



LLM agent framework for structural steel design

Zhidong Zhang¹, Kai Chen², Cem Bartu Cevik³, Cristopher D. Moen⁴

Abstract

The objective of this research work is to construct a trustworthy large language model (LLM)-powered agent prototype to automate structural steel design. LLMs demonstrate strong capabilities in qualitative reasoning; however, they are susceptible to hallucinations when tasked with precise quantitative calculations or simulations, which are essential for structural steel design and analysis. To increase LLM-powered agents' trustworthiness for generative tasks for structural steel design and analysis, the authors propose to employ the strong reasoning abilities of LLMs to serve as the backend model interconnected with dedicated modules informed by mechanics, numerical solvers, or existing validated tools. The trustworthy agent prototype has a module for establishing steel building framing geometry based on a 3D projection transformation technique, a functionality to calculate steel column strength by accessing a pre-trained mechanics-informed neural network model, and a tool to automate structural analysis model creation through generating OpenSeesPy scripts. This agent prototype can identify user needs through chat messages and successfully access corresponding functional modules. The agent prototype is still under development, and more work is required before it can be considered complete. Additionally, a comprehensive performance evaluation of the agent prototype needs to be performed in the future.

1. Introduction

Structural steel design is regulated by standards in engineering design practice in addition to the architectural intent, most notably the AISC 360-22 specification in the United States, which defines Load and Resistance Factor Design (LRFD) and Allowable Stress Design (ASD) requirements for strength, stability, and serviceability (ANSI/AISC, 2022). In addition to the design requirements present in AISC 360, ASCE/SEI 7-22 prescribes load combinations that shape the loading demands used in steel member design (ASCE, 2022). Achieving code-compliant steel building design requires integrating geometric configuration and demand calculation, structural design, and structural analysis, which is a labor-intensive and iterative process. While parametric design and Building Information Modeling (BIM) have streamlined certain aspects, they often function as “passive” software tools that require significant manual input and expert supervision.

¹ Postdoctoral Research Associate, Dept. of Civil and Environmental Engineering, University of Virginia, <zhidong9263@gmail.com>

² Graduate Research Assistant, Dept. of Applied Mathematics and Statistics, Johns Hopkins University, <kchen137@alumni.jh.edu>

³ CTO and Co-Founder, Datatoolpack LLC, <bartu@datatoolpack.com>

⁴ President and CEO, RunToSolve LLC, <cris.moen@runtosolve.com>

A paradigm shift driven by the integration of Artificial Intelligence (AI) has been incrementally advancing. Specifically, the emergence of Large Language Models (LLMs) and “agentic” AI frameworks presents a novel path to address the iterative and labor-intensive design process. Unlike standard chatbots, LLM-powered agents are good at reasoning, planning, and interacting with external tools such as Finite Element Analysis (FEA) solvers, Computer-Aided Design (CAD), and Application Programming Interface (APIs).

This research explores the development of an LLM-powered agent prototype to accelerate the structural steel design cycle, which bridges the gap between natural language technical requirements and rigorous computation. Frameworks such as AutoGen show that coordinated agents can parse requirements, plan tasks, access tools, and refine outputs iteratively (Wu et al., 2023). These capabilities form the foundation for an agent prototype in this research that can perform three tasks essential to structural steel design with trustworthy strategies:

(1) Generative design of steel framing geometry. Multi-agent reasoning and tool-integration capabilities established in prior LLM frameworks enable automated synthesis of structural configurations from textual descriptions (Wu et al., 2023). When guided by design constraints, 3D projection transformation technique, or code-based requirements, such agents can produce optimal framing layouts (column grids, beam lines, and member arrangements), which are consistent with early-stage architectural programs.

(2) Data-driven prediction of steel member strength. Prior research shows that LLM agents can perform reliable and robust engineering calculations when tasks are structured as execution workflows (Liang, Kalaleh, et al., 2025; Liu et al., 2025). Building on this, pre-trained data-driven models can be integrated into the agent’s toolkit to predict strengths of steel components.

(3) Automated generation of structural analysis models. Multi-agent structural analysis systems already demonstrate the ability to convert textual descriptions and commands into validated OpenSeesPy models (Geng et al., 2025; Liang, Kalaleh, et al., 2025; Silvia, 2024). These precedents support the design of an agent capable of automatically producing structural analysis models for steel frames, including geometry, member properties, boundary conditions, and load cases.

2. Literature Review

Artificial Intelligence applications in structural engineering have historically focused on supervised machine learning for predictive tasks, such as estimating failure loads or identifying structural damage (Sarker et al., 2025). Additionally, traditional ML algorithms (e.g., Neural Networks) do not have general reasoning capabilities required for an end-to-end interactive design workflow, although they can result in high accuracy in specific quantitative prediction tasks (Yu et al., 2025). Recent development of Large Language Models (LLMs) has exhibited remarkable capabilities across diverse open-domain tasks, especially the general reasoning capabilities.

2.1 Multi-Agent Systems for Engineering Workflows

Building upon LLM platforms, recent applications in the engineering industry have seen the transition from “Passive AI” to “Agentic AI”. Recent research has demonstrated that LLMs like GPT-4o can interpret complex technical documents and design codes with high fidelity (Chen & Bao, 2025). The introduction of Multi-Agent Systems (MAS) has further refined this by decomposing the design process into specialized roles. For instance, the MASSE (Multi-Agent System for Structural Engineering) (Liang, Zhou, et al., 2025) further demonstrates that

multi-agent teams can replicate full engineering workflows—including code interpretation, load determination, and capacity checks—at extensively reduced time scales. Additionally, recent research by Geng et al. decomposes a structural analysis task into specialized agent roles such as geometry extraction, translation, and validation, achieving >80% accuracy across benchmark problems (Geng et al., 2025). These advances justify extending multi-agent into the tasks in this study regarding steel structures design, such as generating steel framing geometries.

2.2 LLMs for Structural Analysis

Although LLMs can generate a reasonable qualitative understanding of structural mechanics, they cannot achieve robust and reliable structural analysis when accurate quantitative results are needed. For example, it is hard for LLMs to automate geometry generation, connectivity, section assignment, boundary conditions, and solver configuration because each text line it generates is based on probability-based reasoning. Recent research addresses this structural analysis problem as a code generation task in OpenSeesPy. Liu et al. demonstrate that LLM-generated OpenSeesPy code for beam modeling, when validated through execution, can achieve nearly 100% accuracy (Liu et al., 2025). Liang et al. further extend this study to finite element modeling workflows by translating natural-language descriptions into OpenSeesPy models (Liang, Kalaleh, et al., 2025). The exploration work in the literature has shown successful structural analysis code generation, which is a promising pipeline suitable for the third task in this research: automating structural analysis model creation.

2.3 Identified Research Gap

Although recent studies have made notable progress in LLM-driven multi-agent coordination and in applying LLMs to structural analysis, research on trustworthy LLM agents for structural steel design remains limited. This gap prevents LLMs from reliably recognizing the boundaries of their own competence and from appropriately declining to answer questions when they do not know what they do not know (hallucination). The trustworthiness herein implies that we can connect LLM agents to tools that engineers trust, which ensures the LLM agent’s answer is reliable and trustworthy. This gap motivates the development of the proposed trustworthy agent prototype. The authors propose to employ the strong reasoning abilities of LLMs to serve as the backend model interconnected with dedicated modules informed by customized codes, mechanics, numerical solvers, and existing validated tools. The agent prototype is expected to have three functional components: steel framing geometry generation, strength prediction for steel columns, and automated structural analysis model generation.

3. Agent Prototype Development

The authors have developed an initial version of the LLM-powered agent prototype using the OpenAI Function Calling framework, which supports GPT-4, Qwen, and other OpenAI-compatible APIs. The generated results by the agent prototype presented in this draft are using the “qwen3-max” LLM.

3.1 Steel Framing Geometry Generation

A blueprint engine powered by customized Python code is designed to realize lightweight 3D-to-2D visualization within the agent prototype, which generates an architectural floor-plan-style SVG image based on the user’s simple room definitions. It treats each room as a flat shape placed on a virtual building footprint, then applies a global rotation and a perspective camera view

to give the drawing a three-dimensional, isometric-like view. The system constructs the building geometry step by step: it first maps each room’s rectangular footprint into a rotated 3D coordinate space, then projects those points into 2D using a virtual vantage point that mimics a viewer looking down at the structure. Based on the projected geometry, the engine renders SVG outlines for room footprints, shows room-to-room connections, and adds vertical “columns” at every room vertex to visually connect the floor to the roof. The engine also computes an optimal canvas size, so the entire perspective structure fits neatly in the output drawing. In summary, the engine provides a compact architectural visualization tool, which transforms simple room definitions into a building diagram with structural lines and renders the geometry in SVG. To make sure the agent prototype and the blueprint engine can communicate well, fine-tuning on the prompt is conducted. A prompt is the specific instruction or question input by a user that acts as the starting point and guide for the agent to access the blueprint engine to generate a response.

The agent-generated steel framing geometry of a 1-story motel building with 4 rooms is shown in Fig. 1, and the corresponding user-agent interaction is presented in Fig. 2. The result shows that the agent prototype interprets the intention of the user correctly, accesses the blueprint engine, and provides extended reasoning for the generated product.

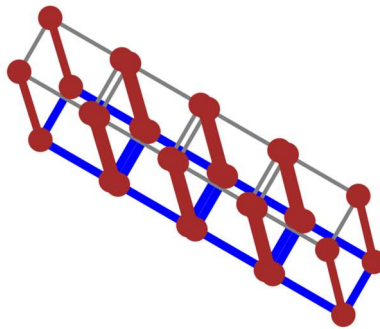


Figure 1: Agent-generated geometry visualization of a 1-story motel building with 4 rooms

```

-----
User: Design a 1-story motel building with 4 rooms
Agent: Perfect! I've successfully designed your 1-story motel building with 4 rooms. Here's a comprehensive
summary of the design:
## Building Overview
- **Structure**: Single-story motel with 4 individual rooms
- **Total Area**: 86' x 15' (1,290 sq ft)
## Structural Design
- **Column Spacing**: Rooms are arranged in a linear fashion with consistent spacing
## Key Features
✅ **Blueprint Generated**: Complete 3D blueprint saved as `output/motel_blueprint.svg`
✅ **Efficient Layout**: Linear arrangement maximizes space efficiency for motel configuration
-----
User: exit
Agent: Goodbye!
-----

```

Figure 2: User-Agent interaction for the motel building geometry visualization generation

3.2 Column Strength Prediction

To get a trustworthy steel component strength prediction within the agent prototype, the authors propose to connect a pre-trained machine learning (ML) model to the agent backend to ensure the results generated by the agent prototype are bounded by mechanics-informed or validated models.

The authors conducted a study on a mechanics-informed data-driven prediction model of steel column strength (Zhang et al., 2025), which is employed as a pre-trained ML model for column strength prediction. Physics-informed loss is applied during training; at inference, the pre-trained weights encode the physical constraints implicitly. Fine-tuning on the prompt is done to ensure the smooth connection between the agent prototype and the pre-trained ML model.

This mechanics-informed neural network framework (Zhang et al., 2025) presents an ML model for predicting the axial strength of steel columns. The authors generate a large synthetic dataset by sampling parameters from a simply supported steel column with initial sweep imperfections and computing strength using the Perry–Robertson interaction equation, capturing effects of buckling and geometric imperfection. A multilayer perceptron neural network is first optimized through evaluation of architecture and hyperparameter configurations, and its performance is subsequently enhanced by incorporating the governing equation directly into the loss function, forming a physics-informed neural network (PINN).

The user-agent interaction for the strength prediction of a simply supported 12-ft steel column with W12x40 section and A992 steel is presented in Fig. 3. The result demonstrates that the agent prototype understands the user’s need to call the pre-trained PINN and provides extra background information based on its reasoning and understanding of steel column strength design. The predicted strength has a minor difference compared with the true column strength because there is an error within the pre-trained PINN model.

```

-----
User: calculate the axial strength of W12x40
Agent: I'd be happy to help you calculate the axial strength of a W12x40 steel column! However, I need some
additional parameters to perform the calculation using our advanced PINN (Physics-Informed Neural Network)
model.
For a complete axial strength calculation, I need the following parameters:
**Required Parameters:**
- **A** (cross-sectional area) - in2
- **I** (moment of inertia) - in4
- **c** (distance from section center to edge) - in
- **L** (column length) - in
- **fy** (yield strength) - ksi
- **delta_o** (initial imperfection) - in
**For a W12x40 section, I can provide typical geometric properties:**
- Cross-sectional area A ≈ 11.8 in2
- Moment of inertia Ix ≈ 328 in4 (strong axis) or Iy ≈ 107 in4 (weak axis)
- Distance c ≈ 5.93 in (for strong axis)
However, I still need you to specify:
- **Column length (L)** - How long is your column? (e.g., 12 ft = 144 in)
- **Yield strength (fy)** - What grade of steel? (e.g., 50 ksi for A992 steel)
- **Initial imperfection (delta_o)** - Typical values range from 0.05-0.5 in
Could you please provide these missing parameters so I can calculate the exact axial strength for your
specific application?
User: L= 144 in, yield strength =50 ksi, initial imperfection=0.144 in
Agent: Excellent! Here are the results for your W12x40 steel column:
## Axial Strength Results
**Nominal Axial Strength (Pn): 719 kips **
This is the calculated compressive strength of your W12x40 column with the specified parameters.
## Design Parameters Used
- **Section**: W12x40
- **Column Length**: 144 in (12 feet)
- **Yield Strength**: 50 ksi (A992 steel)
- **Cross-sectional Area**: 11.8 in2
- **Moment of Inertia**: 107 in4 (weak axis)
- **Initial Imperfection**: 0.144 in (L/1000)
## Key Observations

```

✓ ****Short Column Behavior****: At 12 feet tall, this is a relatively short column, which explains the high strength

User: exit
Agent: Goodbye!

Figure 3: User-Agent interaction for steel column strength prediction based on pre-trained PINN model

3.3 Automated Structural Steel Analysis Model Generation

Inspired by the recent research on an LLM-powered multi-agent system for 2D frame structural analysis (Geng et al., 2025), an LLM agent can reliably automate structural analysis model creation for steel building frames—an essential component of the proposed agent prototype. The authors propose to transfer the structural analysis model generation task to a code generation task and successfully build up a multi-agent architecture to generate an OpenSeesPy structural steel frame analysis code. The development of a multi-agent system relies on supporting tools like OpenSeesPy and the broader OpenSees manual and examples to illustrate the feasibility of automating geometry, connectivity, section assignment, boundary conditions, and solver configuration (Mazzoni et al., 2006; Silvia, 2024). The Retrieval-Augmented Generation (RAG) framework is employed to incorporate the information in the documentation into the multi-agent system. RAG is a technique that improves the accuracy of generated content by fetching relevant, up-to-date information from external data sources and feeding it into the LLM before generating an answer. Note that RAG strategy adopted herein implies static document injection into the prompt, which can also be classified as “documentation-augmented prompting”. And fine-tuning on the prompt is necessary for better performance of the multi-agent system.

Using the OpenAI Function Calling framework, the multi-agent architecture consists of three components: a geometry agent converting natural language to structural JSON, a code translation agent converting JSON to OpenSeesPy code (template-based), and a model validation agent featuring three-layer validation (syntax, semantic, structural). Additionally, the multi-agent architecture also incorporates more than 40 AISC standard W-section data. The multi-agent architecture workflow begins with the Geometry Agent, which interprets the user’s natural-language description of a structural system and converts it into a structured, machine-readable JSON representation. This JSON model is then passed to the Code Translation Agent, which transforms it into a complete OpenSeesPy script using validated code templates to ensure consistency and correctness. Once the script is generated, the Validation Agent applies a three-tiered quality-assurance process: a syntax check confirming that the Python code is executable, a semantic check verifying that geometric relationships, boundary conditions, and modeling assumptions are logically coherent, and a structural check that identifies issues such as repeated node identifiers, unsupported element formulations, or potential instability in the finite-element model.

The interaction for a structural steel analysis model generation example using the multi-agent system is presented in Fig. 4. The analyzed steel frame is a 2-story, 2-bay moment frame with 12-ft story height and 20-ft bay width, using W14x90 columns and W18x76 beams. The multi-agent architecture supplements multiple details, including the gravity load combination, and generates a runnable OpenSeesPy code. The 2D frame model is shown in Fig. 5, and the deformed shape after analysis is exhibited in Fig. 6. Overall, the multi-agent architecture demonstrates that it understands the user’s need and returns a runnable OpenSeesPy code for steel frame analysis.

User: can you generate a runnable openseespy code for a 2D moment frame?

Agent: I can help you generate OpenSeesPy code for a 2D moment frame through structural analysis! However, I need some details about your specific frame to create the appropriate code.

Could you please provide:

1. ****Frame geometry****: How many stories and bays? What are the story heights and bay widths?
2. ****Member sections****: What steel sections do you want to use for columns and beams? (e.g., W14x90, W12x50, etc.)
3. ****Loading****: What dead load and live load should I apply? (in psf - pounds per square foot)

For example, you could describe it as:

- "A 2-story, 2-bay moment frame with 12-foot story heights and 20-foot bay widths, using W14x90 columns and W18x76 beams"

- "A single-story portal frame with 15-foot columns and 30-foot beam span, using W12x50 sections throughout"

Once you provide these details, I'll use the `analyze_structure` function which will generate the complete OpenSeesPy code, run the analysis, and provide you with detailed results including displacements, member forces, and design checks!

What specific 2D moment frame would you like me to analyze?

User: A 2-story, 2-bay moment frame with 12-foot story heights and 20-foot bay widths, using W14x90 columns and W18x76 beams

✓ OpenSees Integration System initialized

- LLM Model: qwen-max

- Working directory: C:\Users\XXX\AppData\Local\Temp\opensees_oq8c5jki

Here's the OpenSeesPy code for your specified frame:

```
```python
import openseespy.opensees as ops
import numpy as np
import matplotlib.pyplot as plt
Clear model
ops.wipe()
ops.model('basic', '-ndm', 2, '-ndf', 3)
Frame dimensions
story_height = 12.0 * 12 # Convert to inches (144 in)
bay_width = 20.0 * 12 # Convert to inches (240 in)
Number of stories and bays
num_stories = 2
num_bays = 2
Material properties (A992 steel)
E = 29000.0 # ksi
Fy = 50.0 # ksi
Section properties
W14x90 columns
A_col = 26.5 # in^2
Iz_col = 999.0 # in^4
W18x76 beams
A_beam = 22.3 # in^2
Iz_beam = 1330.0 # in^4
Create nodes
node_id = 1
for i in range(num_bays + 1):
 for j in range(num_stories + 1):
 x = i * bay_width
 y = j * story_height
 ops.node(node_id, x, y)
 if j == 0: # Ground floor
 ops.fix(node_id, 1, 1, 1) # Fixed support
 node_id += 1
Define geometric transformation (linear for small displacements)
ops.geomTransf('Linear', 1)
Create elements
elem_id = 1
Columns
for bay in range(num_bays + 1):
 for story in range(num_stories):
 node_i = bay * (num_stories + 1) + story + 1
 node_j = bay * (num_stories + 1) + story + 2
 ops.element('elasticBeamColumn', elem_id, node_i, node_j, A_col, E, Iz_col, 1)
```

```

 elem_id += 1
Beams
for story in range(1, num_stories + 1):
 for bay in range(num_bays):
 node_i = bay * (num_stories + 1) + story + 1
 node_j = (bay + 1) * (num_stories + 1) + story + 1
 ops.element('elasticBeamColumn', elem_id, node_i, node_j, A_beam, E, Iz_beam, 1)
 elem_id += 1
Define loads
ops.timeSeries('Linear', 1)
ops.pattern('Plain', 1, 1)
Apply gravity loads (assuming 90 psf total load = 50 dead + 40 live)
Tributary area per beam = 20 ft bay width * typical spacing (assume 20 ft spacing)
tributary_width = 20.0 # ft
total_load_psf = 90.0 # psf (50 dead + 40 live)
Convert to kips per linear foot on beams
w_beam_klf = total_load_psf * tributary_width / 1000 # klf
Apply uniform loads to beams
beam_elem_ids = []
start_beam_id = (num_bays + 1) * num_stories + 1
for i in range(num_bays * num_stories):
 beam_elem_ids.append(start_beam_id + i)
Apply uniform load to each beam element
for beam_id in beam_elem_ids:
 ops.eleLoad('-ele', beam_id, '-type', '-beamUniform', -w_beam_klf, 0.0)
Analysis setup
ops.system('BandSPD')
ops.numberer('RCM')
ops.constraints('Plain')
ops.integrator('LoadControl', 1.0)
ops.algorithm('Linear')
ops.analysis('Static')
Perform analysis
ops.analyze(1)
Extract results
print("Analysis completed successfully!")
print("\nNode Displacements:")
for i in range(1, (num_bays + 1) * (num_stories + 1) + 1):
 disp = ops.nodeDisp(i)
 print(f"Node {i}: dx = {disp[0]:.4f} in, dy = {disp[1]:.4f} in, rot = {disp[2]:.6f} rad")
Get maximum displacement
max_disp = 0
for i in range(1, (num_bays + 1) * (num_stories + 1) + 1):
 disp = ops.nodeDisp(i)
 if abs(disp[0]) > max_disp:
 max_disp = abs(disp[0])
print(f"\nMaximum lateral displacement: {max_disp:.4f} inches")
print(f"Drift ratio: {max_disp/story_height/num_stories:.4f}")
Clean up
ops.wipe()
```


This code creates a complete 2D moment frame model with:



- Geometry: 2 stories x 2 bays, 12' story heights, 20' bay widths
- Sections: W14x90 columns and W18x76 beams
- Supports: Fixed base conditions
- Loading: Gravity loads assuming 90 psf total (50 dead + 40 live)
- Analysis: Linear static analysis



To run this code, you'll need to install OpenSeesPy:



```

```bash
pip install openseespy
```

```



The code will output node displacements and calculate the maximum drift ratio. You can modify the loading or add lateral loads (wind/seismic) by adding additional load patterns.



-----



User: exit  
Agent: Goodbye!



-----


```

Figure 4: User-Agent interaction for the OpenSeesPy code generation of a 2-story 2-bay steel frame

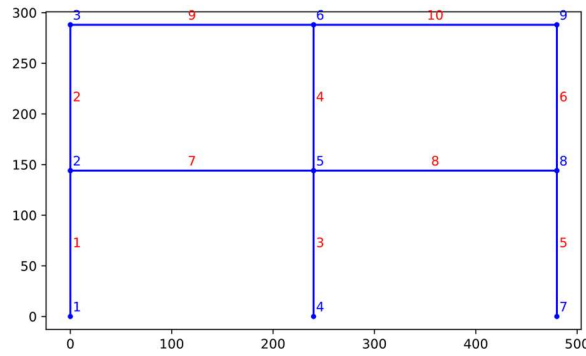


Figure 5: Agent-generated node numbering and element connectivity of a 2-story 2-bay steel frame analysis model

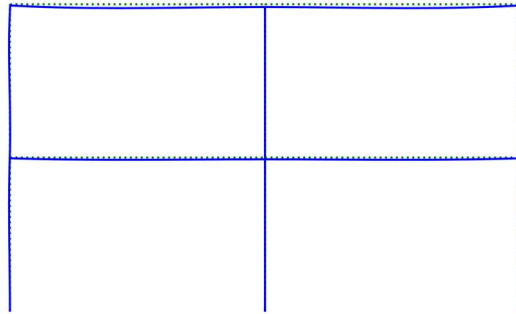


Figure 6: Visualization of the result of the agent-generated OpenSeesPy code analysis (scale factor =10)

4. Discussion on Challenges

4.1 Data Privacy

Although the authors chose the API of commercial LLMs as the technology stack for efficiency purposes, there are concerns regarding data privacy for this strategy because the data and backend support modules can be accessed by commercial LLMs. There are a few potential solutions: (1) use fully local-hosted open-source LLMs or run the agent inside locked-down containers, which eliminates cloud data leakage and gives you full control over where inference happens; (2) ensure strict access control and authentication, for example, individual accounts can opt-out of data training in ChatGPT and business accounts can employ enterprise credentials to activate commercial data protection; (3) secure all model and RAG data storage, which will protect model weights and embeddings on encrypted local volumes and ensure RAG queries operate in isolated spaces.

4.2 Hallucination

Although LLM-powered agents demonstrate great potential for automation, there are still significant “trust” barriers: LLMs are known to “hallucinate” numerical values, which is unacceptable in safety-critical structural steel design. Human-in-the-loop (HITL) architectures are a must in practice to provide accountability and safety.

In addition to the HITL, three strategies are employed in this research to enforce the clear boundary of the agent prototype’s generated content and increase trustworthiness: three modules are supported by customized Python code, a pre-trained ML model, and a validated FEM tool written in Python code. There are other strategies authors would like to explore in the future, including physics-informed methods.

This hallucination gap prevents LLM-powered agents from reliably recognizing the boundaries of their own competence and from appropriately declining to answer questions when they do not know what they do not know (hallucination). Future LLM-powered agent research on structural steel design should also emphasize the quantification of the assessment of the LLM-powered agent's knowledge boundary and stop it from generating unreliable answers to what it does not know.

5. Conclusions and Future Work

This research builds up a trustworthy large language model (LLM)-powered agent prototype to automate structural steel design, which generates steel building framing geometry, calculates steel column strength by calling a pre-trained neural network model, and automates structural analysis OpenSeesPy model creation. This agent prototype identifies the user's intention through chat messages and successfully accesses corresponding functional modules. And the three modules are supported by customized code, a pre-trained ML model, and a validated FEM tool written in Python code, which advances the trustworthiness of the agent prototype.

The agent prototype is still under development, and more work is required in the future. The accuracy, efficiency, robustness, reproducibility, and user interaction of the prototype agent-based steel building design will be evaluated with an AISC steel-framed building archetype example, which will further ensure that the agent prototype is verifiable and trustworthy.

Additional research work on data privacy, Human-in-the-loop (HITL) architecture, application of physics-informed methods, and quantification of the LLM-powered agent's knowledge boundary is expected.

Acknowledgments

The research presented in this paper is supported by EngineerWithData, a collective dedicated to data-driven engineering analysis and design. To facilitate further development and reproducibility, the source code and implementation details are publicly available in our GitHub repository: <https://github.com/EngineerWithData/LLM-steel-design>.

References

- ANSI/AISC. (2022). AISC 360-22: Specification for Structural Steel Buildings. <https://www.aisc.org/aisc/publications/current-standards/aisc-360/>
- ASCE. (2022). ASCE/SEI 7-22 Minimum Design Loads and Associated Criteria for Buildings and Other Structures. American Society of Civil Engineering. <https://sp360.asce.org/personifyebusiness/Merchandise/Product-Details/productId/276865145>
- Chen, J., & Bao, Y. (2025). Multi-agent large language model framework for code-compliant automated design of reinforced concrete structures. *Automation in Construction*, 177, 106331. <https://doi.org/10.1016/j.autcon.2025.106331>
- Geng, Z., Liu, J., Cao, R., Cheng, L., Wang, H., & Cheng, M. (2025). A Lightweight Large Language Model-Based Multi-Agent System for 2D Frame Structural Analysis (arXiv:2510.05414). arXiv. <https://doi.org/10.48550/arXiv.2510.05414>
- Liang, H., Kalaleh, M. T., & Mei, Q. (2025). Integrating Large Language Models for Automated Structural Analysis (arXiv:2504.09754). arXiv. <https://doi.org/10.48550/arXiv.2504.09754>
- Liang, H., Zhou, Y., Kalaleh, M. T., & Mei, Q. (2025). Automating Structural Engineering Workflows with Large Language Model Agents (arXiv:2510.11004). arXiv. <https://doi.org/10.48550/arXiv.2510.11004>

- Liu, J., Geng, Z., Cao, R., Cheng, L., Bocchini, P., & Cheng, M. (2025). A Large Language Model-Empowered Agent for Reliable and Robust Structural Analysis (arXiv:2507.02938). arXiv. <https://doi.org/10.48550/arXiv.2507.02938>
- Mazzoni, S., McKenna, F., Scott, M. H., & Fenves, G. L. (2006). OpenSees command language manual (pp. 137–158). Pacific Earthquake Engineering Research (PEER) Center.
- Sarker, P. K., Shoumik, S. C., Palit, S., Chowdhury, A. A. N., Alam, M. S., Gazi, M. D. H., & Rahman, M. (2025). Machine learning applications in predicting structural failures and earthquake damage. *International Journal of Research in Civil Engineering and Technology*, 6(1), 36–45. <https://doi.org/10.22271/27078264.2025.v6.i1a.79>
- Silvia, M. (2024). The OpenSeesPy Library—OpenSeesPy 3.5.1.3 documentation. <https://openseespydoc.readthedocs.io/en/latest/index.html>
- Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., & Wang, C. (2023). AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation (arXiv:2308.08155). arXiv. <https://doi.org/10.48550/arXiv.2308.08155>
- Yu, H., Wang, Y., Cheng, T., Yan, Y., Dawson, K. A., Li, S. F. Y., Zheng, Y., & Jin, Y. (2025). Empowering scientific discovery with explainable small domain-specific and large language models. *Artificial Intelligence Review*, 58(12), 371. <https://doi.org/10.1007/s10462-025-11365-w>
- Zhang, Z., Chen, K., Cevik, C. B., & Moen, C. D. (2025). Mechanics-informed data-driven prediction model of steel column strength. Proceedings of the Annual Stability Conference Structural Stability Research Council.