Article

Not peer-reviewed version

Human-AI Teaming in Structural Analysis: A Model Context Protocol Approach for Explainable and Accurate Generative AI

Carlos Avila [*] , Daniel Ilbay , David Rivera [*]

Posted Date: 23 July 2025

doi: 10.20944/preprints202507.1968.v1

Keywords: generative AI-assisted structural analysis; model context protocol; human–machine interaction; computational efficiency; LLM–FEM integration

Article

Human-AI Teaming in Structural Analysis: A Model Context Protocol Approach for Explainable and Accurate Generative AI

Carlos Avila[1,*], Daniel Ilbay[2] and David Rivera[1,*]

[1] Universidad UTE, Facultad de Ciencias, Ingeniería y Construcción, Grupo de Investigación de Energía, Minas y Agua (GIEMA), Quito 170527, Ecuador.
[2] Universidad UTE, Facultad de Ciencias, Ingeniería y Construcción, Ingeniería Civil, Quito 170527, Ecuador
* Correspondence: carlos.avila@ute.edu.ec; edgar.rivera@ute.edu.ec

Abstract

The integration of large language models (LLMs) into structural engineering workflows presents both a transformative opportunity and a critical challenge. While LLMs enable intuitive, natural-language interactions with complex data, their limited arithmetic reasoning, contextual fragility, and lack of verifiability constrain their application in safety-critical domains. This study introduces a novel automation pipeline that couples generative AI with finite element modelling through the Model Context Protocol (MCP)—a modular, context-aware architecture that compliments language interpretation from structural computation. By interfacing GPT-4 with OpenSeesPy via MCP (JSON schemas, API interfaces, communication standards), the system allows engineers to specify and evaluate 3D frame structures using conversational prompts, while ensuring computational fidelity and code compliance. Across four case studies, the GPT+MCP framework demonstrated predictive accuracy for key structural parameters, with deviations under 1.5% compared to reference solutions produced using conventional finite element analysis workflows. In contrast, unconstrained LLM use produces deviations exceeding 400%. The architecture supports reproducibility, traceability, and rapid analysis cycles (6–12 seconds), enabling real-time feedback for both design and education. This work establishes a reproducible framework for trustworthy AI-assisted analysis in engineering, offering a scalable foundation for future developments in optimisation, and regulatory automation.

Keywords: generative AI-assisted structural analysis; model context protocol; human–machine interaction; computational efficiency; LLM–FEM integration

Introduction

Complexity has become central to modern engineering, no longer a peripheral issue. Each decision—across systems from power grids to software—is entangled in socio-technical networks [1–3]. Emerging technologies and AI amplify this interconnectedness. When misunderstood, they introduce risk rather than insight [4]. In the past, engineers often relied on trial and error, lacking a formal understanding of complex interdependencies [5]. Seeing complexity as a problem often leads to hesitation and missed opportunities for learning. A more constructive approach is to view complexity as a valuable part of the engineering process. Nowadays, with clear principles, reliable data, and the support of AI tools, it is possible to reduce unnecessary complications while keeping the detail needed for robust and flexible solutions. This approach supports better decisions, faster progress, more effective collaboration across different groups and effective human-machine interactions.

AI emulates human cognition by manipulating structured and unstructured data. Generative AI extends this capacity by synthesizing new solutions, inferring goals, and navigating varied data formats [6]. Large language models (LLMs) emerged from this progress. Trained on massive corpora, they interpret linguistic input, generate text and code, and perform tasks with minimal fine-tuning. In structural engineering, LLMs now assist in design analysis by translating domain-specific queries into useful insights [7–9]. However, their outputs remain statistical guesses, not verifiable computations. They frequently hallucinate numbers, misapply logic, or embed biases—unacceptable in engineering contexts [10,11]. Their accuracy depends entirely on the quality and context of the prompt. To mitigate this, engineers must design domain-specific prompts and use frameworks that enforce structured reasoning. Embedding context-awareness and external validation is critical to make generative outputs trustworthy and auditable.

Generative AI models benefit significantly from context-aware integration. A system is context-aware when it tailors responses using information about its environment or task [12]. Traditional AI architectures, though adept at pattern recognition, lack the semantic depth to interpret complex real-world inputs. This is where context engines excel—they simulate human-like understanding by linking meaning across data. The Model Context Protocol (MCP) was developed to formalise these interactions. It connects LLMs with external tools through APIs, allowing structured communication with analytical software [13]. MCP is model-agnostic. It allows AI to read files, call functions, and respond to enrich contextual prompts [14]. This addresses a longstanding challenge in AI systems: maintaining consistent context across distributed tools [15,16]. When deployed within generative AI systems, MCP grounds the model's responses in explicit data and reduces hallucinations. It strengthens accuracy and makes AI tools genuinely useful for domain-specific decision-making.

Generative AI models such as GPT-4, Claude 4, and LLaMA-3 are transforming the construction industry workflows. Their natural-language interfaces streamline documentation and enable rapid knowledge retrieval from case studies and codes [17]. However, without careful curation, their training data may omit edge cases—such as atypical structural systems or extreme climates[18]. As a result, generated advice may not generalise well across projects. For instance, recent studies showed how LLMs can translate structural descriptions into OpenSeesPy scripts [7,18]. But LLMs still cannot verify geometry, assess structural logic, or detect orphaned elements. They lack intrinsic understanding of code compliance, safety margins, and construction feasibility. These limitations, along with dataset biases, underscore the need for moderated, context-constrained AI integration.

Because LLMs are weak in numerical reasoning and code compliance, structural engineering demands controlled integration frameworks that enforce validation and fidelity. Rather than use LLMs for direct calculations, a better strategy is to delegate computations to verified solvers. LLMs should serve as interfaces—interpreting user intent and relaying structured instructions. This study proposes such a system, embedding conversational LLMs within the Model Context Protocol. This framework connects natural-language prompts to finite-element tools like OpenSeesPy (3.7.1, PEER). By enabling transparent feedback and structured inputs, MCP democratises access to advanced simulations. It helps engineers interpret results safely and guards against overconfidence in generative outputs.

2. Methods

2.1. Architecture Workflow

The proposed system uses a modular client–server design rooted in the MCP. It separates natural language processing from simulation execution. This architecture connects complex structural models with intuitive language-based interfaces (Figure 1). Engineers can run advanced simulations without needing to write code or manipulate software directly. The platform uses FastAPI (3.1.1) design to facilitate communication. Through this, engineers submit prompts, receive feedback, and manage simulations in an accessible and scalable environment. By lowering the barrier to entry, the system promotes interdisciplinary collaboration and accelerates design workflows.

The system consists of three layers (Figure 1). First, the Host Application Layer collects user prompts describing geometry and loading. Second, the Communication Layer validates inputs and routes requests. It uses FastAPI to connect the interface with simulation tools. Third, the External Application Layer includes OpenSeesPy, which performs structural simulations. This layer handles seismic analysis, load cases, and geometry creation. The analytical outputs are then returned to the reasoning engine, as a context, to support results interpretation.

The proposed system architecture is structured into three principal layers, each fulfilling a distinct role in the execution pipeline. First, the Host Application Layer allows users to articulate structural configurations and loading scenarios through natural language prompts, submitted via an interactive interface or API client. Second, the Communication Protocol Layer, Implementable using FastAPI, intermediates between user intent and computational execution. It ensures robust input validation, manages tool invocation, and routes data through a centralised orchestration server. Third, the External Application Layer comprises domain-specific tools such as OpenSeesPy, a Python (3.8.x, Python Software Foundation) interface to OpenSees (3.7.1, PEER), which conducts advanced structural simulations, particularly seismic analyses. The analytical outputs are then returned to the reasoning engine to support results interpretation.

This layered setup ensures traceability by tracking each computational step. It also supports scalability through modular components and extensibility for future tools. Engineers can modify or extend the platform without retraining the language model. This makes the system durable and adaptable for long-term use. Potential extensions include dynamic response analysis, fragility curve estimation, and automated code checks. The separation of roles ensures reliable performance across evolving requirements.

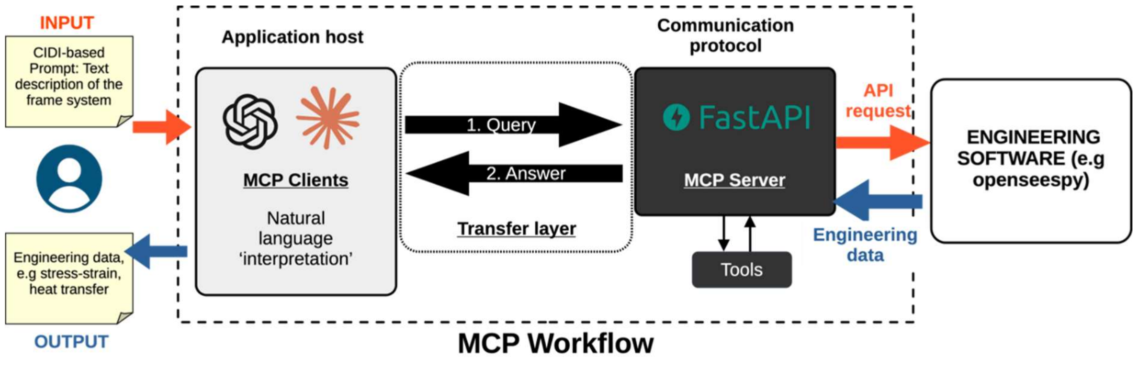

INPUT
CIDI-based Prompt: Text description of the frame system
Application host
Communication protocol
MCP Clients
Natural language 'interpretation'
1. Query
2. Answer
Transfer layer
FastAPI
MCP Server
API request
ENGINEERING SOFTWARE (e.g openseespy)
Engineering data
Tools
Engineering data, e.g stress-strain, heat transfer
OUTPUT
MCP Workflow

Figure 1. MCP Architecture Workflow.

2.2. Implementation of MCP

The MPC rationale implemented in this work is structured into three functional layers that coordinate natural language reasoning with structural simulation tools, enabling an interpretable and modular interface (Table 1). In the Client Layer, ChatGPT (GPT-4, OpenAI) processes user prompts and extracts parameters like geometry, loads, and material properties. It converts these into structured JSON that guides simulation tools. These tools are declared in the Server Layer, which specifies functions, inputs, and execution rules. Results update the context for the LLM hosted in the Client Layer, keeping the interaction coherent. The approach allows autonomous tool use within a verified framework. It enables simulations that are transparent, reproducible, and context-aware—essential features for AI in engineering.

Complementing this, the Server Layer incorporates specialised Python modules—such as seismic loading and static linear analysis tools—that interface with OpenSeesPy via a local API. These modules generate domain-specific outputs, including storey drifts, shear forces, and mode shapes, while maintaining modularity and reusability. The Server Layer manages the execution workflow by validating inputs, sequencing tasks, and handling errors. It returns results in structured, machine-readable JSON, ensuring consistency and traceability. This layered integration not only operationalises language-driven structural analysis but also supports future enhancements such as automated code compliance checks and structural optimization routines.

Table 1. Tool Declaration and Integration Role.

MCP Layer	Role of tools declaration
Client Layer	Hosts the LLM-based application and initiates tool execution requests.
Server Layer	Provides the tools to the Client Layer and executes them upon request.
External Applications	Supply expert context and domain-specific information to support tool execution.

2.3. MCP Client-Server Workflow

The proposed system adopts a REST-based client–server architecture implemented through FastAPI, with a streamlined communication pipeline centred on a unified/analyse endpoint that accepts natural language prompts as POST requests. These two components establish the operational foundation for a modular and scalable workflow that begins with the initialisation of a Model–Context environment, where all simulation and modelling resources are preloaded. Once activated, the prompt is interpreted by ChatGPT, which extracts engineering intent and encodes it into a structured JSON schema defining geometry, materials, and design actions. This structured payload then feeds into a modelling engine that generates a three-dimensional reinforced concrete frame using OpenSeesPy. Subsequent analysis routines simulate static and seismic loading conditions while incorporating code-based constraints. These steps converge to produce critical engineering outputs, including story-level drifts, shear force distributions, and modal responses. Parsed results are evaluated against normative thresholds, such as the Ecuadorean Construction Code NEC-15 [19] and the American Standard for Ensuring Load Safety for All Infrastructure (ASCE 7-22) [20], and subsequently synthesised by the LLM into a coherent technical report. This sequential and modular design ensures deterministic, transparent, and reproducible outputs, while enabling flexible integration with command-line interfaces, interactive dashboards, or external systems, aligning with modern principles of traceable, AI-assisted structural analysis.

2.4. Tool Integration and Execution Logic

The proposed framework integrates advanced modelling, simulation, and validation tools to support structural engineering workflows. While OpenSeesPy was used for its open-source accessibility and seismic analysis capabilities, the architecture is platform-agnostic. Any structural analysis software with API access—open-source or commercial—can be integrated into the MCP architecture. This flexibility enables automated code-based verification and supports diverse engineering environments efficiently.

First, the Modelling Tool enables the construction of detailed 3D structural models by translating LLM-defined parameters into engineering representations, incorporating critical assumptions such as material behaviour and torsional modifiers. Second, the Simulation Tool executes seismic analyses using established code procedures (e.g., ASCE 7-22, NEC-15), ensuring accurate force distribution and job-level isolation for reproducibility. Third, the Parsing and Validation Tool extracts and assesses performance metrics (e.g., inter-storey drift compliance), formatting outputs in a machine-readable structure to support interpretability and traceability.

These three tools converge to support the Tool Invocation Logic, which is orchestrated by the MCP. The MCP architecture ensures correct sequencing, input validation, and dynamic selection of tools based on prompt semantics and LLM reasoning. This orchestration enables the LLM to act not merely as a query interpreter but as an intelligent controller of engineering workflows.

Consequently, the system achieves a stateless, modular, and declarative architecture, where each tool performs a distinct role, and their coordination is dynamically governed to support adaptive, context-aware structural analysis.

2.5. Study case

This study applies to the MCP to establish a structured communication interface between ChatGPT and the OpenSeesPy API, enabling the automated generation of structural analysis models through natural language input. A CIDI-style prompt [21] is formulated to describe a representative three-dimensional frame system, encoding key geometric and mechanical parameters—including member lengths, cross-sectional dimensions, bay spans in the X and Y directions, number of stories, loading conditions, and material properties (Table 2). While the main text details a single reference case to illustrate the methodology, three additional case studies—designed using the same specification logic—are presented in Table 3 to assess the method's generalizability and predictive accuracy.

Table 2. Geometric, mechanical, and loading parameters defining the reference study case.

Category	Parameter	Magnitude
Geometry	Number of Stories	5
	Story Heights	3.0, 2.5, 2.5, 2.5, 2.5 m (from bottom to top)
	Spans in X Direction	4 m, 5 m, 6 m
	Spans in Y Direction	3.5 m, 4.5 m
Sections	Beam Cross-Section	0.30 m × 0.40 m
	Column Cross-Section	0.40 m × 0.40 m
	Cracking Factor (Beams)	0.7
	Cracking Factor (Columns)	0.8
Material	Concrete Young's Modulus	21,458,890.83 kN/m²
Loads	Dead Load	4.9 kN/m²
	Live Load	1.9 kN/m²
	Dead Load Weight Coefficient	1.0
	Live Load Weight Coefficient	0.15
Seismic Parameters	Base Shear Coefficient	0.1488
	Vertical Distribution Coefficient	1.0
	Accidental Torsion Coefficient	0.05
	Drift Amplification Factor (for Inelastic Drift)	6.0
	Maximum Allowable Drift	0.02

Table 3. Basic structural organisation of the study cases.

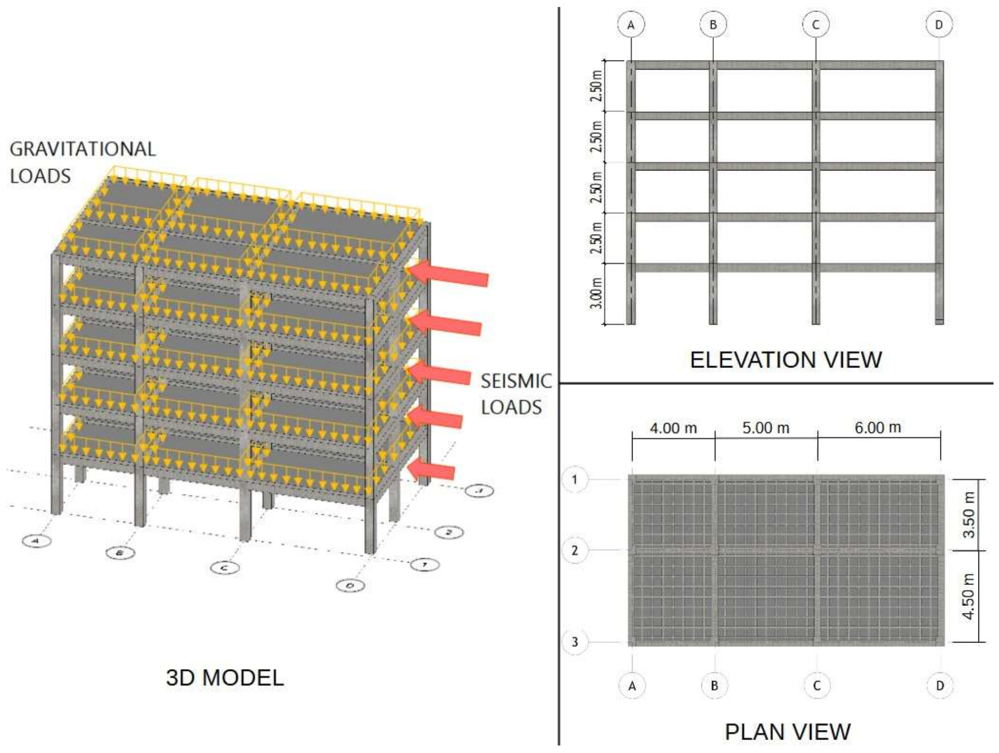
Case ID	No. of Floors	Story Heights (m)	Spans X (m)	Spans Y (m)	Geometry Type
A	2	3.0 – 3.0	4 – 4	4 – 4	Symmetric
B	3	3.0 – 3.0 – 3.0	4 – 5 – 4	3.5 – 4.5	Asymmetric
C	5	3.5 – 2.5 – 2.5 – 2.5 – 2.5	4 – 4 – 4	4 – 4 – 4	Symmetric
D	5	3.5 – 2.5 – 2.5 – 2.5 – 2.5	4 – 5 – 6	3.5 – 4.5	Asymmetric

Note: Case D represents the scenario detailed in the Study Case section.

The defined model is intended for analysis of storey drifts, shear forces, and vibration modes (Table 4). To ensure the correctness of the AI-assisted process, the same structural model is also manually implemented using conventional OpenSees programming techniques and manual modelling in ETABS (20.3.0, Computers and Structures CSI). This enables the establishment of a baseline for comparison and supports validation and verification procedures. All configurations are evaluated with respect to compliance with relevant structural performance criteria, particularly storey drift limitations specified in NEC-15 and ASCE 7-22 standards.

Additionally, the study includes the direct transmission of a structured version of the prompt to ChatGPT without MCP-based context handling. This serves to evaluate the behaviour of the model when lacking contextual constraints and formal validation layers, which informs the necessity of structured mediation and procedural rigour in prompt design. These steps collectively support the methodological aim of ensuring reproducibility and validation in AI-assisted structural modelling workflows.

Table 4. Structural Analysis Case Study: (a) Three-Dimensional Model for OpenSees and ETABS Implementation, and (b) Natural Language Specification via MIDI-Formatted.



(a)

```
prompt = (
    "Context:"
    "You are an expert in structural analysis using natural language and numerical simulation with OpenSeesPy. "
    "The implemented system is capable of interpreting technical prompts and generating automated structural simulations "
    "based on international seismic-resistant design standards."
    "Instructions:"
    "Analyse a three-dimensional reinforced concrete frame, verify code compliance for inter-story drift, "
    "and apply structural optimization if necessary."
    "Details:"
    "The modulus of elasticity for concrete is 21,458,890.83 kN/m². "
    "The structural system has spans of 4, 5, and 6 meters in the X direction, and spans of 3.5 and 4.5 meters in the Y direction. "
    "The structure has 5 stories, with story heights of: 3.0, 2.5, 2.5, 2.5, and 2.5 meters respectively. "
    "Beams have a cross-sectional dimension of 0.3 x 0.4 meters, and columns are 0.40 x 0.45 meters. "
    "Cracking factors are 0.7 for beams and 0.8 for columns. "
    "Dead load is 4.9 Kn/m² and live load is 1.9 kN/m². "
    "The weight coefficients are: 1.0 for dead load, 0.15 for live load, 0.1488 for base shear coefficient, "
    "1.0 for vertical distribution of base shear, 0.05 for accidental torsion, "
    "and a drift amplification factor of 6.0 is applied to estimate inelastic drift. The maximum allowable drift is 0.02."
    "Tasks:"
    "1. Perform linear static seismic analysis using the equivalent lateral force method with OpenSeesPy. "
    "2. Compute maximum displacements and story drifts per level and direction (X and Y). "
    "3. Perform strict numerical validation:"
    " - Iterate through all obtained inelastic drift values. "
    " - For each value, compare it against the allowable maximum (0.02). "
    " - If *at least one value* exceeds 0.02, *you must not state that all values are compliant*. "
    " - Report precisely: story number, direction (X or Y), and the drift value that exceeds the limit. "
    " - Only if *all drifts* are ≤ 0.02, the code compliance can be confirmed. "
    " - Present results in tabular format and be rigorous with numerical precision."
    "4. Also determine floor-by-floor shear forces and vibration modes."
    "5. Structural optimization:"
    " - If any drift exceeds the limit, propose a structural optimisation based on displacements, "
    "    storey drifts, shear forces, and vibration modes. "
    " - Generate 10 alternatives by modifying material properties and section dimensions. "
    " - Evaluate drift for each alternative and present the comparison in tabular format. "
    " - Highlight the configurations that meet code requirements and provide better structural efficiency."
    "Intent:"
    "Generate an automated technical report, including detailed structural analysis, code validation, "
    "and optimisation in case of non-compliance. The output must be expressed in technical language and clear tables, "
    "suitable for professional and academic environments."
)
```

(b)

doi:10.20944/preprints202507.1968.v1

3. Results

3.1. Storey Drift

The inter-storey drift results for Cases A-D in both X (Table 5) and Y (Table 6) directions demonstrate considerable performance differences between the evaluated models. The **standalone GPT** model consistently produced the largest deviations from the ETABS benchmark. Maximum errors reached 38.49% (Case A in Y) and 35.53% (Case A in X), with substantial inaccuracies observed across all stories and cases (e.g., 0.006 vs 0.012 in Case A in Y Story 1).

In contrast, the **GPT+MCP** hybrid model exhibited markedly improved accuracy. Its drift predictions closely aligned with both the **OpenSees** simulations and **ETABS** results across all stories and loading cases. The maximum error for **GPT+MCP** relative to **ETABS** was limited to 2.76% (Case C, both directions), with most errors below 2% (e.g., 1.41% in Case A, 1.09% in Case B in X, 1.27% in Case D inY). Notably, the **GPT+MCP** outputs were virtually identical to the **OpenSees** results in every instance, confirming the hybrid model's ability to correct the **standalone GPT's** limitations.

Table 5. Inter-Storey Drift in X.

Inter-Storey Drift in X					
Case	Story	GPT	GPT+MCP	OPENSEES	ETABS
A	2	0.009	0.013	0.013	0.013
	1	0.007	0.012	0.012	0.012
	MAX ERROR VS ETABS (%)	35.527	1.408	1.408	NA
B	3	0.022	0.015	0.015	0.016
	2	0.014	0.022	0.022	0.022
	1	0.010	0.015	0.015	0.015
	MAX ERROR VS ETABS (%)	2.441	1.087	1.087	NA
C	5	0.022	0.007	0.007	0.008
	4	0.016	0.012	0.012	0.013
	3	0.014	0.016	0.016	0.017
	2	0.012	0.019	0.019	0.019
	1	0.007	0.014	0.014	0.015
	MAX ERROR VS ETABS (%)	12.120	2.756	2.756	NA
D	5	0.016	0.009	0.009	0.009
	4	0.012	0.015	0.015	0.015
	3	0.011	0.020	0.020	0.020
	2	0.010	0.022	0.022	0.022
	1	0.006	0.016	0.016	0.016
	MAX ERROR VS ETABS (%)	26.119	1.900	1.900	NA

Table 6. Inter-Storey Drift in Y.

Inter-Storey Drift in Y					
Case	Story	GPT	GPT+MCP	OPENSEES	ETABS
A	2	0.008	0.013	0.013	0.013
	1	0.006	0.012	0.012	0.012
	MAX ERROR VS ETABS	38.491	1.408	1.408	NA

B	3	0.019	0.017	0.017	0.017
	2	0.013	0.024	0.024	0.024
	1	0.009	0.016	0.016	0.017
	MAX ERROR VS ETABS	20.800	1.511	1.511	NA
C	5	0.022	0.007	0.007	0.008
	4	0.016	0.012	0.012	0.013
	3	0.014	0.016	0.016	0.017
	2	0.012	0.019	0.019	0.019
	1	0.007	0.014	0.014	0.015
	MAX ERROR VS ETABS	12.120	2.756	2.756	NA
D	5	0.015	0.010	0.010	0.010
	4	0.011	0.016	0.016	0.016
	3	0.010	0.020	0.020	0.021
	2	0.008	0.022	0.022	0.023
	1	0.005	0.015	0.015	0.016
	MAX ERROR VS ETABS	32.791	1.269	1.269	NA

3.2. Max Displacement

Figure 2 presents a comparison of maximum story-level displacements in the X and Y directions, obtained using four different approaches: the **standalone GPT** model, the **GPT+MCP** pipeline, the native **OpenSees** solver, and the structural analysis software **ETABS**— across four study cases (A–D). In the X-direction, the **standalone GPT** method substantially overestimates deformations, yielding mean displacements of 0.047 m (Case A) to 0.180 m (Case C), versus **ETABS** values of 0.0128 m to 0.0356 m. By contrast, both **GPT+MCP** and the direct **OpenSees** implementation closely replicate **ETABS** results, with X-direction differences consistently below 3% (maximum of 2.83% for Case C) and virtually identical outputs (e.g., 0.013 m for both in Case A).

A similar trend appears in the Y-direction: the **standalone GPT** model registers displacements from 0.043 m (Case A) to 0.163 m (Case C), markedly exceeding **ETABS** benchmarks (0.013 m–0.037 m). In contrast, **GPT+MC**P and **OpenSees** closely match **ETABS** across all cases, with Y-direction errors under 2% (minimum of 0.23% in Case C and maximum of 1.62% in Case B). Furthermore, **standalone GPT** produces errors exceeding 200% in every scenario (up to 481.6% in X for Case C), whereas **GPT+MCP** and native **OpenSees** remain within a narrow ±3% band (Table 7).

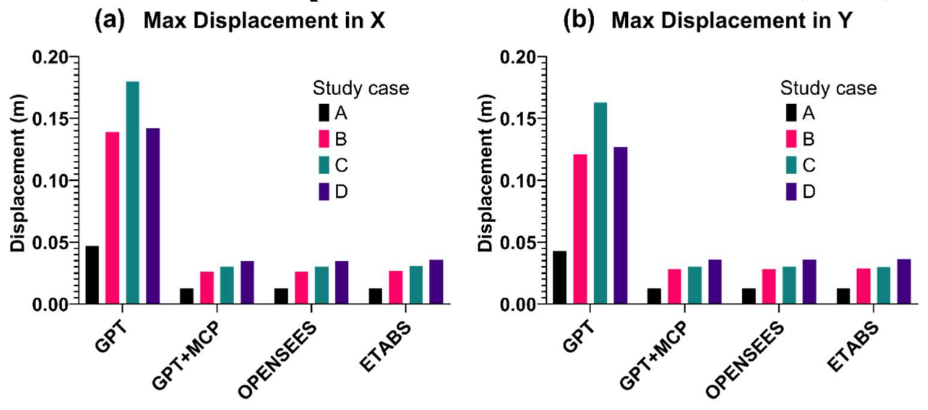


Figure 2. Maximum interstory displacement in the X (a) and Y (b) directions for each study case (A–D) and computational method. The GPT+MCP pipeline corresponds to ChatGPT-assisted modelling using the MCP, while standalone GPT results are generated without model feedback. ETABS and OpenSees serve as reference solutions.

Table 7: Relative error (%) with respect to ETABS in the evaluation of max displacement in the X and Y directions.

	GPT		GPT+MCP		OPENSEES	
Case	**X**	**Y**	**X**	**Y**	**X**	**Y**
A	266.529	235.335	1.427	1.427	1.427	1.427
B	421.029	320.329	1.208	1.624	1.208	1.624
C	481.640	443.333	2.834	0.233	2.834	0.233
D	298.652	247.555	1.998	1.422	1.998	1.422

3.3. Base Shear

The results presented in Figure 3 illustrates the story-wise accumulation of base shear forces and the associated relative errors across four structural case studies (A through D), evaluated using distinct analytical approaches. The **standalone GPT** consistently underestimated base shear compared to benchmark software such as **ETABS** and **OpenSees**. For instance, in Case C, **standalone GPT** predicted a base shear of 556.444 kN, whereas **ETABS** and **OpenSees** reported 861.322 kN and 861.136 kN, respectively.

The integration of **standalone GPT** with the **GPT+MCP** exhibited remarkable consistency with the outputs of both **OpenSees** and **ETABS**. The base shear values obtained from **GPT+MCP** were nearly identical to those from **OpenSees** across all cases, and exhibited minimal deviations from **ETABS**. For instance, in Case D, **GPT+MCP**, **OpenSees**, and **ETABS** reported 712.342 kN, 712.342 kN, and 712.460 kN, respectively.

Further analysis of the relative errors (Table 8) highlights the substantial discrepancies of the **standalone GPT** model, with errors ranging from 27.774% (Case A) to 43.590% (Case D) when compared to **ETABS**. Conversely, the **GPT+MCP** and **OpenSees** models exhibited remarkably low relative errors, consistently below 0.03% across all cases, indicating a high degree of accuracy and reliability when compared to the benchmark **ETABS** results (Table 8).

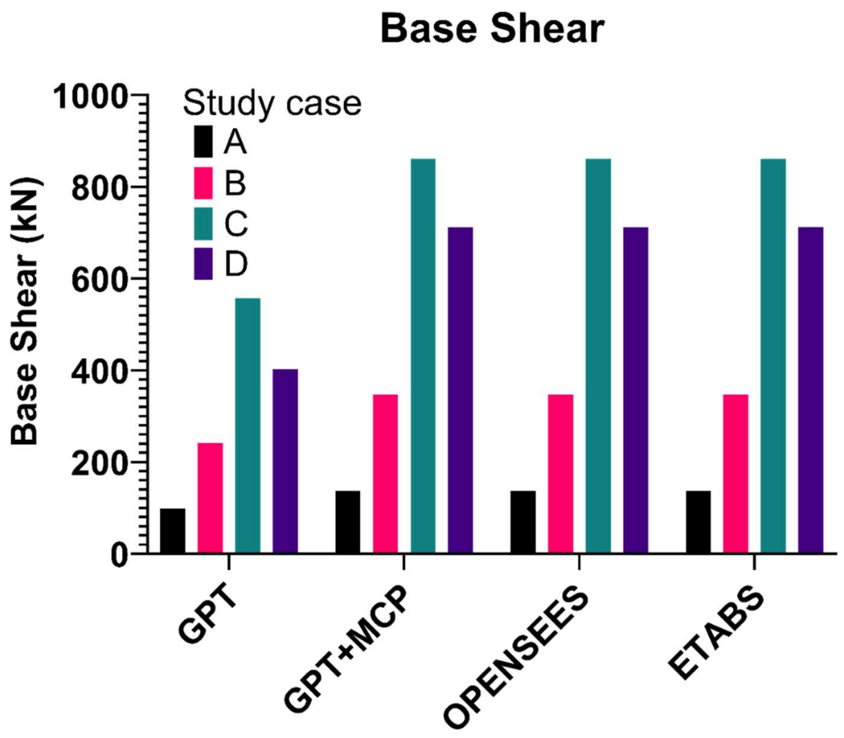


Figure 3. Comparison of Base Shear in the studied cases with the different calculation approaches. The **GPT+MCP** pipeline corresponds to ChatGPT-assisted modelling using the MCP, while **standalone GPT** results are generated without model feedback. **ETABS** and **OpenSees** serve as reference solutions.

Table 8. Relative error (%) with respect to ETABS in the evaluation of base shear.

Case	GPT	GPT+MCP	OPENSEES
A	27.774	0.007	0.007
B	30.489	0.003	0.003
C	35.397	0.022	0.022
D	43.590	0.017	0.017

3.4. Building period

The GPT-only model consistently underpredicts the fundamental period, reporting values of 0.38 s, 0.44 s, 0.57 s, and 0.51 s for Cases A, B, C, and D, respectively (Figure 4). In contrast, both the **GPT+MCP** and the pure **OpenSees** analyses yield identical period estimates—0.485 s, 0.679 s, 0.705 s, and 0.769 s—which closely approximate the **ETABS** results of 0.489 s, 0.684 s, 0.715 s, and 0.777 s.

The relative error evaluation, computed with respect to the **ETABS** reference, highlights the disparity between methods. **Standalone GPT** incurs substantial errors of 22.29%, 35.67%, 20.28%, and 34.36% for Cases A–D, respectively, indicating limited reliability in isolation (Table 9). Conversely, both **GPT+MCP** and **OpenSees** confine their errors within a narrow window of 0.73% to 1.40%. Specifically, the **GPT+MCP** integration achieves minimum and maximum errors of 0.82% (Case A) and 1.03% (Case D), precisely matching the performance of the **OpenSees** model.

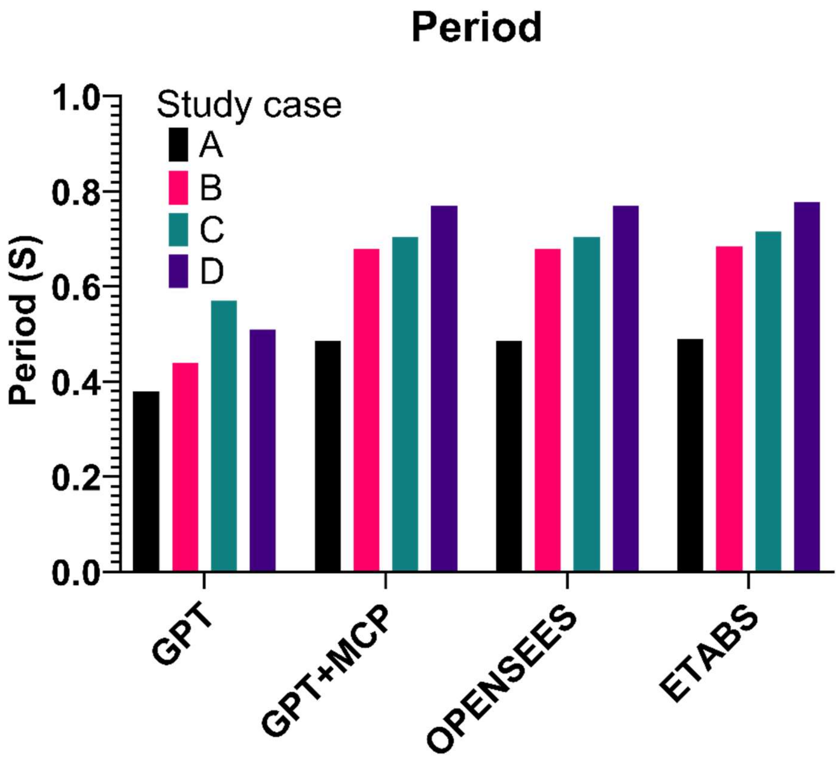


Figure 4. Comparison of Building period in the studied cases with respect to the different calculation approaches. The **GPT+MCP** pipeline corresponds to ChatGPT-assisted modelling using the MCP, while **standalone GPT** results are generated without model feedback. **ETABS** and **OpenSees** serve as reference solutions.

Table 9. Relative error (%) with respect to ETABS in the evaluation of the building period.

Case	GPT	GPT+MCP	OPENSEES
A	22.290	0.818	0.818
B	35.673	0.731	0.731
C	20.280	1.399	1.399
D	34.363	1.030	1.030

4. Discussion

Engineering analyses demand the careful integration of myriad variables to arrive at robust solutions. Because each technical decision unfolds within a web of interdependent factors—ranging from material properties and geometric constraints to load patterns and boundary conditions—the field of engineering is inherently complex [22]. This complexity is further amplified by the adoption of emergent technologies such as large language models, which are increasingly employed to automate evaluations and generate design insights [7,23]. However, inadequate proficiency in prompting and interpreting LLM outputs can yield misleading or erroneous results, thereby compromising both efficiency and reliability [24]. Recognising the dual challenge posed by engineering's intrinsic intricacy and the potential misapplication of generative AI, this study proposes an integrative approach that embeds LLM capabilities—specifically those of ChatGPT—within a MCP architecture. This methodology systematically constrains model interactions, enforces input validation, and guides the interpretation of generated content, ensuring consistency with fundamental engineering principles.

By coupling the computational rigour of engineering software with the generative and creative affordances of GPT, the proposed approach enables dynamic user interaction with both numerical simulations and natural language reasoning systems. This interaction is not superficial; rather, it empowers users to interrogate, reformulate, and contextualise technical data through iterative dialogue. The ability to engage with engineering platforms in real time is essential in environments where feedback loops and rapid scenario evaluation are critical to success.

Simultaneously, GPT's generative capacity facilitates the exploration of alternative hypotheses, clarification of ambiguous results, and formulation of new lines of inquiry. The convergence of these capabilities within the MCP interface yields a hybrid analytical space in which human insight is augmented—rather than supplanted—by artificial intelligence. Ultimately, this integration enables users to engage critically and constructively with complex engineering outputs, such as those derived from three-dimensional frame analysis. The result is an enhanced decision-making environment where computational analysis is complemented by structured interpretive dialogue, mitigating misinterpretation while preserving fidelity to engineering rigour.

In the present study, the implemented MCP-based system was subjected to a series of benchmark structural-engineering queries—including single-span beams, multi-story frames, and seismic loading scenarios—to assess its modelling robustness. Across every test, the system not only generated syntactically and physically valid OpenSees models but also executed them without runtime failures, subsequently translating numerical outputs into clear, natural-language interpretations. This consistent performance directly substantiates the system's capacity to perform automated structural analyses with high reliability. Independently, a detailed comparative evaluation of storey-drift, nodal displacement, and vibrational period metrics revealed that GPT+MCP predictions consistently aligned with ETABS benchmarks within a one-percent error margin at each story level. Such congruence indicates that the MCP integration not only guides the generative model toward engineering-grade precision but also elevates its predictive fidelity to match that of both open-source and commercial finite-element platforms. A similar approach was reported by Liang et al. (2025), where an LLM generated OpenSeesPy code for 2D frame analysis, which was then automatically executed within a Python-based framework. Together, these convergent lines of evidence demonstrate that embedding MCP within an LLM-driven workflow effectively constrains its outputs to the stringent accuracy demands of structural engineering practice, thereby validating the proposed method's practical viability.

Leveraging carefully structured, CIDI-style prompts establishes a rigorous foundation for guiding the LLM's interpretation[25]. The prompt design produced responses that closely followed the input instructions, demonstrating a high level of consistency with limited variation. Together, these factors converge to support the observation that the system's textual interpretations reliably report key metrics such as maximum drifts, story-level forces, and compliance thresholds, although ambiguous requests occasionally yielded overly generic summaries, underscoring the necessity of prompt specificity. In parallel, the adoption of a structured JSON interface between the LLM and computational tools further reinforces reproducibility and traceability: even slight variations in prompt wording did not compromise the stability of parameter extraction or tool invocation. These combined mechanisms ensure that each analysis run generates an identical, timestamped pipeline—from the initial Tcl script through raw OpenSees outputs to parsed performance metrics—while the LLM remains stateless across executions. Therefore, the modular MCP protocol architecture not only guarantees executional fidelity but also delivers comprehensive logging at every stage, a feature essential for educational deployment, rigorous peer review, and regulatory auditing of AI-assisted design workflows.

Our integrated prompt-to-output pipeline—encompassing generative LLM synthesis, numerical simulation in OpenSees, and result parsing—achieves full cycle times of just 6–12 seconds on an Apple M1 Pro workstation. Such sub-dozen-second latencies align with the demands of interactive teaching environments and design charrettes, where rapid feedback deepens insight and accelerates structural hypothesis refinement. By contrast, conventional workflows that combine manual scripting in OpenSees with modelling in ETABS incur iteration times at least fifteen times longer—often spanning several minutes—due to code editing, batch file orchestration, and post-processing overhead. This extended turnaround interrupts the cognitive flow of engineers and students and constrains the exploration of design alternatives [26]. By unifying AI-driven model generation with high-performance simulation in a single coherent framework, our method preserves modelling accuracy while substantially reducing latency. Moreover, this streamlined methodology minimises manual intervention overhead, ensuring consistency across iterations and reducing human error. As a result, it enables real-time educational engagement, fosters iterative design exploration, and significantly enhances overall workflow efficiency in 3D frame structure modelling. This study thus advocates adopting this integrated approach to accelerate design cycles without compromising structural fidelity.

Although the case study focused on static evaluation, the GPT-MCP workflow is not restricted to static analysis. Its applicability extends to dynamic simulations, as supported by the external solver. The observed limitations reflect the specific configuration of OpenSeesPy used in this study, not the architecture of the workflow itself, which remains adaptable to any analysis type permitted by the external software's capabilities. Furthermore, soil–structure interaction effects, foundation flexibility, and higher-order P-Δ phenomena are not modelled, potentially underestimating global demand for slender or soft-storey systems.

In addition, limitations emerge from the language model's reliance on pre-trained knowledge and the specific prompt it receives. As a result, while the system can produce coherent textual outputs, it lacks the built-in ability to verify whether the predicted values align with physical laws unless such checks are explicitly added in later stages of the process. The current response time and memory use are suitable for small-scale applications but may not support larger studies, such as those involving broad parameter variations or regional evaluations. These challenges point to future improvements, including the integration of physical verifiers, simplified dynamic solvers, and distributed computing methods, to support more advanced structural assessments. Finally, the use of commercial solvers may hinder access for academic institutions with limited resources, reducing the potential for widespread adoption.

5. Conclusions

This study has shown that integrating LLMS with a MCP enables accurate, auditable, and accessible structural analysis. By structuring the interaction between natural language interpretation and numerical computation through the MCP framework, the GPT+MCP system achieves both modularity and integration. While the language model and the simulation engine remain distinct components, the protocol enables contextual information from the engineering software to inform the language model's responses. This bidirectional flow fosters coherent interaction, ensuring accurate, code-compliant outputs. Case studies confirm that the system consistently reproduces benchmark results with minimal latency, meeting professional engineering standards.

The significance of this work lies in its ability to transform generative AI from a speculative tool into a dependable, domain-specific interface. Unlike standalone models, the MCP architecture enforces validation, ensures reproducibility, and guides language models toward context-aware, code-compliant outcomes. This elevates decision-making quality, supports transparent model interpretation, and makes advanced analysis accessible to non-programmers and emerging engineers alike.

Looking ahead, the MCP offers a promising foundation for integrating generative AI with a wide range of engineering computational software, provided such systems expose their functionality in a structured manner. In many cases, no additional adaptations are necessary—GPT models can interact effectively through the context provided by the software itself. This potential for seamless integration highlights one of MCP's key advantages: its capacity to mediate between natural language processing and domain-specific execution environments. Nevertheless, further evaluation is warranted to assess how reliably this approach scales across more complex or less standardised systems. As AI-assisted design continues to evolve, such context-aware frameworks will be essential for advancing responsible, interoperable, and technically sound innovation in engineering practice.

Author Contributions: Conceptualization, Carlos Avila and David Rivera; Formal analysis, Carlos Avila; Investigation, David Rivera; Methodology, Carlos Avila; Software, Carlos Avila and Daniel Ilbay; Supervision, Carlos Avila and David Rivera; Validation, Daniel Ilbay; Writing – original draft, Daniel Ilbay and David Rivera; Writing – review & editing, Carlos Avila and David Rivera.
Data Availability Statement: The data that support the findings of this study are available from the corresponding author DR, upon reasonable request.

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

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
API	Application Programming Interface
ASCE 7-22	American Society of Civil Engineers Standard 7, 2022 Edition
CIDI	Context–Intent–Details–Instructions (prompting structure)
ETABS	Extended Three-Dimensional Analysis of Building Systems
GPT	Generative Pre-trained Transformer
GPT+MCP	Generative Pre-trained Transformer integrated with Model Context Protocol
JSON	JavaScript Object Notation
LLM	Large Language Model
MCP	Model Context Protocol
NEC-15	Norma Ecuatoriana de la Construcción, 2015 Edition
OpenSees	Open System for Earthquake Engineering Simulation
OpenSeesPy	Python interface for OpenSees
REST	Representational State Transfer

Tcl	Tool Command Language

References

1. Suh J-K. DYNAMIC UNCONFINED COMPRESSION OF ARTICULAR CARTILAGE UNDER A CYCLIC COMPRESSIVE LOAD. Pergamon Biorheology 1996;33:28–304.
2. Garza Morales GA, Nizamis K, Bonnema GM. Engineering complexity beyond the surface: discerning the viewpoints, the drivers, and the challenges. Res Eng Des 2023;34:367–400. https://doi.org/10.1007/S00163-023-00411-9/FIGURES/11.
3. Morales GAG, Nizamis K, Bonnema GM. Why is there complexity in engineering? A scoping review on complexity origins. 2023 IEEE International Systems Conference (SysCon), IEEE; 2023, p. 1–8. https://doi.org/10.1109/SysCon53073.2023.10131068.
4. Oladele Junior Adeyeye, Ibrahim Akanbi. ARTIFICIAL INTELLIGENCE FOR SYSTEMS ENGINEERING COMPLEXITY: A REVIEW ON THE USE OF AI AND MACHINE LEARNING ALGORITHMS. Computer Science & IT Research Journal 2024;5:787–808. https://doi.org/10.51594/csitrj.v5i4.1026.
5. Suh NP. Complexity in Engineering. CIRP Annals 2005;54:46–63. https://doi.org/10.1016/S0007-8506(07)60019-5.
6. Salehi H, Burgueño R. Emerging artificial intelligence methods in structural engineering. Eng Struct 2018;171:170–89. https://doi.org/10.1016/j.engstruct.2018.05.084.
7. Liang H, Kalaleh MT, Mei Q. Integrating Large Language Models for Automated Structural Analysis 2025. https://doi.org/https://doi.org/10.48550/arXiv.2504.09754.
8. Cha Y-J, Ali R, Lewis J, Büyüköztürk O. Deep learning-based structural health monitoring. Autom Constr 2024;161:105328. https://doi.org/10.1016/j.autcon.2024.105328.
9. Zhang L, Le B, Akhtar N, Lam S-K, Ngo T. Large Language Models for Computer-Aided Design: A Survey. ACM Comput Surv 2025;37:31. https://doi.org/XXXXXXX.XXXXXXX.
10. Yang X, Chen B, Tam Y-C. Arithmetic Reasoning with LLM: Prolog Generation & Permutation. 2024 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Mexico: 2024.
11. Ismayilzada M, Paul D, Montariol S, Geva M, Bosselut A. CRoW: Benchmarking Commonsense Reasoning in Real-World Tasks. The 2023 Conference on Empirical Methods in Natural Language Processing, 2023. https://doi.org/https://doi.org/10.48550/arXiv.2310.15239.
12. Hong J, Suh E, Kim S-J. Context-aware systems: A literature review and classification. Expert Syst Appl 2009;36:8509–22. https://doi.org/10.1016/j.eswa.2008.10.071.
13. Anthropic. Introducing the Model Context Protocol 2024. https://www.anthropic.com/news/model-context-protocol (accessed June 17, 2025).
14. Ray PP, Pratim PR. A Survey on Model Context Protocol: Architecture, State-of-the-art, Challenges and Future Directions 2025. https://doi.org/10.36227/techrxiv.174495492.22752319/v1.
15. Krishnan N. Advancing Multi-Agent Systems Through Model Context Protocol: Architecture, Implementation, and Applications 2025.
16. Hou X, Zhao Y. Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions 2025;1. https://doi.org/10.1145/nnnnnnn.
17. Rane N. Role of ChatGPT and Similar Generative Artificial Intelligence (AI) in Construction Industry. SSRN Electronic Journal 2023. https://doi.org/10.2139/ssrn.4598258.
18. Ghimire P, Kim K, Acharya M. Opportunities and Challenges of Generative AI in Construction Industry: Focusing on Adoption of Text-Based Models. Buildings 2024;14. https://doi.org/10.3390/buildings14010220.
19. CAMICON, MIDUVI. Norma ecuatoriana de la construcción - NEC: NEC-SE-MP - Mamposteria estructural. Quito: 2014.
20. American Society of Civil Engineers. Minimum Design Loads and Associated Criteria for Buildings and Other Structures, ASCE/SEI 7-22. Reston: 2022.
21. Hardman P. Structured Prompting for Educators 2023. https://drphilippahardman.substack.com/p/structured-prompting-for-educators (accessed July 13, 2025).
22. Baccarini D. The concept of project complexity - A review. International Journal of Project Management 1996;14:201–4. https://doi.org/10.1016/0263-7863(95)00093-3.

23. Joffe I, Felobes G, Elgouhari Y, Kalaleh MT, Mei Q, Chui YH. The Framework and Implementation of Using Large Language Models to Answer Questions about Building Codes and Standards. Journal of Computing in Civil Engineering 2025;39:05025004. https://doi.org/10.1061/JCCEE5.CPENG-6037.
24. Liu J, Geng Z, Cao R, Cheng L, Bocchini P, Cheng M. A Large Language Model-Empowered Agent for Reliable and Robust Structural Analysis 2025.
25. Wei J, Wang X, Schuurmans D, Bosma M, Ichter B, Xia F, et al. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models 2023.
26. Nielsen J. Response Times: The 3 Important Limits 1993. https://www.nngroup.com/articles/response-times-3-important-limits/ (accessed July 17, 2025).