} Pa, column cross-sectional area of 2×10^{-3} m^2, girder cross-sectional area of 6×10^{-3} m^2, column moment of inertia of 1.6×10^{-5} m^4, and girder moment of inertia of 5.4×10^{-5} m^4. All supports are fixed. What are the resulting deformations and internal forces (axial force, shear force, and bending moment) within the frame?	
12. How does a two-story two-bay 2D frame, where the first bay has two stories and the second bay has one story, consisting of 5 vertical columns (4 m in height each) and 3 horizontal girders (6 m in length each), behave under the uniform vertical distributed load of 1×10^4 N/m along each girder? Consider elastic material properties with Young's modulus of 2×10^{11} Pa, column cross-sectional area of 2×10^{-3} m^2, girder cross-sectional area of 6×10^{-3} m^2, column moment of inertia of 1.6×10^{-5} m^4, and girder moment of inertia of 5.4×10^{-5} m^4. All supports are fixed. What are the resulting deformations and internal forces (axial force, shear force, and bending moment) within the frame?	

General Template

```
You are an experienced structural engineer. Given the problem description, your task is to use Python codes to
achieve finite element analysis using your finite element analysis and Python coding knowledge.
```

Fig. 18. General template.

Table 7
Additional problem descriptions and ground truth schematics.

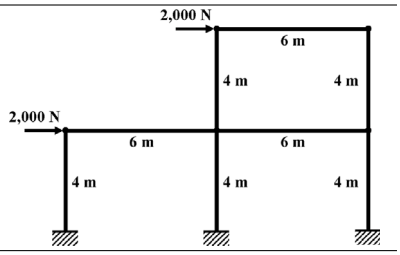
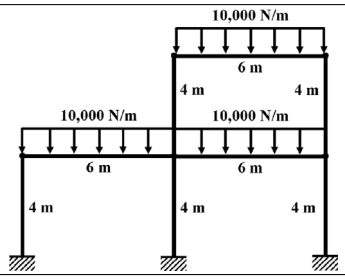
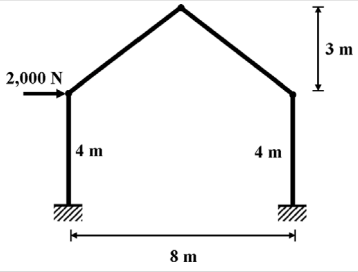
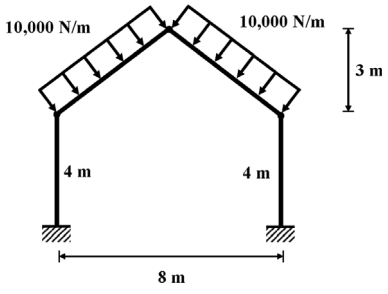
Problem description	Ground truth
13. How does a two-story two-bay 2D frame, where the first bay has one story and the second bay has two stories, consisting of 5 vertical columns (4 m in height each) and 3 horizontal girders (6 m in length each), behave under the horizontal point load of 2×10^3 N at the column on the left side on the first story and on the second story? Consider elastic material properties with Young's modulus of 2×10^{11} Pa, column cross-sectional area of 2×10^{-3} m^2, girder cross-sectional area of 6×10^{-3} m^2, column moment of inertia of 1.6×10^{-5} m^4, and girder moment of inertia of 5.4×10^{-5} m^4. All supports are fixed. What are the resulting deformations and internal forces (axial force, shear force, and bending moment) within the frame?	
14. How does a two-story two-bay 2D frame, where the first bay has one story and the second bay has two stories, consisting of 5 vertical columns (4 m in height each) and 3 horizontal girders (6 m in length each), behave under the uniform vertical distributed load of 1×10^4 N/m along each girder? Consider elastic material properties with Young's modulus of 2×10^{11} Pa, column cross-sectional area of 2×10^{-3} m^2, girder cross-sectional area of 6×10^{-3} m^2, column moment of inertia of 1.6×10^{-5} m^4, and girder moment of inertia of 5.4×10^{-5} m^4. All supports are fixed. What are the resulting deformations and internal forces (axial force, shear force, and bending moment) within the frame?	
15. How does a simple 2D frame, consisting of two vertical columns (4×10^0 meters in height) with a spacing of (8×10^0) meters between the two columns, have two identical diagonal members forming the roof in the middle of the columns, where the vertical height from the top of the columns to the peak of the roof is (3×10^0) meters, behave under the effect of a horizontal point load of (2×10^3) N at the left column? Consider elastic material properties with a Young's modulus E of (2×10^{11}) Pa, column cross-sectional area of (2×10^{-3}) m^2, diagonal member cross-sectional area of (6×10^{-3}) m^2, column moment of inertia of (1.6×10^{-5}) m^4, and diagonal member moment of inertia of (5.4×10^{-5}) m^4. All supports are fixed. What are the resulting deformations and internal forces (axial, shear, and moment) in the frame?	
16. How does a simple 2D frame, consisting of two vertical columns (4×10^0 meters in height) with a spacing of (8×10^0) meters between the two columns, have two identical diagonal members forming the roof in the middle of the columns, where the vertical height from the top of the columns to the peak of the roof is (3×10^0) meters, behave under the uniform distributed load of (1×10^4) N/m along each diagonal member? The direction of the distributed load is inward. Considering elastic material properties with a Young's modulus E of (2×10^{11}) Pa, column cross-sectional area of (2×10^{-3}) m^2, diagonal member cross-sectional area of (6×10^{-3}) m^2, column moment of inertia of (1.6×10^{-5}) m^4, and diagonal member moment of inertia of (5.4×10^{-5}) m^4. All supports are fixed. What are the resulting deformations and internal forces (axial, shear, and moment) in the frame?	

Table 8
Additional problem descriptions and ground truth schematics.

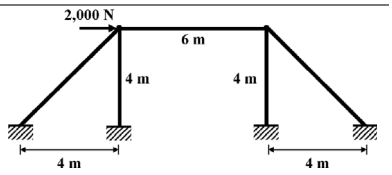
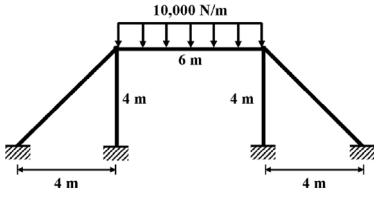
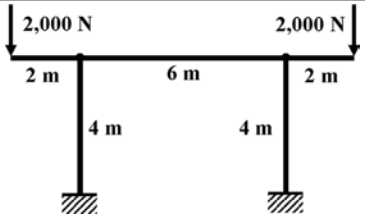
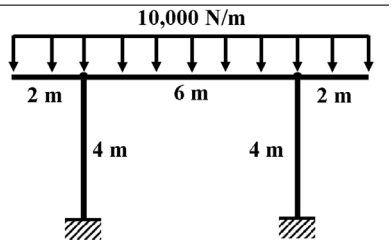
Problem description	Ground truth
17. How does a simple 2D frame, consisting of two vertical columns (4×10^0 meters in height), one horizontal girder (6×10^0 meters in length) and two diagonal members forming the braces (one node of the diagonal member is connected to the top of the column and another node is connected to the ground), one is on the left side of the left column and another is on the right side of the right column, where the horizontal length from the top of the column to the support of the diagonal member is (4×10^0) meters, behave under the effect of a horizontal point load of (2×10^3) N at the left column? Consider elastic material properties with a Young's modulus E of (2×10^{11}) Pa, column cross-sectional area of (2×10^{-3}) m^2, diagonal member cross-sectional area of (6×10^{-3}) m^2, girder cross-sectional area of (6×10^{-3}) m^2, column moment of inertia of (1.6×10^{-5}) m^4, diagonal member moment of inertia of (5.4×10^{-5}) m^4, girder moment of inertia of (5.4×10^{-5}) m^4. All supports are fixed. What are the resulting deformations and internal forces (axial, shear, and moment) in the frame?	
18. How does a simple 2D frame, consisting of two vertical columns (4×10^0 meters in height), one horizontal girder (6×10^0 meters in length) and two diagonal members forming the braces (one node of the diagonal member is connected to the top of the column and another node is connected to the ground), one is on the left side of the left column and another is on the right side of the right column, where the horizontal length from the top of the column to the support of the diagonal member is (4×10^0) meters, behave under the uniform distributed load of (1×10^4) N/m along the girder? Consider elastic material properties with a Young's modulus E of (2×10^{11}) Pa, column cross-sectional area of (2×10^{-3}) m^2, diagonal member cross-sectional area of (6×10^{-3}) m^2, girder cross-sectional area of (6×10^{-3}) m^2, column moment of inertia of (1.6×10^{-5}) m^4, diagonal member moment of inertia of (5.4×10^{-5}) m^4, girder moment of inertia of (5.4×10^{-5}) m^4. All supports are fixed. What are the resulting deformations and internal forces (axial, shear, and moment) in the frame?	
19. How does a simple 2D frame, consisting of two vertical columns (4×10^0 meters in height), one horizontal girder (6×10^0 meters in length) and two cantilever beams (2×10^0 meters in length) on both sides which are connected to the top of two columns, behave under the combined effects of two vertical point loads of (2×10^3) N at the end of each cantilever beam on both sides? Consider elastic material properties with a Young's modulus E of (2×10^{11}) Pa, column cross-sectional area of (2×10^{-3}) m^2, girder and cantilever beam cross-sectional area of (6×10^{-3}) m^2, column moment of inertia of (1.6×10^{-5}) m^4, and girder and cantilever beam moment of inertia of (5.4×10^{-5}) m^4. All supports are fixed. What are the resulting deformations and internal forces (axial, shear, and moment) in the frame?	
20. How does a simple 2D frame, consisting of two vertical columns (4×10^0 meters in height), one horizontal girder (6×10^0 meters in length) and two cantilever beams (2×10^0 meters in length) on both sides which are connected to the top of two columns, behave under the uniform vertical distributed load of (1×10^4) N/m along the girder and two cantilever beams? Consider elastic material properties with a Young's modulus E of (2×10^{11}) Pa, column cross-sectional area of (2×10^{-3}) m^2, girder and cantilever beam cross-sectional area of (6×10^{-3}) m^2, column moment of inertia of (1.6×10^{-5}) m^4, and girder and cantilever beam moment of inertia of (5.4×10^{-5}) m^4. All supports are fixed. What are the resulting deformations and internal forces (axial, shear, and moment) in the frame?	

Table 9
Summary of the 20 benchmark problems with key characteristics and ground-truth types.

Problem	Bays	Stories	Diagonal members	Cantilever members	Numerical result	Visual diagram
1	1	1	0	0	✓	✓
2	1	1	1	0	✓	✓
3	1	1	0	0	✓	✓
4	1	1	0	0	✓	✓
5	1	2	0	0	✓	✓
6	1	2	0	0	✓	✓
7	2	1	0	0	✓	✓
8	2	1	0	0	✓	✓
9	3	2	0	0	✓	✓
10	3	2	0	0	✓	✓
11	2	2	0	0	✓	✓
12	2	2	0	0	✓	✓
13	2	2	0	0	✓	✓
14	2	2	0	0	✓	✓
15	1	1	2	0	✓	✓
16	1	1	2	0	✓	✓
17	1	1	2	0	✓	✓
18	1	1	2	0	✓	✓
19	1	1	0	2	✓	✓
20	1	1	0	2	✓	✓

Task-specific Template

```
Now I will give you the description of the background information that tries to solve.
How does a simple 2D frame, consisting of two vertical columns (4e0 meters in height) and one horizontal girder (6e0
meters in length), behave under the combined effects of a horizontal point load of 2e3 N at the top of one column and
a uniform vertical distributed load of 1e4 N/m along the girder, considering elastic material properties with a
Young's modulus E of 2e11 Pa, column cross-sectional area of 2e-3 m^2, girder cross-sectional area of 6e-3 m^2,
column moment of inertia Ic of 1.6e-5 m^4, and girder moment of inertia Ig of 5.4e-5 m^4, all supports are fixed, and
what are the resulting deformations and internal forces (axial, shear, and moment) in the frame?

[Output explanation]
Now, please generate the codes for the following problem.

[Output Restrictions]
Please keep the format of the codes strictly the same as the codes I have provided as the background information, and
you do not have to generate the codes already provided as parameters configuration and post-processing.
Please only keep generating codes; do not generate any descriptions which cannot be run.
Remove the "Python" from the first line, because it cannot be run by the IDE.
Generate the code up to and including the step "Apply distributed loads on the elements".

Remove the following parts from the generated code.
Parameters configuration
import openseespy.opensees as ops  # Import OpenSeesPy for structural analysis
import opsvis as opsv              # Import opsvis for visualization
import matplotlib.pyplot as plt    # Import Matplotlib for plotting
ops.wipe()                         # Clear any existing model
ops.model('basic', '-ndm', 2, '-ndf', 3)  # Define a 2D model with 3 degrees of freedom per node (DOF)

# Column and girder lengths
colL, girL = 4., 6.

# Section properties: cross-sectional area (A) and moment of inertia (Iz)
```

Fig. 19. Task-specific instruction template.

```
Acol, Agir = 2.e-3, 6.e-3
IzCol, IzGir = 1.6e-5, 5.4e-5

# Young's modulus (E)
E = 200.e9

# Define time series for constant loads
ops.timeSeries('Constant', 1)

# Define load pattern using the constant time series
ops.pattern('Plain', 1, 1)

Post-processing
# Analysis settings
ops.constraints('Transformation')          # Apply transformation constraints
ops.numberer('RCM')                        # Renumber nodes using Reverse Cuthill-McKee (RCM)
ops.system('BandGeneral')                  # Define the solution algorithm
ops.test('NormDispIncr', 1.0e-6, 6, 2)     # Convergence test criteria
ops.algorithm('Linear')                    # Use linear algorithm for solving
ops.integrator('LoadControl', 1)           # Control load increments
ops.analysis('Static')                     # Define a static analysis
ops.analyze(1)                             # Perform the analysis

# Print the model data
ops.printModel()

# Plot the model after defining elements
opsv.plot_model()
plt.title('plot_model after defining elements')

# Plot the applied loads on the model in 2D
opsv.plot_loads_2d(
    nep=10,                 # Number of points along each element
    sfac=1,                 # Scale factor for loads
    fig_wi_he=(10, 5),      # Width and height of the figure
    fig_lbrt=(0.1, 0.1, 0.9, 0.9),  # Left, bottom, right, top margins
    fmt_model_loads='color': 'red', 'linewidth': 1.5,  # Formatting for load arrows
    node_supports=True,     # Display node supports
    truss_node_offset=0.05, # Offset for truss elements
    ax=None                 # Matplotlib axis, None to use current axis
)

# Plot deformations (scaled) after analysis
opsv.plot_defo()
# Optional: plot deformations with a scale factor
# opsv.plot_defo(sfac)
# Optional: customize plot format for deformation visualization
# fmt_interp = 'color': 'blue', 'linestyle': 'solid', 'linewidth': 1.2, 'marker': '.', 'markersize': 6
# opsv.plot_defo(sfac, fmt_interp=fmt_interp)

# Plot internal force diagrams: N (axial), V (shear), M (moment)
sfacN, sfacV, sfacM = 5.e-5, 5.e-5, 5.e-5  # Scale factors for internal force diagrams
# Plot axial force distribution
opsv.section_force_diagram_2d('N', sfacN)
plt.title('Axial force distribution')
# Plot shear force distribution
opsv.section_force_diagram_2d('T', sfacV)
plt.title('Shear force distribution')
# Plot bending moment distribution
opsv.section_force_diagram_2d('M', sfacM)
plt.title('Bending moment distribution')

# Show all plots
plt.show()

# Exit the program
exit()
```

Fig. 19. (*continued*).

ICL Part 1

```
[Zero-shot prompt]:
Generate python code, use openseespy, opsvis and matplotlib. Please only provide code for importing necessary
libraries,
```

Fig. 20. ICL Part 1.

```
and re-generate the code for import opsvis, the code you provide is wrong.

[Python code]:
# Import necessary libraries
import openseespy.opensees as ops
import opsvis as opsv
import matplotlib.pyplot as plt

[Zero-shot prompt]:
Generate code for clearing any existing model.

[Python code]:
# Clear any existing OpenSees model
ops.wipe()

[Zero-shot prompt]:
Define a 2D model with 3 degrees of freedom per node (DOF).

[Python code]:
# Define a 2D model with 3 degrees of freedom per node
ops.model('basic', '-ndm', 2, '-ndf', 3)

[Zero-shot prompt]:
Define column and girder lengths as 4 and 6.
The variable names of column length and girder length are changed to colL and girL.

[Python code]:
# Define column and girder lengths
colL = 4.0  # Length of the columns
girL = 6.0  # Length of the girders

[Zero-shot prompt]:
The cross-sectional areas of column and girder are defined as Acol and Agir with values 2e-3 and 6e-3.
The moments of inertia of column and girder are defined as IzCol and IzGir with values 1.6e-5 and 5.4e-5.

[Python code]:
# Define cross-sectional areas for column and girder
Acol = 2e-3  # Cross-sectional area of the column
Agir = 6e-3  # Cross-sectional area of the girder
# Define moments of inertia for column and girder
IzCol = 1.6e-5  # Moment of inertia of the column
IzGir = 5.4e-5  # Moment of inertia of the girder

[Zero-shot prompt]:
The Young's modulus (E) is 200e9.

[Python code]:
# Define Young's modulus
E = 200e9  # Young's modulus in Pascals (N/m^2)

[One-shot prompt]:
Please create material properties dictionary for each element.
For example, if we have one column element, I will write code like this
Ep = 1: [E, Acol, IzCol]
If element 1 is column element and element 2 is girder element, I will write code like this
Ep = 1: [E, Acol, IzCol], 2: [E, Acol, IzCol]
Now, element 1 and element 2 are column elements and element 3 is girder element. Please write the corresponding code.

[Python code]:
# Material properties dictionary for each element
Ep =
1: [E, Acol, IzCol],  # Element 1 is a column
2: [E, Acol, IzCol],  # Element 2 is a column
3: [E, Agir, IzGir]   # Element 3 is a girder


[One-shot prompt]:
Please define node coordinates.
For example, if node 1 is at (0, 0), I will write code like this
ops.node(1, 0, 0)
Now we have 4 nodes, their coordinates are (0, 0), (0, colL), (girL, 0), (girL, colL).

[Python code]:
# Define node coordinates
ops.node(1, 0, 0)          # Node 1 at (0, 0)
ops.node(2, 0, colL)       # Node 2 at (0, colL)
ops.node(3, girL, 0)       # Node 3 at (girL, 0)
ops.node(4, girL, colL)    # Node 4 at (girL, colL)
```

Fig. 20. (*continued*).

```
[One-shot prompt]:
Please define boundary conditions (supports).
For example, if we fix node 1 in all 3 DOFs (x, y, rotation), I will write code like this
ops.fix(1, 1, 1, 1)
Now, I want to fix all 3 DOFs for node 1 and fix node 3 in x and y, but allow rotation.

[Python code]:
# Define boundary conditions (supports)
ops.fix(1, 1, 1, 1)  # Fix all 3 DOFs (x, y, rotation) for node 1
ops.fix(3, 1, 1, 0)  # Fix x and y for node 3, allow rotation

[Zero-shot prompt]:
Please plot the model before defining elements and add title 'plot_model before defining elements'.

[Python code]:
# Plot the model before defining elements
opsv.plot_model()
# Add title
plt.title('plot_model before defining elements')
```

Fig. 20. (*continued*).

ICL Part 2

```
[Zero-shot prompt]:
please define transformation type for elements (Linear)

[Python code]:
# Define transformation type for elements (Linear)
ops.geomTransf('Linear', 1)  # Transformation ID 1 for linear transformation

[One-shot prompt]:
please define column and girder elements (elastic beam-column elements)
for example, if I want to define a column element between nodes 1 and 2, I will write code like this
ops.element('elasticBeamColumn', 1, 1, 2, Acol, E, IzCol, 1)
Now column element 1 is between nodes 1 and 2, column element 2 is between nodes 3 and 4, girder element 3 is between
nodes 2 and 4

[Python code]:
# Define column and girder elements (elastic beam-column elements)
ops.element('elasticBeamColumn', 1, 1, 2, Acol, E, IzCol, 1)  # Column element 1 between nodes 1 and 2
ops.element('elasticBeamColumn', 2, 3, 4, Acol, E, IzCol, 1)  # Column element 2 between nodes 3 and 4
ops.element('elasticBeamColumn', 3, 2, 4, Agir, E, IzGir, 1)  # Girder element 3 between nodes 2 and 4

[Zero-shot prompt]:
Define external loads, point load (Px) is 2e3, uniform load (Wy) is -10e3, uniform load (Wx) is 0.
Just define the variable that I gave

[Python code]:
# Define external loads
Px = 2e3     # Point load in the x-direction
Wy = -10e3   # Uniform load in the y-direction
Wx = 0.0     # Uniform load in the x-direction

[One-shot prompt]:
Create dictionary to store element loads
for example, if element 1 has a uniform distributed load, I will write code like this
Ew = 1: ['-beamUniform', Wy, Wx]
Now, write the code for element 3 has a uniform distributed load

[Python code]:
# Create a dictionary to store element loads
Ew =     3: ['-beamUniform', Wy, Wx]  # Element 3 has a uniform distributed load

[Zero-shot prompt]:
Define constant load pattern

[Python code]:
# Define time series for constant loads
ops.timeSeries('Constant', 1)
# Define load pattern using the constant time series
ops.pattern('Plain', 1, 1)

[Zero-shot prompt]:
```

Fig. 21. ICL Part 2.

```
Apply point load at node 2 (Px in x-direction)

[Python code]:
# Apply point load Px at node 2 in the x-direction
ops.load(2, Px, 0.0, 0.0)  # Px applied in x-direction, no load in y and rotation

[One-shot prompt]:
Apply distributed loads on elements
I will give you an example, then refactor the code as the example.
for etag in Ew:
    ops.eleLoad('-ele', etag, '-type', Ew[etag][0], Ew[etag][1], Ew[etag][2])

[Python code]:
# Apply distributed loads on the elements using the refactored code
for etag in Ew:
    ops.eleLoad('-ele', etag, '-type', Ew[etag][0], Ew[etag][1], Ew[etag][2])

[Zero-shot prompt]:
Now, write the code for analysis settings
Apply transformation constraints
Renumber the nodes using Reverse Cuthill-McKee (RCM)
Define the solution algorithm
Convergence test criteria (set as 1e-6, 6, 2)
Use linear algorithm for solving
Control load increments (set as 1)
Define a static analysis
Perform the analysis (set as 1)
set constraints as 'Transformation'

[Python code]:
# Analysis settings
# Apply transformation constraints
ops.constraints('Transformation')
# Renumber nodes using Reverse Cuthill-McKee (RCM)
ops.numberer('RCM')
# Define the solution algorithm
ops.system('BandGeneral')
# Define convergence test criteria (tolerance, max iterations, print flag)
ops.test('NormUnbalance', 1e-6, 6, 2)
# Use a linear algorithm for solving
ops.algorithm('Linear')
# Control load increments (set as 1)
ops.integrator('LoadControl', 1.0)
# Define a static analysis
ops.analysis('Static')
# Perform the analysis (1 step)
ops.analyze(1)
```

Fig. 21. (*continued*).

ICL Part 3

```
[Zero-shot prompt]:
Print the model data. Directly use the function from ops to print the model data.

[Python code]:
# Print the entire model data using OpenSeesPy function
ops.printModel()

[Zero-shot prompt]:
Plot the model after defining elements and add title 'plot_model after defining elements'.

[Python code]:
# Plot the model after defining elements
opsv.plot_model()
# Add title to the plot
plt.title('plot_model after defining elements')

[One-shot prompt]:
Plot the applied loads on the model in 2D.
Number of points along each element is 10.
Scale factor for loads is 1.
Width and height of the figure are 10 and 5.
Left, bottom, right, top margins are 0.1, 0.1, 0.9, 0.9.
Formatting for load arrows: color is red and linewidth is 1.5.
```

Fig. 22. ICL Part 3.

```
Display node supports.
Offset for truss elements set as 0.05.
Matplotlib axis: None to use current axis.
Directly use function plot_loads_2d from opsv.
I will provide the format of the function opsv.plot_loads_2d:
opsv.plot_loads_2d(nep=, sfac=, fig_wi_he=(,), fig_lbrt=(,,,),
fmt_model_loads='color': '', 'linewidth': , node_supports=, truss_node_offset=, ax=)

[Python code]:
# Plot the applied loads in 2D using the provided function format
opsv.plot_loads_2d(
    nep=10,  # Number of points along each element
    sfac=1,  # Scale factor for loads
    fig_wi_he=(10, 5),  # Width and height of the figure
    fig_lbrt=(0.1, 0.1, 0.9, 0.9),  # Left, bottom, right, top margins
    fmt_model_loads='color': 'red', 'linewidth': 1.5,  # Formatting for load arrows
    node_supports=True,  # Display node supports
    truss_node_offset=0.05,  # Offset for truss elements
    ax=None  # Use current axis (None)
)

[Zero-shot prompt]:
Plot deformations (scaled) after analysis.
Directly use function plot_defo from opsv.
The scale factors for internal force programs sfacN, sfacV, sfacM are all 5e-5.

[Python code]:
# Plot the scaled deformations after analysis using the plot_defo function
opsv.plot_defo(sfacN=5e-5, sfacV=5e-5, sfacM=5e-5)

[Zero-shot prompt]:
Use section_force_diagram_2d from opsv to draw axial, shear, and moment diagrams and add titles to each diagram.

[Python code]:
# Draw the axial force diagram
fig1, ax1 = plt.subplots()
opsv.section_force_diagram_2d(etype="N", sfacN=5e-5, ax=ax1)  # Axial force (N)
ax1.set_title("Axial Force Diagram")

# Draw the shear force diagram
fig2, ax2 = plt.subplots()
opsv.section_force_diagram_2d(etype="V", sfacV=5e-5, ax=ax2)  # Shear force (V)
ax2.set_title("Shear Force Diagram")

# Draw the moment diagram
fig3, ax3 = plt.subplots()
opsv.section_force_diagram_2d(etype="M", sfacM=5e-5, ax=ax3)  # Moment (M)
ax3.set_title("Moment Diagram")

# Show the plots
plt.show()

[Zero-shot prompt]:
Exit the program. Directly use exit().

[Python code]:
# Exit the program
exit()
```

Fig. 22. (*continued*).

ICL Part 4

```
[Space reasoning instruction]

[Direction reasoning]
In the problem description, if you are instructed to apply loads on the left side of the structure, identify all
nodes (excluding support nodes).
The nodes with the smallest x-coordinate represent the left side, and the loads should be applied to these nodes.
Similarly:
- For the right side, apply loads to the nodes with the largest x-coordinate.
- For the top side, apply loads to the nodes with the largest y-coordinate.
- For the bottom side, apply loads to the nodes with the smallest y-coordinate.

[Number reasoning]
```

Fig. 23. ICL Part 4.

```
In the problem description, ensure that when referencing 5 columns, there are exactly 5 vertical members (columns)
defined as elements.
Similarly, when referencing 5 girders, confirm that there are exactly 5 horizontal members (girders) defined as
elements.

[Space rationality reasoning]
Unless otherwise specified, the y-axis coordinate of the support is generally set to 0.
Ensure that the two nodes of a horizontal member (column) have the same y-axis coordinate,
and ensure that the two nodes of a vertical member (girder) have the same x-axis coordinate.
If the problem description does not mention diagonal members, no element should have two nodes with both x and y
coordinates differing when defining elements.

[Distributed loading direction reasoning]
When applying a distributed load to an element, if the load direction is inward, a negative sign should be added to
the load value.
Additionally, the treatment depends on the coordinates of the element's nodes.
If the x-coordinate of the starting node is less than that of the ending node, no adjustment is needed.
However, if the x-coordinate of the starting node is greater than that of the ending node, a negative sign must be
added to the distributed load when applying it to the corresponding element.

For example, consider the following nodes and elements:

ops.node(3, 0, 4.0)     # Node 3 at (0, 4.0)
ops.node(4, 8.0, 4.0)   # Node 4 at (8.0, 4.0)
ops.node(5, 4.0, 7.0)   # Node 5 at (4.0, 7.0)

ops.element('elasticBeamColumn', 3, 3, 5, 6e-3, 2e11, 5.4e-5, 1)  # Diagonal element 3 between nodes 3 and 5
ops.element('elasticBeamColumn', 4, 4, 5, 6e-3, 2e11, 5.4e-5, 1)  # Diagonal element 4 between nodes 4 and 5

When applying an inward distributed load on elements 3 and 4:

Define the external load:
Wy = -1e4  # Uniformly distributed load in the inward direction

For **element 3**:
The start node is node 3, and the end node is node 5.
The x-coordinate of node 3 (0) is less than the x-coordinate of node 5 (4.0).
Therefore, we directly apply Wy to the element.

For **element 4**:
The start node is node 4, and the end node is node 5.
The x-coordinate of node 4 (8.0) is greater than the x-coordinate of node 5 (4.0).
Thus, we apply -Wy to the element to account for the reversed coordinate order.

The distributed loads for the elements can be specified as:

Ew =
3: ['-beamUniform', Wy, 0],   # Element 3 has a uniform distributed load
4: ['-beamUniform', -Wy, 0]   # Element 4 has a uniform distributed load


[Independent element reasoning]
When defining elements, please ensure that there are no nodes located within an element.
```

Fig. 23. (*continued*).

Question 12

```
How does a two-story two-bay 2D frame, the first bay has two stories and the second bay has one story, the frame
consists of 5 vertical columns (4 meters in height each) and 3 horizontal girders (6 meters in length each), behave
under the uniform vertical distributed load of 1e4 N/m along each girder? Consider elastic material properties with
Young's modulus of 2e11 Pa, column cross-sectional area of 2e-3 m^2, girder cross-sectional area of 6e-3 m^2, column
moment of inertia of 1.6e-5 m^4, and girder moment of inertia of 5.4e-5 m^4, and all supports are fixed. What are the
resulting deformations and internal forces (axial force, shear force, and bending moment) within the frame?
```

Fig. 24. Question 12.

Failure case 1: GPT-4o's response without providing ICL template

Question: Solve the following problem: How does a simple 2D frame, consisting of two vertical columns (4e0 meters in height) and one horizontal girder (6e0 meters in length), behave under the combined effects of a horizontal point load of 2e3 N at the top of one column and a uniform vertical distributed load of 1e4 N/m along the girder, considering elastic material properties with a Young's modulus E of 2e11 Pa, column cross-sectional area of 2e-3 mˆ2, girder cross-sectional area of 6e-3 mˆ2, column moment of inertia Ic of 1.6e-5 mˆ4, and girder moment of inertia Ig of 5.4e-5 mˆ4, all supports are fixed and what are the resulting deformations and internal forces (axial, shear, and moment) in the frame?

Answer:
Problem Statement How does a simple 2D frame, consisting of two vertical columns (4.0 meters in height) and one horizontal girder (6.0 meters in length), behave under the combined effects of a horizontal point load of 2×10^3 N at the top of one column and a uniform vertical distributed load of 1×10^4 N/m along the girder? Assume the frame is made of an elastic material with the following properties:

- Young's modulus: $E = 2 \times 10^{11}$ Pa
- Column cross-sectional area: $A_c = 2 \times 10^{-3}$ m^2
- Girder cross-sectional area: $A_g = 6 \times 10^{-3}$ m^2
- Column moment of inertia: $I_c = 1.6 \times 10^{-5}$ m^4
- Girder moment of inertia: $I_g = 5.4 \times 10^{-5}$ m^4
- All supports are fixed.

Determine the resulting deformations and internal forces (axial, shear, and moment) in the frame.
Qualitative Behavior *Effect of Horizontal Load*

- The frame experiences lateral sway due to the horizontal load.
- The top of the loaded column deflects horizontally.
- Axial forces develop in the girder and columns.

Effect of Vertical Uniform Load

- The girder bends downward, with maximum bending at midspan.
- Shear and moment reactions are induced at the column bases.
- Axial forces arise in the columns due to vertical load.

Deflection Estimates *Horizontal Deflection at Top of Left Column*

$$\delta_H = \frac{PH^3}{3EI_c} = \frac{2000 \times 4^3}{3 \times 2 \times 10^{11} \times 1.6 \times 10^{-5}} = \frac{128000}{9.6 \times 10^6} \approx 0.0133\,\text{m} = 13.3\,\text{mm}$$

Vertical Deflection at Midspan of Girder

$$\delta_V = \frac{5wL^4}{384EI_g} = \frac{5 \times 10^4 \times 6^4}{384 \times 2 \times 10^{11} \times 5.4 \times 10^{-5}} = \frac{6.48 \times 10^7}{4.1472 \times 10^9} \approx 0.0156\,\text{m} = 15.6\,\text{mm}$$

Internal Force Estimates *Girder*

- Maximum bending moment: $M_{\max} = \frac{wL^2}{8} = \frac{10^4 \times 6^2}{8} = 45{,}000\,\text{N m}$.
- Shear force at supports: $V = \frac{wL}{2} = 30{,}000\,\text{N}$.
- Axial force due to sway: non-zero.

Columns

- Bending moments at top and bottom due to lateral sway and fixed supports.
- Axial forces: one column in compression, the other in tension.
- Shear forces result from both horizontal and vertical loading.

Summary Table

Quantity	Approximate Value
Horizontal deflection (top of column)	≈ 13.3 mm
Vertical deflection (midspan of girder)	≈ 15.6 mm
Max bending moment in girder	$\approx 45{,}000$ N m
Shear force at girder supports	$\approx 30{,}000$ N
Axial force in girder	Non-zero due to sway
Column moments	Significant at top and base
Axial forces in columns	Opposite (one in tension, one in compression)

Conclusion The frame experiences both vertical and horizontal displacements due to combined loads. Bending, shear, and axial forces are generated in both the girder and columns. For exact analysis, matrix stiffness method or finite element software (e.g., SAP2000, ANSYS) should be used.

Failure case 2: GPT-4o's response without providing ICL template (Use code)

Question: Solve the following problem: How does a simple 2D frame, consisting of two vertical columns (4e0 meters in height) and one horizontal girder (6e0 meters in length), behave under the combined effects of a horizontal point load of 2e3 N at the top of one column and a uniform vertical distributed load of 1e4 N/m along the girder, considering elastic material properties with a Young's modulus E of 2e11 Pa, column cross-sectional area of 2e-3 m^2, girder cross-sectional area of 6e-3 m^2, column moment of inertia Ic of 1.6e-5 m^4, and girder moment of inertia Ig of 5.4e-5 m^4, all supports are fixed and what are the resulting deformations and internal forces (axial, shear, and moment) in the frame? **Use code**.

Answer:
Problem Description We analyze a simple 2D frame consisting of:

- Two vertical columns, each of height 4 m.
- One horizontal girder of length 6 m.
- A horizontal point load of 2×10^3 N at the top of the left column.
- A uniform vertical distributed load of 1×10^4 N/m applied on the girder.

Material and section properties:

- Young's Modulus: $E = 2 \times 10^{11}$ Pa
- Cross-sectional area of columns: 2×10^{-3} m^2
- Cross-sectional area of girder: 6×10^{-3} m^2
- Moment of inertia of columns: 1.6×10^{-5} m^4
- Moment of inertia of girder: 5.4×10^{-5} m^4

All supports are fixed.
Python Code for Structural Analysis

```
import numpy as np
import matplotlib.pyplot as plt

# Frame geometry
H = 4e0
L = 6e0

# Loads
P_horizontal = 2e3
w_vertical = 1e4

# Material properties
E = 2e11

# Section properties
A_column = 2e-3
A_girder = 6e-3
I_column = 1.6e-5
I_girder = 5.4e-5

# Nodes: 0 (left base), 1 (left top), 2 (right top), 3 (right base)
ndof = 3 * 4
elements = [
    {'nodes': (0, 1), 'A': A_column, 'I': I_column, 'E': E},
    {'nodes': (1, 2), 'A': A_girder, 'I': I_girder, 'E': E},
    {'nodes': (2, 3), 'A': A_column, 'I': I_column, 'E': E}
]

coords = {
    0: [0, 0],
    1: [0, H],
    2: [L, H],
    3: [L, 0]
}
```

```
K_global = np.zeros((ndof, ndof))
F_global = np.zeros(ndof)



def element_stiffness(E, A, I, L, angle):
    c = np.cos(angle)
    s = np.sin(angle)
    k = np.array([
        [A*c**2, A*c*s, 0, -A*c**2, -A*c*s, 0],
        [A*c*s, A*s**2, 0, -A*c*s, -A*s**2, 0],
        [0, 0, 12*I/L**3, 0, 0, -6*I/L**2],
        [-A*c**2, -A*c*s, 0, A*c**2, A*c*s, 0],
        [-A*c*s, -A*s**2, 0, A*c*s, A*s**2, 0],
        [0, 0, -6*I/L**2, 0, 0, 4*I/L]
    ])
    return E/L * k

for element in elements:
    n1, n2 = element['nodes']
    x1, y1 = coords[n1]
    x2, y2 = coords[n2]
    dx = x2 - x1
    dy = y2 - y1
    length = np.sqrt(dx**2 + dy**2)
    angle = np.arctan2(dy, dx)

    k_local = element_stiffness(element['E'], element['A'], element['I'],
    length, angle)
    dof_map = [3*n1, 3*n1+1, 3*n1+2, 3*n2, 3*n2+1, 3*n2+2]

    for i in range(6):
        for j in range(6):
            K_global[dof_map[i], dof_map[j]] += k_local[i, j]

F_global[4] -= w_vertical * L / 2
F_global[7] -= w_vertical * L / 2
F_global[3] += P_horizontal

fixed_dofs = [0, 1, 2, 9, 10, 11]
free_dofs = list(set(range(ndof)) - set(fixed_dofs))

K_ff = K_global[np.ix_(free_dofs, free_dofs)]
F_f = F_global[free_dofs]
u_f = np.linalg.solve(K_ff, F_f)

u = np.zeros(ndof)
u[free_dofs] = u_f
```

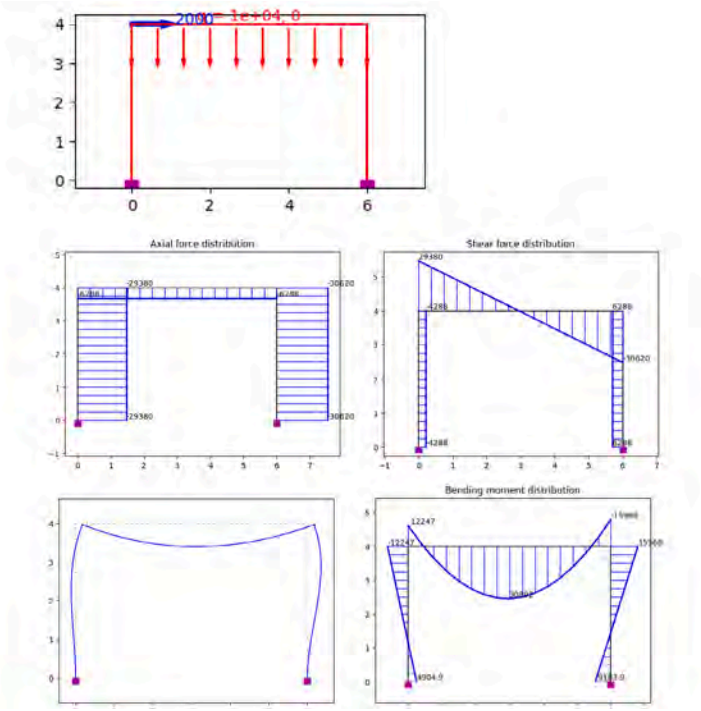
Nodal Displacement Results

Node	u (m)	v (m)	θ (rad)
0	0.000000×10^{0}	0.000000×10^{0}	0.0
1	-5.334188×10^{27}	3.266248×10^{11}	0.0
2	-5.334188×10^{27}	-3.266248×10^{11}	0.0
3	0.000000×10^{0}	0.000000×10^{0}	0.0

Ground Truth Solution of Example 1

Node Displacements	
Node 1	(0, 0, 0)
Node 2	(0.00203106, -0.000293798, -0.00458888)
Node 3	(0, 0, 0)
Node 4	(0.00199962, -0.000306202, 0.0042402)
Element End Forces (P, V, M)	
Element 1 - End 1	(29379.8, -4288.01, -4904.93)
Element 1 - End 2	(-29379.8, 4288.01, -12247.1)
Element 2 - End 1	(30620.2, 6288.01, 9183.87)
Element 2 - End 2	(-30620.2, -6288.01, 15968.2)
Element 3 - End 1	(6288.01, 29379.8, 12247.1)
Element 3 - End 2	(-6288.01, 30620.2, -15968.2)

Ground Truth Schematic



Ground truth schematic for Example 1.

Appendix D. Additional experimental details

D.1. Failure modes

See Table 10.

D.2. Pass@k vs. Pass^k

Pass^k metric. For tasks such as code generation and solving mathematical problems with reliable verification procedures, the community has adopted the pass@k (pass at k) metric, defined as the probability that at least one of k i.i.d. task trials succeeds. This metric reflects the potential of LLMs to enable the *discovery* of solutions through scaling of inference-time compute [35]. For real-world engineering tasks requiring *reliability* and *consistency*, such as structural engineering, the pass^k (pass hat k) metric was introduced [43], defined as the probability that all k i.i.d. task trials are successful, averaged across tasks. Therefore, if a task is executed for n trials and c of them are successful ($r = 1$), the unbiased estimates of pass^k and pass@k are given by:

$$\text{pass}\hat{}\text{k} = \mathbb{E}_{\text{task}}\left[\binom{c}{k}\Big/\binom{n}{k}\right], \quad \text{pass@k} = 1 - \mathbb{E}_{\text{task}}\left[\binom{n-c}{k}\Big/\binom{n}{k}\right].$$

Table 10
Per-problem results (S: success, F: fail) across all evaluated LLMs on 20 structural analysis problems.

Problem	GPT-5.4		GPT-4o (2024-11-20)		GPT-4-turbo		Gemini 1.5 Pro		Llama-3.3-70b-versatile	
	S/F	Note	S/F	Note	S/F	Note	S/F	Note	S/F	Note
1	S		S		S		S		S	
2	S		S		S		S		F	1
3	S		S		S		S		S	
4	S		S		S		S		S	
5	S		S		S		S		S	
6	S		S		S		S		F	2
7	S		S		S		S		F	1
8	S		S		S		S		F	1
9	S		S		S		F	4	F	1
10	S		S		S		F	1	F	1
11	S		S		S		S		F	1
12	S		S		S		S		F	1
13	S		S		S		S		F	1
14	S		S		F	1	F	1	F	1
15	S		S		S		S		S	
16	S		S		S		S		S	
17	S		S		S		S		F	1
18	S		S		S		S		F	1
19	S		S		F	3	S		F	3
20	S		S		F	3	F	3	F	3

Note: Failure modes — 1: Wrong geometry, 2: Wrong support definition, 3: Wrong load definition, 4: Coding error.

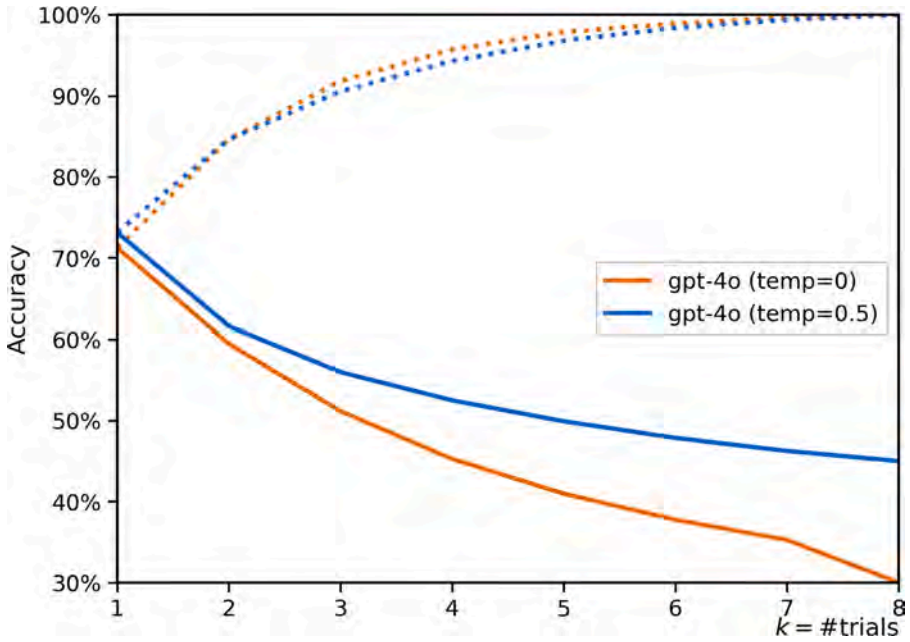


Fig. 25. Pass^k (solid line) and Pass@k (dashed line) on SAWP-20.

As shown in Fig. 25, the performance of LLMs on the pass@k metric increases and converges to 100%, while the chance of reliably and consistently solving the task multiple times significantly drops as the number of trials k increases on the pass^k metric. Based on different temperature settings, the framework based on GPT-4o has a > 70% average task success, while pass^8 drops to < 50%. In real-world structural engineering scenarios, as LLMs have not yet been widely adopted in practice, it is still not well established which metrics should be used to evaluate their performance on structural engineering tasks. On the one hand, some sub-tasks of structural engineering consulting, such as code-based structural analysis like the problems proposed in this paper, can be treated as code generation tasks, meaning that LLM performance can be improved via test-time scaling. On the other hand, all tasks within structural engineering consulting can have real-world impacts within a short period of time. For example, a structural engineer may use this framework to perform structural analysis, then use the results to size members, and these sizes are incorporated into structural drawings issued for construction, after which fabricators immediately produce the members based on the engineer's decisions. Based on this, it is necessary to ensure that LLMs in this setting are reliable and consistent. Therefore, in real-world scenarios, it is important and challenging not only to build LLM agents with high average success (pass^1), but also to ensure robustness and consistency (pass^k trend) [43].

Data availability

The data and code have been made available via a link in the paper.

References

[1] Frank McKenna, OpenSees: a framework for earthquake engineering simulation, Comput. Sci. Eng. 13 (4) (2011) 58–66, http://dx.doi.org/10.1109/MCSE.2011.66.

[2] Raffaello Antonutti, Christophe Peyrard, Atilla Incecik, David Ingram, Lars Johanning, Dynamic mooring simulation with Code_Aster with application to a floating wind turbine, Ocean Eng. 151 (2018) 366–377, http://dx.doi.org/10.1016/j.oceaneng.2017.11.018.

[3] Guido Dhondt, Calculix crunchix user's manual version 2.12, 2017, Munich, Germany, https://www.dhondt.de/ccx_2.12.pdf. (Accessed 21 September 2017).

[4] Laurent Pasticier, Claudio Amadio, Massimo Fragiacomo, Non-linear seismic analysis and vulnerability evaluation of a masonry building by means of the SAP2000 V. 10 code, Earthq. Eng. Struct. Dyn. 37 (3) (2008) 467–485, http://dx.doi.org/10.1002/eqe.770.

[5] Gong-Dong Wang, Stephen Kirwa Melly, Three-dimensional finite element modeling of drilling CFRP composites using Abaqus/CAE: a review, Int. J. Adv. Manuf. Technol. 94 (2018) 599–614, http://dx.doi.org/10.1007/s00170-017-0754-7.

[6] P.C. Kohnke, Ansys, in: Finite Element Systems: A Handbook, Springer, 1982, pp. 19–25, http://dx.doi.org/10.1007/978-3-662-07229-5_2.

[7] A. Elkady, FM-2D - open-source platform for the 2-dimensional numerical modeling and seismic analysis of buildings, SoftwareX 17 (2022) 100927, http://dx.doi.org/10.1016/j.softx.2021.100927.

[8] Soheil Radfar, Mohammad Afrazi, Arash Fakhournezhad, Ali Akbar Golshani, Finite element analysis of a 2D truss using MATLAB and OpenSees, in: Proceedings of the 5th International Congress on Civil Engineering, Architecture and Urban Development, 2017, https://www.researchgate.net/publication/341495069_Finite_Element_Analysis_of_a_2D_Truss_Using_MATLAB_and_OpenSees.

[9] Astha Verma, Neeraj Chandra, Pratik Kumar Singh, Vishal Kadian, Suraj Rai, Evaluation of wind effects on high-rise buildings using ansys CFX, J. Eng. Res. Appl. 2 (2023) 01–11, http://dx.doi.org/10.55953/JERA.2023.2101.

[10] Farzad Hejazi, Hojjat Mohammadi Esfahani, Solving Complex Problems for Structures and Bridges Using ABAQUS Finite Element Package, CRC Press, 2021, http://dx.doi.org/10.1201/9781003213369.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, Attention is all you need, 2017, http://dx.doi.org/10.48550/arXiv.1706.03762, CoRR abs/1706.03762, URL http://arxiv.org/abs/1706.03762.

[12] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, Exploring the limits of transfer learning with a unified text-to-text transformer, J. Mach. Learn. Res. 21 (140) (2020) 1–67, http://dx.doi.org/10.48550/arXiv.1910.10683.

[13] William Peebles, Saining Xie, Scalable diffusion models with transformers, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2023, pp. 4195–4205, http://dx.doi.org/10.1109/ICCV51070.2023.00387.

[14] Ross Taylor, Marcin Kardas, Guillem Cucurull, Thomas Scialom, Anthony Hartshorn, Elvis Saravia, Andrew Poulton, Viktor Kerkez, Robert Stojnic, Galactica: A large language model for science, 2022, http://dx.doi.org/10.48550/arXiv.2211.09085, arXiv preprint arXiv:2211.09085.

[15] Yingyu Liang, Jiangxuan Long, Zhenmei Shi, Zhao Song, Yufa Zhou, Beyond linear approximations: A novel pruning approach for attention matrix, in: The Thirteenth International Conference on Learning Representations, 2025, http://dx.doi.org/10.48550/arXiv.2410.11261.

[16] Xuan Shen, Zhao Song, Yufa Zhou, Bo Chen, Jing Liu, Ruiyi Zhang, Ryan A. Rossi, Hao Tan, Tong Yu, Xiang Chen, Yufan Zhou, Tong Sun, Pu Zhao, Yanzhi Wang, Jiuxiang Gu, Numerical pruning for efficient autoregressive models, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2025, http://dx.doi.org/10.1609/aaai.v39i19.34249.

[17] Yingyu Liang, Zhenmei Shi, Zhao Song, Yufa Zhou, Differential privacy of cross-attention with provable guarantee, 2024, http://dx.doi.org/10.48550/arXiv.2407.14717, arXiv preprint arXiv:2407.14717.

[18] Samuel Schmidgall, Rojin Ziaei, Carl Harris, Eduardo Reis, Jeffrey Jopling, Michael Moor, AgentClinic: a multimodal agent benchmark to evaluate AI in simulated clinical environments, 2024, http://dx.doi.org/10.48550/arXiv.2405.07960, arXiv preprint arXiv:2405.07960.

[19] Yijia Xiao, Edward Sun, Di Luo, Wei Wang, TradingAgents: Multi-agents LLM financial trading framework, 2024, http://dx.doi.org/10.48550/arXiv.2412.20138, arXiv preprint arXiv:2412.20138.

[20] Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, Tao Yu, Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows, in: International Conference on Learning Representations, 2025, http://dx.doi.org/10.48550/arXiv.2411.07763.

[21] Saket Kumar, Abul Ehtesham, Aditi Singh, Tala Talaei Khoei, Architectural flaw detection in civil engineering using GPT-4, 2024, http://dx.doi.org/10.48550/arXiv.2410.20036, arXiv preprint arXiv:2410.20036.

[22] Laura Cruz-Castro, Gabriel Castelblanco, Pavlo Antonenko, LLM-based system for technical writing real-time review in urban construction and technology, Proc. 60th Annu. Assoc. Sch. 5 (2024) 130–138, http://dx.doi.org/10.29007/d9j3.

[23] Chao Fan, Qipei Mei, Xiaonan Wang, Xinming Li, ErgoChat: a visual query system for the ergonomic risk assessment of construction workers, 2024, http://dx.doi.org/10.48550/arXiv.2412.19954, arXiv preprint arXiv:2412.19954.

[24] Isaac Joffe, George Felobes, Youssef Elgouhari, Mohammad Talebi Kalaleh, Qipei Mei, Ying Hei Chui, The framework and implementation of using large language models to answer questions about building codes and standards, J. Comput. Civ. Eng. (2025) http://dx.doi.org/10.1061/JCCEE5.CPENG-6037.

[25] Chuan Tian, Yilei Zhang, Optimizing collaboration of LLM based agents for finite element analysis, 2024, http://dx.doi.org/10.48550/arXiv.2408.13406, arXiv preprint arXiv:2408.13406.

[26] Sizhong Qin, Hong Guan, Wenjie Liao, Yi Gu, Zhe Zheng, Hongjing Xue, Xinzheng Lu, Intelligent design and optimization system for shear wall structures based on large language models and generative artificial intelligence, J. Build. Eng. 95 (2024) 109996, http://dx.doi.org/10.1016/j.jobe.2024.109996.

[27] Minjie Zhu, Frank McKenna, Michael H. Scott, OpenSeesPy: Python library for the OpenSees finite element framework, SoftwareX 7 (2018) 6–11, http://dx.doi.org/10.1016/j.softx.2017.10.009.

[28] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al., Gpt-4 technical report, 2023, http://dx.doi.org/10.48550/arXiv.2303.08774, arXiv preprint arXiv:2303.08774.

[29] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al., Gpt-4o system card, 2024, http://dx.doi.org/10.48550/arXiv.2410.21276, arXiv preprint arXiv:2410.21276.

[30] OpenAI, Introduction to GPT-5.4 thinking, 2026, https://deploymentsafety.openai.com/gpt-5-4-thinking/introduction. (Accessed 30 March 2026).

[31] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al., The llama 3 herd of models, 2024, http://dx.doi.org/10.48550/arXiv.2407.21783, arXiv preprint arXiv:2407.21783.

[32] Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al., Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context, 2024, http://dx.doi.org/10.48550/arXiv.2403.05530, arXiv preprint arXiv:2403.05530.

[33] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al., Language models are few-shot learners, Adv. Neural Inf. Process. Syst. 33 (2020) 1877–1901, http://dx.doi.org/10.48550/arXiv.2005.14165.

[34] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, et al., A survey on in-context learning, in: Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2024, pp. 1107–1128, http://dx.doi.org/10.18653/v1/2024.emnlp-main.64.

[35] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al., Evaluating large language models trained on code, 2021, http://dx.doi.org/10.48550/arXiv.2107.03374, arXiv preprint arXiv:2107.03374.

[36] Taicheng Guo, Bozhao Nan, Zhenwen Liang, Zhichun Guo, Nitesh Chawla, Olaf Wiest, Xiangliang Zhang, et al., What can large language models do in chemistry? a comprehensive benchmark on eight tasks, Adv. Neural Inf. Process. Syst. 36 (2023) 59662–59688, http://dx.doi.org/10.52202/075280-2607.

[37] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al., Chain-of-thought prompting elicits reasoning in large language models, Adv. Neural Inf. Process. Syst. 35 (2022) 24824–24837, http://dx.doi.org/10.52202/068431-1800.

[38] Jihan Yang, Shusheng Yang, Anjali W Gupta, Rilyn Han, Li Fei-Fei, Saining Xie, Thinking in space: How multimodal large language models see, remember, and recall spaces, 2024, http://dx.doi.org/10.48550/arXiv.2412.14171, arXiv preprint arXiv:2412.14171.

[39] Sayash Kapoor, Benedikt Stroebl, Zachary S Siegel, Nitya Nadgir, Arvind Narayanan, Ai agents that matter, 2024, http://dx.doi.org/10.48550/arXiv.2407.01502, arXiv preprint arXiv:2407.01502.

[40] Haoran Liang, Yufa Zhou, Mohammad Talebi Kalaleh, Qipei Mei, Automating structural engineering workflows with large language model agents, 2025, http://dx.doi.org/10.48550/arXiv.2510.11004, arXiv preprint arXiv:2510.11004.

[41] Zhuosheng Zhang, Aston Zhang, Mu Li, Alex Smola, Automatic chain of thought prompting in large language models, 2022, http://dx.doi.org/10.48550/arXiv.2210.03493, arXiv preprint arXiv:2210.03493.

[42] Haitao Li, Qian Dong, Junjie Chen, Huixue Su, Yujia Zhou, Qingyao Ai, Ziyi Ye, Yiqun Liu, Llms-as-judges: a comprehensive survey on llm-based evaluation methods, 2024, http://dx.doi.org/10.48550/arXiv.2412.05579, arXiv preprint arXiv:2412.05579.

[43] Shunyu Yao, Noah Shinn, Pedram Razavi, Karthik Narasimhan, τ-Bench: A benchmark for tool-agent-user interaction in real-world domains, 2024, http://dx.doi.org/10.48550/arXiv.2406.12045.