

Automating Structural Engineering Workflows with Large Language Model Agents

Haoran Liang^{1*}, Yufa Zhou^{2*}, Mohammad Talebi-Kalaleh¹, Qiwei Mei¹

¹University of Alberta, ²Duke University

{hliang7, talebika, qiwei.mei}@ualberta.ca, yufa.zhou@duke.edu

We introduce **MASSE**, the first Multi-Agent System for Structural Engineering, effectively integrating large language model (LLM)-based agents with real-world engineering workflows. Structural engineering is a fundamental yet traditionally stagnant domain, with core workflows remaining largely unchanged for decades despite its substantial economic impact and global market size. Recent advancements in LLMs have significantly enhanced their ability to perform complex reasoning, long-horizon planning, and precise tool utilization—capabilities well aligned with structural engineering tasks such as interpreting design codes, executing load calculations, and verifying structural capacities. We present a proof-of-concept showing that most real-world structural engineering workflows can be fully automated through a training-free LLM-based multi-agent system. MASSE enables immediate deployment in professional environments, and our comprehensive validation on real-world case studies demonstrates that it can reduce expert workload from approximately two hours to mere minutes, while enhancing both reliability and accuracy in practical engineering scenarios.

Code: <https://github.com/DelosLiang/masse>

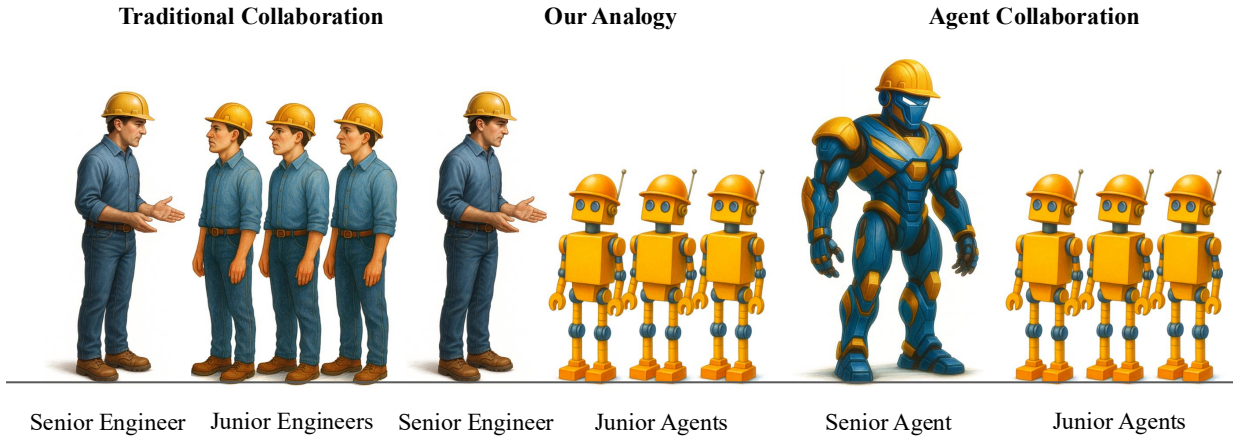


Figure 1: Analogy of future human-AI collaborations. Traditional practice relies on long apprenticeships, with senior engineers transferring expertise to junior engineers through mentorship and problem solving. In contrast, LLM-based multi-agent systems instantiate scalable junior engineer agents that inherit workflows, perform specialized tasks, and coordinate under senior engineers’ oversight. As these systems evolve toward self-planning and adaptive learning, fully agentic hierarchies—with senior engineer agent directing junior engineer agents—could transform engineering into a continuously improving, highly efficient practice.

*Equal contribution.

1 Introduction

Structural engineering, the discipline responsible for designing and building of structures including buildings, bridges, and other infrastructure, has shaped human civilization for millennia—from pyramids and aqueducts to skyscrapers and transportation networks [58, 1]. What began as intuition and craftsmanship has become a science grounded in mechanics, materials, and computation [57, 4]. Today, engineers translate ambiguous requirements into models, simulate loading scenarios with specialized software (e.g., OpenSees, SAP2000, Abaqus/CAE) [45, 51, 73], and verify compliance against extensive building codes [47, 12, 55, 67]. Yet despite underpinning a multi-trillion-dollar global sector [9], structural engineering remains among the least digitalized industries, hindered by fragmented workflows, manual knowledge transfer, and coordination bottlenecks [6, 3, 36]. The result is inefficiency, cost overruns, and lost opportunities for sustainability and resilience.

In parallel, large language models (LLMs) have transformed AI. Scaling transformer-based architectures [70] to billions of parameters yields broad generalization [10], predictable scaling laws [33, 29], and emergent competencies suggestive of general intelligence [11, 24]. Advances such as chain-of-thought prompting [75, 35] and instruction tuning [50] have unlocked strong reasoning, while frontier models—GPT-4o [48], GPT-5 [49], Qwen-3 [82], DeepSeek-R1 [27]—achieve unprecedented performance in math, code, and tool use. The trajectory has shifted toward *agents*: LLM systems that plan, invoke tools, and coordinate tasks [83, 65]. Frameworks such as ReAct [85], Tree of Thoughts [84], Voyager [74], and StateFlow [78] externalize reasoning into structured workflows, enabling deployment in domains that are procedural, codified, and tool-centric.

Structural engineering tasks, particularly structural design, exemplifies a domain where tasks are verbalizable, procedural, and tool-centric: requirements map to models, load cases to simulations, and code clauses to compliance checks. Single LLMs can aid structural analysis [40], yet their accuracy collapses when tasks demand multi-tool calls or chained sub-tasks, especially with complex geometries, underscoring the need for a resilient multi-agent framework (see Figure 2). We present **MASSE**, a *multi-agent system* tailored to structural design tasks. MASSE operationalizes professional workflows by assigning specialized LLM agents to distinct roles—*Analyst* (data and code extraction), *Engineer* (modeling and limit-state checks), and *Manager* (coordination and final adequacy decision). These roles are orchestrated through the AutoGen [77], supported by structured memory for persistent analysis data. By embedding FEM solvers and code documents directly in the loop, MASSE achieves stable tool-augmented reasoning, formal safety verification, and reduces hours of expert iteration to minutes. Therefore, we hypothesize that such agent-driven workflows can generalize beyond structural engineering to other domains where tasks are verbalizable, procedural, and tool-mediated (see Figure 1). Our contributions are:

- We pioneer an LLM-based multi-agent system that mirrors end-to-end structural design workflows with explicit safety and verification loops.

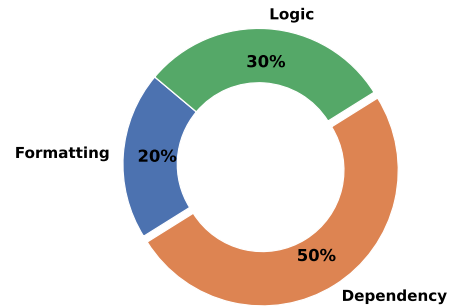


Figure 2: **Failure Modes of Single-Agent.** We evaluate the same structural engineering problem over 10 trials, comparing single-agent and multi-agent setups. The single-agent failed in all 10 trials, with error distributed as shown in the figure. In contrast, our multi-agent framework succeeded in every trial. See details in Appendix A.

- We introduce a new dataset and case studies grounded in real-world problems, demonstrating automation of complex, tool-mediated tasks.
- We demonstrate substantial reductions in expert time while preserving accuracy, supporting the broader thesis that verbalizable, tool-centered professional workflows can be automated at scale.

2 Related Work

2.1 LLMs in Civil Engineering Applications

LLMs have been extensively applied in Civil Engineering and related domains [80]. For instance, natural language-to-code frameworks have advanced structural optimization [54], prompting and in-context learning have enhanced structural analysis pipelines [40, 7, 44, 26], and multi-agent router systems have improved foundation design automation [87]. LLM-driven frameworks further advanced code-compliant reinforced concrete design [15] and LLM-based BIM authoring systems streamlined modeling tasks [21, 20]. Multi-agent collaboration has also been applied to Ultra-High Performance Concrete (UHPC) design, enabling knowledge-guided and data-driven material development [28]. Vision-language systems contributed to safety and compliance through ergonomic risk assessment [22], structured hazard detection [2], and RAG-based engineering code consultation fine-tuned on NBCC data [32, 5]. There are emerging benchmarks designed for civil engineering tasks such as drafting [38], strength of materials [72], and structural analysis [71], aiming to systematically evaluate LLM-driven automation in engineering workflows.

2.2 Multi-Agent Systems with LLMs

Recent advancements in multi-agent systems (MAS) have demonstrated the potential of LLM-powered agents to collaborate effectively across domains such as chemistry [68], healthcare [59], and finance [79], where specialized roles improve domain-specific performance. In software engineering, frameworks like ChatDev [53], HyperAgent [52], MetaGPT [30], and Magentic-One [25] coordinate agents for design, coding, and testing, while general-purpose infrastructures such as AutoGen [77] and ApWorld [69] provide flexible environments for orchestration and benchmarking. At the same time, researchers have begun systematically analyzing MAS limitations, highlighting failure modes [14, 46], stressing the need for rigorous benchmarking practices [34], and proving the potential of self-evolution [86]. Complementary to these system-level advances, efforts to enhance the efficiency and interpretability of LLMs themselves include model compression [43, 37, 41, 64], inference acceleration techniques [62, 63, 61, 31, 60], and theoretical analyses of their representational power [17, 18, 42].

3 System Role Design

In this section, we introduce the specific agent role design of our MASSE framework. We clearly define roles and specific goals for LLM agents, enabling complex structural engineering tasks to be efficiently segmented into smaller, manageable components. Structural engineering is inherently multifaceted, requiring diverse inputs, specialized knowledge, and coordinated expertise, such that professional practice relies heavily on multidisciplinary teams to collaboratively address complex, high-stakes decisions [76, 19, 88, 8, 16, 56], a workflow that LLM agents are well positioned to replicate and automate.

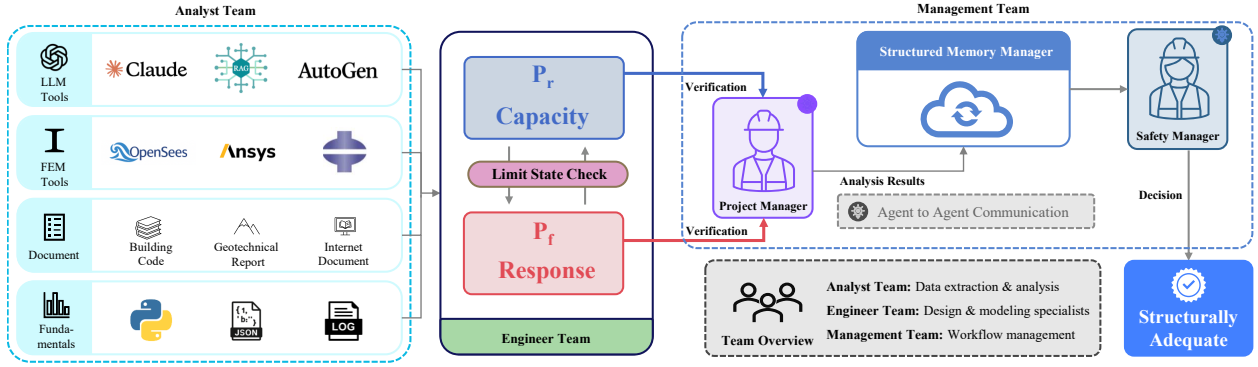


Figure 3: *MASSE* Overall Framework. The **Analyst Team** combines four layers of tools (LLMs, FEM solvers, engineering documents, and fundamentals) to enable multi-agent collaboration. The **Engineer Team** performs limit state verification ($\text{Response} < \text{Capacity}$), while the **Project Manager** coordinates workflows and the **Structural Memory** stores analysis data. The **Safety Manager** conducts the final adequacy check. Three specialized teams are shown: **Analyst** (data extraction and analysis), **Engineer** (design and verification), and **Management** (coordination and decision-making).

3.1 Traditional Workflow

In conventional structural engineering practice, the design process is carried out through a sequence of tightly coupled steps, beginning with consulting building codes to obtain site-specific seismic parameters or climatic data. Engineers then collect geometric information from site measurements or architectural drawings and determine applied loads—including dead loads, live loads, and other types of loads—using code-based occupancy provisions and spreadsheets or rule-based calculations. With these inputs, they evaluate the load-bearing capacities of structural members from section dimensions and material properties, and use either hand calculations or finite element models to simulate structural responses under multiple load combinations. If requirements are not met, the workflow must be repeated with adjusted assumptions or revised sections. For specialized problems such as warehouse racking systems, this process may take several hours per rack, and with multiple racks per site, the workload becomes extensive. Overall, the traditional workflow is time-consuming, error-prone, and difficult to scale, motivating the need for automation frameworks such as *MASSE*.

3.2 Our System

Reflecting this practical organizational model, *MASSE* (Figure 3) introduces three distinct agent teams within a simulated structural engineering consultancy environment: Analyst Team, Engineer Team and Management Team. For comprehensive descriptions of the system instructions assigned to each agent, see Appendix B. Each agent is assigned a unique role, goal, and set of constraints, and is further equipped with predefined contexts and specialized tools aligned with these responsibilities. We will then introduce how *MASSE* organizes these agent roles into the following structured teams. More detailed descriptions could be found in Appendix C.

3.3 Analyst Team

The Analyst Team (Figure 4) automates the preparation of structural engineering data by coordinating specialized agents that extract loading conditions, retrieve information from engineering documenta-

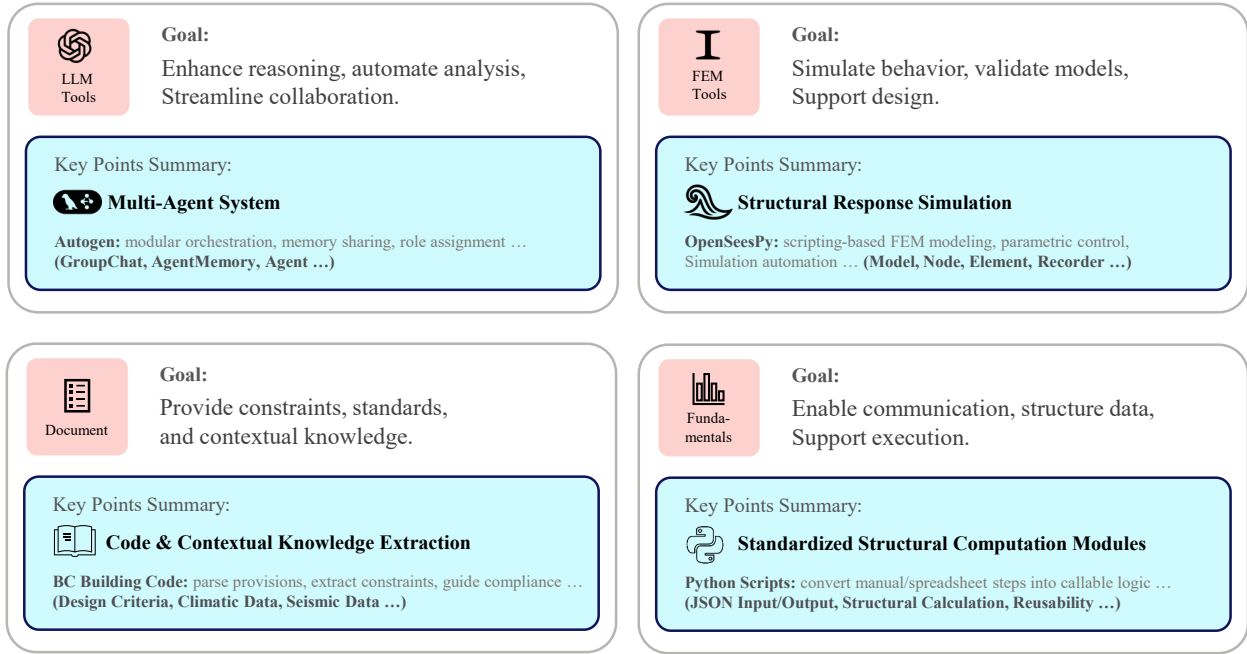


Figure 4: MASSE Analyst Team with Tool Integration

tion, execute load determination using rule-based methods, and generate structural models. At a high level, this team transforms unstructured project information and regulatory data into standardized engineering inputs, ensuring that subsequent model, design and verification tasks can be carried out with consistency, efficiency, and scalability.

3.4 Engineer Team

The Engineer Team (Figure 5) operationalizes the data prepared by the Analyst Team by conducting structural analysis, structural design, and adequacy verification. In broad terms, this team integrates automated simulations and capacity checks to evaluate structural integrity under prescribed loading conditions, enabling systematic, tool-driven assessment of design safety and structural performance.

3.5 Management Team

The Management Team oversees and coordinates the overall MASSE workflow, transforming analytical outputs from the Analyst Team and Engineer Team into authoritative engineering decisions. Broadly speaking, this team manages task allocation, integrates intermediate results, and issues final structural safety conclusions that guide the entire system. As illustrated in Figure 6, the Safety Manager plays a central role within this process by delivering the ultimate adequacy verdict, ensuring that all decisions are consistent with professional safety standards.

4 Agentic Workflow Details

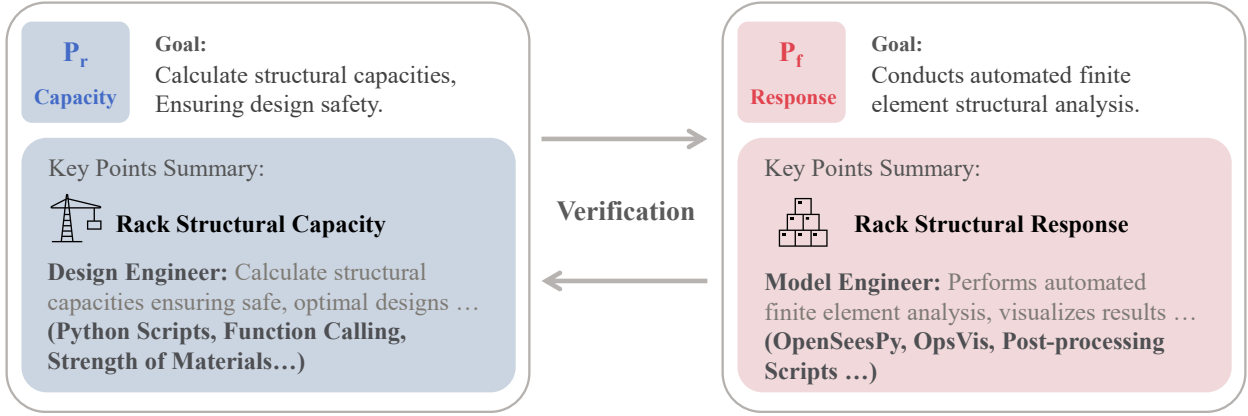


Figure 5: MASSE Engineer Team: Analyze, Design and Verify

4.1 Backbone Models

In our system, agents are configured to be initialized with powerful LLMs (e.g., GPT-4o, Claude 3.5 Sonnet), or reasoning models (e.g., o4-mini). We also use Embedding models (e.g., text-embedding-3) to support retrieval-augmented generation for accessing building codes and technical documents. Temperatures are set to 0 for GPT-4o, Claude 3.5 Sonnet, and GPT-3.5-turbo, and 1 for o4-mini.

4.2 Communication Protocol

MASSE prioritizes structured-format communication over verbose natural language exchanges. Natural language reliance often degrades performance in structural engineering tasks that require extended planning horizons, as verbose dialogue can overflow context windows and obscure key details. To address these issues, MASSE adopts structured communication paradigms, where each agent’s operational state is defined through formalized protocols, such as JSON-based input–output schemas and turn-level state tracking mechanisms. This ensures concise, verifiable outputs while reducing redundancy and distortion.

4.3 Agent Interactions

We design agent interactions around structured artifacts that encode analytical results, ensuring traceability and systematic integration. We enforce predefined protocols with input–output formats, schemas, and validation rules to eliminate ambiguity and enable orchestration. In this pipeline, we let analysts convert noisy problem descriptions and unstructured documents into structured memory, engineers transform them into deterministic outputs such as finite element analysis results and capacity checks, and management roles validate outputs against codes before authorizing final decisions. When inconsistencies occur, we allow brief natural language exchanges to recover missing information, but all outcomes are distilled back into structured form to preserve accountability. We log every exchange for auditing and debugging, and the overall framework mirrors professional engineering workflows where domain-specific expertise is coordinated through standardized communication. A detailed trajectory of a racking system case is provided in Appendix D.

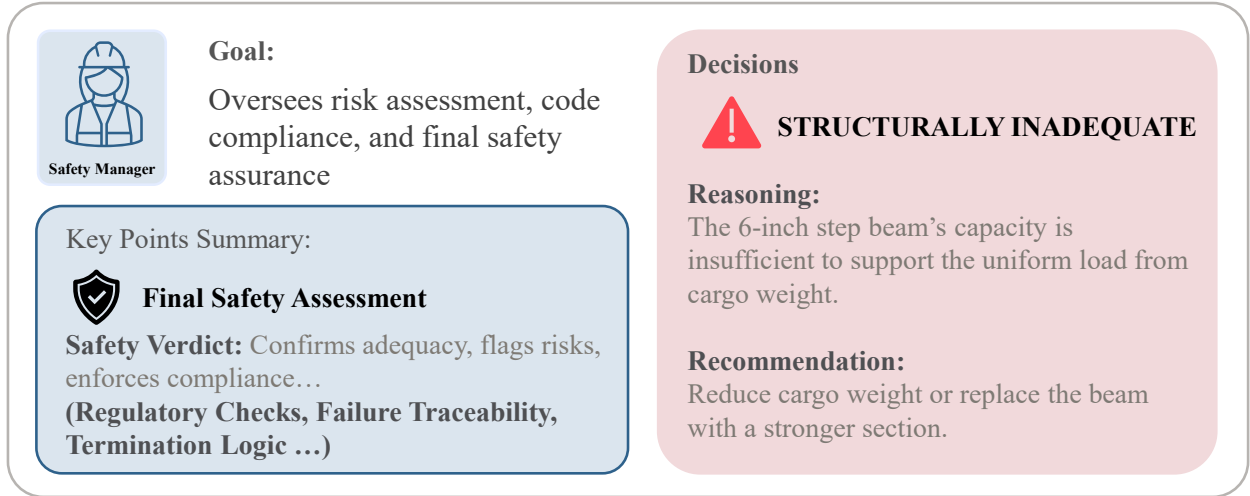


Figure 6: MASSE’s Safety Manager’s Decision-Making Process

5 Experiments

5.1 Simulation Setup

We evaluate MASSE in a racking system design scenario [13], a representative task in structural engineering where engineers must determine allowable loads and ensure compliance with seismic safety requirements. This setup integrates core engineering steps—data retrieval, load transformation, structural modeling, analysis, and design—directly aligning with the functionality of MASSE agents. Full details of the simulation configuration are provided in Appendix E.1.

5.2 Dataset

We constructed a domain-specific dataset consisting of one hundred different levels of difficulty, each paired with validated ground-truth solutions, enabling robust evaluation reflective of real-world engineering challenges. To comply with privacy constraints, the released dataset is reorganized but faithfully mirrors the original cases, ensuring reproducibility of system performance. Each sample contains a natural language problem description, intermediate reasoning steps, and final results used as evaluation criteria. Notably, loading reports are mandated and must be certified by structural engineers in earthquake-prone regions such as British Columbia and California, underscoring the practical significance of this scenario. Our modular design allows straightforward adaptation to many structural engineering tasks, making them broadly applicable as plug-and-play components. An illustrative racking system problem and its structural scheme are provided in Appendix F.

5.3 Evaluation Metrics

To benchmark MASSE against real-world structural engineering tasks, we design novel evaluation metrics aligned with the core agent roles: Structural Analysis Agent Benchmark (SAAB), Structural Design Agent Benchmark (SDAB), Loading Agent Benchmark (LAB), and the holistic Multi-Agent Structural Engineering Benchmark (MASEB). Each benchmark evaluates the ability to convert natural language

inputs into structured analyses, tool executions, and safety decisions, while MASEB additionally incorporates system cost and runtime to balance efficiency and performance. To ensure these benchmarks reflect professional practice, we base our evaluation on data reorganized from real-world projects in British Columbia, Canada. This design allows the metrics to capture both technical accuracy and the fidelity with which agents replicate the workflow of a consulting firm. Detailed rubrics, task specifications, and evaluation protocols are provided in Appendix E.2.

6 Results and Analysis

6.1 Main Results

6.1.1 Performance Comparison

Table 1 summarizes MASSE performance across four benchmarks under different LLM backends. Bold numbers denote the best model on each benchmark, while underlined numbers indicate the second best. Each problem was evaluated over ten independent trials, and a total of one hundred traces were collected to assess the overall system performance. Among language models, *Claude 3.5 Sonnet* delivers the strongest overall consistency, leading in SDAB (89.2) and remaining competitive elsewhere, while *GPT-4o* achieves the highest LAB score (98.1) but trails slightly in SAAB and MASEB. By contrast, *GPT-3.5-turbo* underperforms (e.g., 64.6 in SDAB, 67.7 in MASEB).

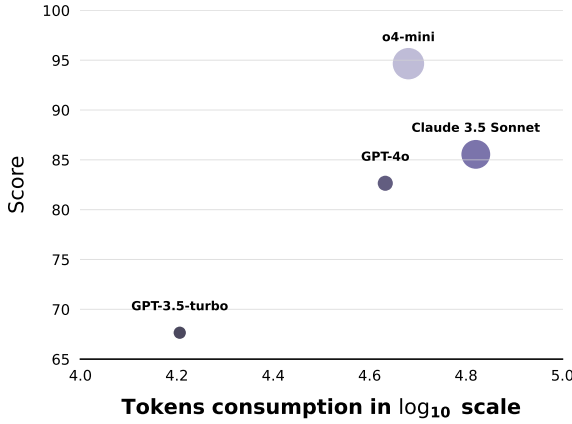
Table 1: Performance Comparison of MASSE under Different LLMs. The best score per benchmark is in bold, the second-best is underlined. Avg. is reported across four benchmarks.

Models	SAAB	SDAB	LAB	MASEB	Avg.
Language Models					
<i>Claude 3.5 Sonnet</i>	<u>87.0</u>	<u>89.2</u>	94.8	<u>85.6</u>	<u>89.2</u>
<i>GPT-3.5-turbo</i>	71.4	64.6	90.5	67.7	73.6
<i>GPT-4o</i>	85.6	87.4	98.1	82.7	88.5
Reasoning Models					
<i>o4-mini</i>	96.6	91.4	<u>93.8</u>	94.7	94.1

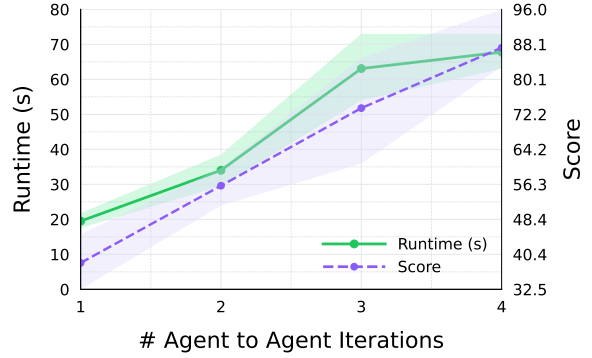
Reasoning model *o4-mini* outperforms across three of four benchmarks, with peak results in SAAB (96.6), SDAB (91.4), and MASEB (94.7), and only narrowly behind *GPT-4o* in LAB (93.8 vs. 98.1). This balance highlights the intelligence and robustness of reasoning models, showing that reasoning yields higher reliability across heterogeneous tasks. Overall, the results indicate that (1) reasoning models like *o4-mini* provide the most stable and generalizable performance, and (2) among standard LLMs, large-scale models such as *Claude 3.5 Sonnet* and *GPT-4o* remain essential for high-fidelity structural engineering workflows.

6.1.2 Cost Analysis

Figure 7a illustrates a clear performance–efficiency trade-off. *o4-mini* achieves the best overall score but with the heaviest computational cost—both in runtime and token consumption. *Claude 3.5 Sonnet* performs slightly worse while also incurring substantial token usage and longer runs. *GPT-4o* offers a balanced middle ground, delivering strong accuracy at a moderate cost. In contrast, *GPT-3.5-turbo* is



(a) **Cost analysis.** Bubble size denotes average runtime count per method; y-axis shows average score across four datasets.



(b) **Runtime.** X-axis: agent communication rounds; left y-axis: total runtime; right y-axis: test performance. The shaded regions indicate the 95% CI computed via bootstrap.

Figure 7: Results Analysis: (a) cost–performance trade-off and (b) runtime–performance relationship.

the most economical and fastest, though its performance lags behind the others. These patterns suggest that MASSE benefits from stronger backbones when quality is paramount, but real-world deployment must balance performance with latency and budget.

6.1.3 Runtime Analysis

In this experiment, we vary the maximum number of agent-to-agent communication rounds, which is set to 1 to 4. Allowing more rounds of communication enables agents to exchange additional information, identify and fill in missing details, refine intermediate reasoning steps, and apply self-correction before producing final outputs. The evaluation is performed on ten representative problems selected from MASEB. To mitigate randomness in model behavior and ensure robust measurement, each problem instance is independently repeated ten times.

The runtime analysis (Figure 7b) demonstrates the fundamental trade-off in multi-agent coordination. As the number of agent-to-agent communication rounds increases from one to four, runtime rises steadily—from about 20 seconds at a single round to nearly 70 seconds at four rounds—reflecting the additional overhead of iterative dialogue. However, this increase in computational effort correlates with consistent improvements in system score, which grows from below 40 to nearly 90. These results highlight a key feature of multi-agent systems: while deeper rounds of communication impose latency, they also enable richer reasoning and more reliable outputs.

6.2 Human Evaluation

To empirically evaluate MASSE’s efficiency, we conducted a comparative study with 11 experienced structural engineers, each independently completing a standardized racking system design task. The conventional workflow—dependent on fragmented software, manual data retrieval, repetitive entry, and interpretive effort—required an average of 132 minutes per task. By contrast, MASSE, powered by GPT-4o, automated these processes and reduced the average completion time to approximately two minutes, a reduction of over 98%. This result demonstrates a marked improvement in both productivity

and operational reliability within professional practice. In consulting environments, executing tasks at a fraction of the traditional time and cost (on the order of 1/100) is likely to enhance profit margins, streamline organizational workflows, and redefine competitive positioning. More broadly, firms that strategically adopt and effectively integrate AI systems such as MASSE may not only secure substantial advantages in the engineering sector, but also be positioned to shape new paradigms of practice, potentially influencing how design, safety, and efficiency are balanced across the built environment.

6.3 Ablation Study

Table 2: Ablation study on agent components: agent memory M and JSON format J .

Agent Component	SAAB	SADB	LAB	MASEB	Avg.
None	52.5	65.9	81.6	47.1	61.8
+ M	55.9	<u>69.8</u>	<u>89.6</u>	50.6	<u>66.5</u>
+ J	<u>56.6</u>	67.2	84.1	<u>50.1</u>	64.5
+ M, J	85.6	87.4	98.1	82.7	88.5

Agent memory and structured input–output are increasingly recognized as common practices and active research directions in the design of multi-agent systems [81, 39, 23]. To assess the effect of individual agent components, we performed an ablation study across four benchmarks (Table 2). The baseline configuration without memory or structured I/O yields the lowest scores in all settings. Incorporating multi-agent memory (M) improves performance by preserving intermediate reasoning traces across turns, with notable gains in LAB (81.6 \rightarrow 89.6). Similarly, enforcing JSON-based input–output constraints (J) provides consistent benefits by reducing ambiguity in communication, especially in SAAB and SADB. The combination of both components (+ M, J) achieves the strongest results overall, setting the best score on every benchmark. These findings indicate that memory and structured I/O constraints are crucial: memory enhances contextual continuity, while JSON formalization enforces precise data exchange.

7 Discussion

Transparency. Reliability and accountability are prerequisites for engineering adoption. MASSE is therefore designed with verifiable processes rather than opaque autonomy. Every artifact is logged with explicit reasoning traces, giving practitioners immediate access to the system’s decision path and enabling systematic validation. Deterministic outputs further support reproducible quality checks, while localized error handling ensures that small mistakes do not propagate into larger failures. By structuring agents to follow well-defined workflows, MASSE effectively plays the role of a junior engineer whose steps are fully auditable—providing transparency and traceability without conceding control to uncontrolled autonomy.

Safety. Because structural engineering involves high-stakes decisions, ultimate responsibility must remain with senior practitioners; even small misjudgments can produce catastrophic outcomes. MASSE is designed to augment, not replace, this expertise by compressing the end-to-end review process from roughly two hours to only a few minutes (Section 6.2). This acceleration allows experts to evaluate a wider range of design alternatives and stress scenarios within the same timeframe, thereby improving

robustness and resilience of built systems. Equally important, MASSE produces structured intermediate outputs that mirror professional practice, akin to junior engineers drafting reports for senior verification. As AI systems continue to improve, such workflows may increasingly reduce reliance on traditional junior roles, reallocating human attention to the oversight and judgment that matter most for public safety.

Real-World Impact. The efficiency gains of MASSE highlight how domain-specific multi-agent systems can convert the raw capability of frontier LLMs into transformative productivity advantages. Reducing structural design cycles from hours to minutes (Section 6.2) does more than boost users’ competitiveness—it has the potential to alter the economics of engineering services by lowering costs, increasing throughput, and enabling firms to take on larger, more complex projects. Freed from routine procedures, engineers can concentrate on creativity, innovation, and safety-critical deliberation. At scale, such reallocation of effort has implications well beyond structural engineering: knowledge-intensive sectors such as architecture, finance, and healthcare could be reorganized around agentic AI collaborators, yielding faster design cycles, more equitable access to expertise, and accelerated progress on global challenges from sustainable infrastructure to resilient urban development. MASSE thus exemplifies how carefully engineered AI systems can bridge the gap between technical advances in LLM orchestration and tangible societal benefit.

8 Conclusion

We introduced **MASSE**, a multi-agent framework designed specifically for structural engineering, integrating LLM-based agents into end-to-end professional workflows. By coordinating role-specialized agents, MASSE is able to parse technical standards, perform domain-specific computations, and interact seamlessly with engineering scripts and resources. Its structured communication backbone and iterative reasoning cycle allow the system to achieve robust, high-quality outcomes. Safety verification is embedded through a dedicated management role, and the modular design enables flexible adaptation across a wide range of engineering tasks. Our evaluation on real-world case studies shows that MASSE can reduce expert workload by approximately 98%, cutting the required time from hours to minutes while maintaining both reliability and accuracy. This proof-of-concept demonstrates the feasibility of automating structural engineering workflows without task-specific training. Looking ahead, future work will emphasize deployment in consulting environments, refinement of agent specialization, and integration of real-time feedback to support adaptive self-improvement.

Ethic Statement

Research and Educational Purpose Only. The multi-agent system, methods, code, and data described in this paper are developed solely for academic research and educational use. They are not intended, nor should they be relied upon, for direct application in real-world engineering design, construction, or deployment. The authors, their institutions, and any collaborators explicitly disclaim all liability for any consequences, including but not limited to structural deficiencies, under-design, damages, or failures, arising from the use, misuse, or adaptation of the methods, parameters, or formulas presented herein. Any attempt to implement or deploy the system in practice is done entirely at the user’s own risk and responsibility.

Human Evaluation. All human evaluation conducted in this study was performed exclusively for the purpose of benchmarking and comparative analysis. Data generated by participants is kept strictly confidential, used only within the scope of this research, and will not be applied to any commercial, industrial, or real-world engineering activities.

Privacy and Data Protection. The dataset employed in this work is derived from production records within structural engineering consultancy practice but has been anonymized, cleaned, and used strictly for academic research. No identifying information, proprietary designs, or sensitive client data are disclosed. All outputs generated by the multi-agent system remain confined to research purposes and will not be released, applied, or repurposed for operational engineering, deployment, or commercial use. This research adheres to applicable data privacy principles and safeguards the rights and interests of all parties involved.

Acknowledgement

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

References

- [1] William Addis. *Building: 3000 years of design engineering and construction*, volume 601. Phaidon London, 2007.
- [2] Muhammad Adil, Gaang Lee, Vicente A Gonzalez, and Qipei Mei. Using vision language models for safety hazard identification in construction. *arXiv preprint arXiv:2504.09083*, 2025.
- [3] Rajat Agarwal, Shankar Chandrasekaran, and Mukund Sridhar. Imagining construction’s digital future. *McKinsey & Company*, 24(06):1–13, 2016.
- [4] Edward Allen. *Fundamentals of building construction*, 2001.
- [5] Mohammad Aqib, Mohd Hamza, Qipei Mei, and Ying Hei Chui. Fine-tuning large language models and evaluating retrieval methods for improved question answering on building codes. *arXiv preprint arXiv:2505.04666*, 2025.
- [6] ASCE. Confirmation and update: A vision for the future of structural engineering and structural engineers: A case for change. Task committee report, American Society of Civil Engineers, Structural Engineering Institute, April 2019. URL <https://www.asce.org/-/media/asce-images-and-files/communities/institutes-and-technical-groups/structural-engineering/documents/sei-2019-vision-task-committee-report.pdf>. Accessed 2025-08-16.
- [7] Carlos Avila, Daniel Ilbay, and David Rivera. Human–ai teaming in structural analysis: A model context protocol approach for explainable and accurate generative ai. *Buildings*, 15(17):3190, 2025.
- [8] Olaitan Awomolo et al. Exploring multiple disciplines in building design practice. *Enquiry The ARCC Journal for Architectural Research*, 16(1):29–45, 2019.

- [9] Filipe Barbosa, Jonathan Woetzel, and Jan Mischke. Reinventing construction: A route of higher productivity. Technical report, McKinsey Global Institute, 2017.
- [10] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.
- [11] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.
- [12] Canadian Institute of Steel Construction. *Handbook of Steel Construction*. Canadian Institute of Steel Construction, Willowdale, Ontario, 1992.
- [13] Carlo Andrea Castiglioni et al. *Seismic behavior of steel storage pallet racking systems*. Springer, 2016.
- [14] Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*, 2025.
- [15] Jinxin Chen and Yi Bao. Multi-agent large language model framework for code-compliant automated design of reinforced concrete structures. *Automation in Construction*, 177:106331, 2025.
- [16] Wai-Fah Chen and JY Richard Liew. *The civil engineering handbook*. Crc Press, 2002.
- [17] Yifang Chen, Xiaoyu Li, Yingyu Liang, Zhenmei Shi, and Zhao Song. The computational limits of state-space models and mamba via the lens of circuit complexity. In *The Second Conference on Parsimony and Learning (Proceedings Track)*, 2025. URL <https://openreview.net/forum?id=bImLLT3r62>.
- [18] Yifang Chen, Xiaoyu Li, Yingyu Liang, Zhenmei Shi, and Zhao Song. Universal approximation of visual autoregressive transformers. *arXiv preprint arXiv:2502.06167*, 2025.
- [19] Jerome J Connor and Susan Faraji. *Fundamentals of structural engineering*. Springer, 2016.
- [20] Zihan Deng, Changyu Du, Stavros Nouisias, and André Borrmann. Bimgent: Towards autonomous building modeling via computer-use agents, 2025. URL <https://arxiv.org/abs/2506.07217>.
- [21] Changyu Du, Sebastian Esser, Stavros Nouisias, and André Borrmann. Text2bim: Generating building models using a large language model-based multi-agent framework, 2025. URL <https://arxiv.org/abs/2408.08054>.
- [22] Chao Fan, Qipei Mei, Xiaonan Wang, and Xinming Li. Ergochat: a visual query system for the ergonomic risk assessment of construction workers. *arXiv preprint arXiv:2412.19954*, 2024.
- [23] Shiqing Fan, Xichen Ding, Liang Zhang, and Linjian Mo. Mcptoolbench++: A large scale ai agent model context protocol mcp tool use benchmark. *arXiv preprint arXiv:2508.07575*, 2025.
- [24] Tao Feng, Chuanyang Jin, Jingyu Liu, Kunlun Zhu, Haoqin Tu, Zirui Cheng, Guanyu Lin, and Jiaxuan You. How far are we from agi: Are llms all we need? *Transactions on Machine Learning Research*, 2024.

- [25] Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, et al. Magentic-one: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468*, 2024.
- [26] Ziheng Geng, Jiachen Liu, Ran Cao, Lu Cheng, Haifeng Wang, and Minghui Cheng. A lightweight large language model-based multi-agent system for 2d frame structural analysis. *arXiv preprint arXiv:2510.05414*, 2025.
- [27] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- [28] Pengwei Guo, Zhan Jiang, Weina Meng, and Yi Bao. Multi-agent collaboration for knowledge-guided data-driven design of ultra-high-performance concrete (uhpc) incorporating solid wastes. *Cement and Concrete Composites*, page 106230, 2025.
- [29] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.
- [30] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 3(4):6, 2023.
- [31] Ting Jiang, Yixiao Wang, Hancheng Ye, Zishan Shao, Jingwei Sun, Jingyang Zhang, Zekai Chen, Jianyi Zhang, Yiran Chen, and Hai Li. Sada: Stability-guided adaptive diffusion acceleration. *arXiv preprint arXiv:2507.17135*, 2025.
- [32] Isaac Joffe, George Felobes, Youssef Elgouhari, Mohammad Talebi Kalaleh, Qipei Mei, and Ying Hei Chui. The framework and implementation of using large language models to answer questions about building codes and standards. *Journal of Computing in Civil Engineering*, 2025. ISSN 0887-3801. doi: 10.1061/JCCEE5.CPENG-6037.
- [33] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020. URL <https://arxiv.org/abs/2001.08361>.
- [34] Sayash Kapoor, Benedikt Stroebel, Zachary S Siegel, Nitya Nadgir, and Arvind Narayanan. Ai agents that matter. *arXiv preprint arXiv:2407.01502*, 2024.
- [35] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35: 22199–22213, 2022.
- [36] KPMG. Global construction survey 2019: Future-ready index – leaders and followers in the engineering & construction industry. Technical report, KPMG International Cooperative, April 2019. URL <https://assets.kpmg.com/content/dam/kpmg/xx/pdf/2019/04/global-construction-survey-2019.pdf>. Accessed 2025-08-16.
- [37] Tanishq Kumar, Zachary Ankner, Benjamin F Spector, Blake Bordelon, Niklas Muennighoff, Manish Paul, Cengiz Pehlevan, Christopher Ré, and Aditi Raghunathan. Scaling laws for precision. *arXiv preprint arXiv:2411.04330*, 2024.

- [38] Yinsheng Li, Zhen Dong, and Yi Shao. Drafterbench: Benchmarking large language models for tasks automation in civil engineering, 2025. URL <https://arxiv.org/abs/2507.11527>.
- [39] Zhiyu Li, Shichao Song, Chenyang Xi, Hanyu Wang, Chen Tang, Simin Niu, Ding Chen, Jiawei Yang, Chunyu Li, Qingchen Yu, et al. Memos: A memory os for ai system. *arXiv preprint arXiv:2507.03724*, 2025.
- [40] Haoran Liang, Mohammad Talebi Kalaleh, and Qipei Mei. Integrating large language models for automated structural analysis. *arXiv preprint arXiv:2504.09754*, 2025.
- [41] Yingyu Liang, Jiangxuan Long, Zhenmei Shi, Zhao Song, and Yufa Zhou. Beyond linear approximations: A novel pruning approach for attention matrix. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=sgbI8Pxwie>.
- [42] Yingyu Liang, Zhizhou Sha, Zhenmei Shi, Zhao Song, and Yufa Zhou. Looped relu mlps may be all you need as programmable computers. In *The 28th International Conference on Artificial Intelligence and Statistics*, 2025.
- [43] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems*, 6: 87–100, 2024.
- [44] Jiachen Liu, Ziheng Geng, Ran Cao, Lu Cheng, Paolo Bocchini, and Minghui Cheng. A large language model-empowered agent for reliable and robust structural analysis, 2025. URL <https://arxiv.org/abs/2507.02938>.
- [45] Frank McKenna. Opensees: a framework for earthquake engineering simulation. *Computing in Science & Engineering*, 13(4):58–66, 2011.
- [46] Microsoft Corporation. Taxonomy of failure modes in agentic ai systems: Whitepaper. Technical report, Microsoft, 2025. URL <https://cdn-dynmedia-1.microsoft.com/is/content/microsoftcorp/microsoft/final/en-us/microsoft-brand/documents/Taxonomy-of-Failure-Mode-in-Agentic-AI-Systems-Whitepaper.pdf>.
- [47] National Research Council of Canada. *National Building Code of Canada 2020*. National Research Council of Canada, Ottawa, Ontario, Canada, 2020. National Building Code of Canada (NBC).
- [48] OpenAI. Hello gpt-4o. <https://openai.com/index/hello-gpt-4o/>, 2024. Accessed: May 13.
- [49] OpenAI. Gpt-5 system card. Technical report, OpenAI, August 2025.
- [50] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 2022.
- [51] Laurent Pasticier, Claudio Amadio, and Massimo Fragiaco. Non-linear seismic analysis and vulnerability evaluation of a masonry building by means of the sap2000 v. 10 code. *Earthquake engineering & structural dynamics*, 37(3):467–485, 2008.
- [52] Huy Nhat Phan, Tien N Nguyen, Phong X Nguyen, and Nghi DQ Bui. Hyperagent: Generalist software engineering agents to solve coding tasks at scale. *arXiv preprint arXiv:2409.16299*, 2024.

- [53] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. *arXiv preprint arXiv:2307.07924*, 2023.
- [54] Sizhong Qin, Hong Guan, Wenjie Liao, Yi Gu, Zhe Zheng, Hongjing Xue, and Xinzheng Lu. Intelligent design and optimization system for shear wall structures based on large language models and generative artificial intelligence. *Journal of Building Engineering*, 95:109996, 2024.
- [55] Charles E Reynolds, James C Steedman, and Anthony J Threlfall. *Reinforced concrete designer's handbook*. CRC Press, 2007.
- [56] Brett Rocha, Aaron Hill, Nathan Hedgecock, Skylar Franz, Morgan Ernst, and Mark Sallot. Evidence of the benefits of interdisciplinary engineering teams: Incorporating systems engineering into civil engineering design. In *2022 ASEE Annual Conference & Exposition*, 2022.
- [57] Mario Salvadori. *Why buildings stand up: The strength of architecture*. WW Norton & Company, 1990.
- [58] Victor E Saouma. Structural engineering, analysis and design. *Lecture Notes*, 2010.
- [59] Samuel Schmidgall, Rojin Ziaei, Carl Harris, Eduardo Reis, Jeffrey Jopling, and Michael Moor. Agentclinic: a multimodal agent benchmark to evaluate ai in simulated clinical environments. *arXiv preprint arXiv:2405.07960*, 2024.
- [60] Zishan Shao, Yixiao Wang, Qinsi Wang, Ting Jiang, Zhixu Du, Hancheng Ye, Danyang Zhuo, Yiran Chen, and Hai Li. Flashsvd: Memory-efficient inference with streaming for low-rank models. *arXiv preprint arXiv:2508.01506*, 2025.
- [61] Xuan Shen, Chenxia Han, Yufa Zhou, Yanyue Xie, Yifan Gong, Quanyi Wang, Yiwei Wang, Yanzhi Wang, Pu Zhao, and Jiuxiang Gu. Draftattention: Fast video diffusion via low-resolution attention guidance. *arXiv preprint arXiv:2505.14708*, 2025.
- [62] Xuan Shen, Weize Ma, Yufa Zhou, Enhao Tang, Yanyue Xie, Zhengang Li, Yifan Gong, Quanyi Wang, Henghui Ding, Yiwei Wang, et al. Fastcar: Cache attentive replay for fast auto-regressive video generation on the edge. *arXiv preprint arXiv:2505.14709*, 2025.
- [63] Xuan Shen, Zhao Song, Yufa Zhou, Bo Chen, Yanyu Li, Yifan Gong, Kai Zhang, Hao Tan, Jason Kuen, Henghui Ding, et al. Lazydit: Lazy learning for the acceleration of diffusion transformers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 20409–20417, 2025.
- [64] Xuan Shen, Zhao Song, Yufa Zhou, Bo Chen, Jing Liu, Ruiyi Zhang, Ryan A Rossi, Hao Tan, Tong Yu, Xiang Chen, et al. Numerical pruning for efficient autoregressive models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 20418–20426, 2025.
- [65] David Silver and Richard S Sutton. Welcome to the era of experience. *Google AI*, 2025.
- [66] Akshit Sinha, Arvinth Arun, Shashwat Goel, Steffen Staab, and Jonas Geiping. The illusion of diminishing returns: Measuring long horizon execution in llms. *arXiv preprint arXiv:2509.09677*, 2025.
- [67] Judith Stalnaker and Ernest Harris. *Structural design in wood*. Springer Science & Business Media, 1997.

- [68] Xiangru Tang, Tianyu Hu, Muyang Ye, Yanjun Shao, Xunjian Yin, Siru Ouyang, Wangchunshu Zhou, Pan Lu, Zhuosheng Zhang, Yilun Zhao, et al. Chemagent: Self-updating library in large language models improves chemical reasoning. *arXiv preprint arXiv:2501.06590*, 2025.
- [69] Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. *arXiv preprint arXiv:2407.18901*, 2024.
- [70] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [71] Carlos Fabian Avila Vega, Daniel Jefferson Ibay Yupa, Paola Vanessa Tapia Tapia, and Edgar David Rivera Tapia. Toward responsible ai in high-stakes domains: A dataset for building static analysis with llms in structural engineering. 2025.
- [72] Qixin Wan, Zilong Wang, Jingwen Zhou, Wanting Wang, Ziheng Geng, Jiachen Liu, Ran Cao, Minghui Cheng, and Lu Cheng. Som-1k: A thousand-problem benchmark dataset for strength of materials. *arXiv preprint arXiv:2509.21079*, 2025.
- [73] Gong-Dong Wang and Stephen Kirwa Melly. Three-dimensional finite element modeling of drilling cfrp composites using abaqus/cae: a review. *The International Journal of Advanced Manufacturing Technology*, 94:599–614, 2018.
- [74] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.
- [75] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.
- [76] Karin Wolff and Kathy Luckett. Integrating multidisciplinary engineering knowledge. *Teaching in Higher Education*, 18(1):78–92, 2013.
- [77] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.
- [78] Yiran Wu, Tianwei Yue, Shaokun Zhang, Chi Wang, and Qingyun Wu. Stateflow: Enhancing llm task-solving through state-driven workflows. In *First Conference on Language Modeling*, 2024.
- [79] Yijia Xiao, Edward Sun, Di Luo, and Wei Wang. Tradingagents: Multi-agents llm financial trading framework. *arXiv preprint arXiv:2412.20138*, 2024.
- [80] Hao Xie, Qipei Mei, and Ying Hei Chui. Ai applications for structural design automation. *Automation in Construction*, 179:106496, 2025.
- [81] Zidi Xiong, Yuping Lin, Wenya Xie, Pengfei He, Jiliang Tang, Himabindu Lakkaraju, and Zhen Xiang. How memory management impacts llm agents: An empirical study of experience-following behavior. *arXiv preprint arXiv:2505.16067*, 2025.

- [82] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.
- [83] Shunyu Yao. The second half. <https://ysymyth.github.io/The-Second-Half/>, April 2025.
- [84] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.
- [85] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.
- [86] Xunjian Yin, Xinyi Wang, Liangming Pan, Xiaojun Wan, and William Yang Wang. Gödel agent: A self-referential agent framework for recursive self-improvement, 2024. URL <https://arxiv.org/abs/2410.04444>.
- [87] Sompote Youwai, David Phim, Vianne Gayl Murcia, and Rianne Clair Onas. Investigating the potential of large language model-based router multi-agent architectures for foundation design automation: A task classification and expert selection study, 2025. URL <https://arxiv.org/abs/2506.13811>.
- [88] Cheng Zhang, Jinwoo Kim, JungHo Jeon, Jinding Xing, Changbum Ahn, Pingbo Tang, and Hubo Cai. Toward integrated human-machine intelligence for civil engineering: An interdisciplinary perspective. *Computing in Civil Engineering 2021*, pages 279–286, 2021.

Contents

1	Introduction	2
2	Related Work	3
2.1	LLMs in Civil Engineering Applications	3
2.2	Multi-Agent Systems with LLMs	3
3	System Role Design	3
3.1	Traditional Workflow	4
3.2	Our System	4
3.3	Analyst Team	4
3.4	Engineer Team	5
3.5	Management Team	5
4	Agentic Workflow Details	5
4.1	Backbone Models	6
4.2	Communication Protocol	6
4.3	Agent Interactions	6
5	Experiments	7
5.1	Simulation Setup	7
5.2	Dataset	7
5.3	Evaluation Metrics	7
6	Results and Analysis	8
6.1	Main Results	8
6.1.1	Performance Comparison	8
6.1.2	Cost Analysis	8
6.1.3	Runtime Analysis	9
6.2	Human Evaluation	9
6.3	Ablation Study	10
7	Discussion	10
8	Conclusion	11
A	Single- vs Two-Agent Experiment	21
B	MASSE Workflow: Role Specification	21

C	MASSE Component Details	25
C.1	Analyst Team	25
C.2	Engineer Team	26
C.3	Management Team	26
D	Specific Case Trajectories	27
E	Experiment Details	31
E.1	Simulation Setup	31
E.2	Evaluation Metrics	31
E.2.1	Evaluation Judge Instructions and Benchmark Rubrics	32
F	Problem Example	33

Appendix

A Single- vs Two-Agent Experiment

Structural engineering workflow usually includes long-horizon tasks. Recent study proves that, without chain-of-thought, non-thinking models struggle to chain even two steps in a single turn [66]. An obvious solution is to split these steps across two models to execute sequentially. Therefore, we design an experiment to demonstrate that a multi-agent system is necessary for long-horizon tasks in structural engineering. This appendix reports a controlled comparison between two agent systems for generating OpenSeesPy code from natural language problem descriptions (PD): (1) a two-agent collaborative system (AgentA + AgentB) and (2) a single-agent end-to-end system (AgentAB). The primary goal of this experiment is to evaluate the execution success rate.

For the two-agent system, AgentB extracts parameters from PD into JSON and computes section properties, which are then passed to AgentA for generating complete OpenSeesPy scripts. In contrast, the single-agent system (AgentAB) directly converts PD into executable scripts in one step. To ensure fairness, both systems employ the same model (GPT-4o) under identical environments (temperature=0.0). The PD itself is formulated as a two-dimensional structural analysis task involving a two-step procedure. In the first step, material and geometric properties are extracted from the problem description, and a Python script is invoked to compute intermediate parameters such as the moment of inertia and cross-sectional area. In the second step, a finite element model is constructed using both the original parameters from the PD and the derived intermediate values. Outputs were standardized to executable Python scripts performing model assembly, load application, finite element analysis via `openseespy`, visualization via `opsvis`, and printing displacements, reactions, and internal forces.

For evaluation, we focus on execution success rates and systematically characterize the associated failure modes. The results clearly indicate that the two-agent system consistently outperformed the single-agent system in execution success rates. Specifically, the single-agent system failed in all ten trials (see Figure 2), primarily due to cascading errors in parsing and modeling logic. By contrast, the two-agent system achieved robust parameter extraction and stable execution across all cases, albeit with slightly higher latency. In summary, the findings highlight the importance of decomposition and specialization. The multi-agent system introduced modest overhead but ensured higher correctness and execution reliability.

B MASSE Workflow: Role Specification

This appendix outlines the organizational structure of agent roles within MASSE, detailing how specialized teams coordinate to address real-world structural engineering problems. The framework is anchored around three primary groups—the *Analyst Team*, the *Engineer Team*, and the *Management Team*—each responsible for distinct stages of the problem-solving pipeline.

The distribution of responsibilities across these roles draws inspiration from established practices in engineering consultancy, where complex projects are decomposed into well-defined tasks that are either managed individually by an engineer or collaboratively across teams. MASSE reflects this dual perspective: on the one hand, it mirrors the multi-tasking strategies of a single engineer navigating coupled subtasks, while on the other, it replicates the collaborative dynamics observed in professional firms. The system’s design principle is straightforward—once a large, long-horizon problem is partitioned into pre-

cise subtasks that can be verbalized with clarity, each subtask becomes tractable for a single LLM-based agent. This transformation allows the originally complex workflow to be executed seamlessly through a multi-agent system. The sections that follow present detailed specifications of these agent roles and demonstrate how their interaction yields transparent, resilient, and domain-grounded engineering outcomes.

Analyst Team Instruction Prompts

1. Loading Analyst

```
system_message = """
Extract building information from racking system description and return as JSON:

"location": "city, province/state",
"building_type": "racking_system",
"floor_elevations_ft": [list of elevations in feet],
"loads_lbs": [list of loads in pounds],
"dimensions":
  "width_ft": number,
  "height_ft": number,
  "beam_length_ft": number
,
"structural_info": "column and brace specifications"

Extract exact numerical values from the description.
"""
```

2. Seismic Analyst

```
system_message = "You extract seismic data from building codes. Return only JSON. No
explanations."
user_message = f"""Extract seismic parameters for location from the document below.
Document:
context
Find the exact numerical values for location and return ONLY this JSON:

"Sa_02": <number>,
"Sa_05": <number>,
"Sa_10": <number>,
"Sa_20": <number>,
"PGA": <number>,
"PGV": <number>

Rules:
- Return ONLY the JSON object
- Use actual numerical values from the document
- If location is not found, return: "error": "City not found"
- No explanations, no text outside JSON"""
```

3. Dynamic Analyst

```
system_message = """You are a Dynamic Analyst.
Get data from memory, then call
calculate_seismic_loads(floor_elevations_ft, loads_lbs, seismic_parameters)."""
```

4. Structural Analyst

```
system_message = """You are a StructuralAnalyst.

** CRITICAL EXECUTION ORDER **
STEP 1: Create load application nodes at EXACT required elevations FIRST - these are MANDATORY
and cannot be omitted
STEP 2: Create all brace connection nodes at their exact coordinates
```

STEP 3: Create column nodes for beam-column elements

STEP 4: Verify all required nodes exist before creating elements

Provide output strictly as JSON matching the following format (same keys, types, order). Return only the JSON.

CRITICAL INSTRUCTIONS:

1. Generate a structural model based on the geometry description provided.
2. ****ALL COORDINATES MUST BE IN FEET**** - Keep coordinates in feet, do NOT convert to inches.
3. Forces are in kip, stiffness is in kip/in².
4. ****LOAD NODES ARE MANDATORY**** - You MUST create load application nodes BEFORE any other nodes.
5. ****CREATE ALL BRACES EXACTLY AS SPECIFIED**** - You MUST create truss elements using the EXACT coordinates provided in the description.
6. Use sequential node IDs starting from 1.
7. AVOID duplicate nodes - merge nodes with same coordinates to prevent zero-length elements.
8. ****BRACE COORDINATES ARE MANDATORY**** - For each coordinate pair (x1,y1)->(x2,y2) in the description, create nodes at those EXACT coordinates and connect them with a truss element.
9. ****DO NOT MODIFY BRACE COORDINATES**** - Use the coordinates exactly as written, do not change them or infer different positions.
10. ****VERIFY****: Count your truss elements before finishing - must match the required count exactly.
11. ****UNITS****: Keep all coordinates in feet as provided in the description.
12. ****ELEMENT TYPE CONSISTENCY**** - Model braces strictly as 2-node axial-only `truss` elements; use beam/column elements (e.g., `elasticBeamColumn`) for framing members where bending is required, keeping section units consistent with stiffness units.
13. ****BOUNDARY CONDITIONS FIRST CLASS**** - Explicitly define base restraints (e.g., fix supports with (ux=uy=rz=0) for 2D) before applying any loads; all other DOFs remain free unless specified.

Required JSON format:

```
"units":
  "length": "ft (feet)",
  "force": "kip",
  "stiffness": "kip/in^2"
,
"materials":
  "E": 29000.0
,
"sections":
  "column": "A": 0.705, "I": 1.144 ,
  "brace": "A": 0.162
,
"nodes": [
  // Create nodes for all structural geometry points AND load application points
  // Example: "id": 1, "x": 0.0, "y": 0.0
],
"elements": [
  // Create beam-column and truss elements based on description
  // Include ALL column elements and ALL brace elements as specified
  // For braces: Use EXACT coordinates from description, do not modify or infer positions
  // Example: "id": 1, "type": "elasticBeamColumn", "nodes": [1, 2], "section": "column",
  "matTag": 1, "transfTag": 1
  // Example: "id": 2, "type": "truss", "nodes": [3, 4], "section": "brace", "matTag": 1
],
"supports": [
  // Fixed supports at base nodes
  // Example: "node": 1, "fixity": [1, 1, 1]
],
"loads": [
  // Will be populated automatically based on load application nodes
]
"""
```

Figure 8: Instruction prompts for the *Analyst Team*, including Loading, Seismic, Dynamic, and Structural Analysts.

Engineer Team Instruction Prompts**1. Design Engineer**

```
system_message = """You are a Design Engineer. Use run_complete_opensees_analysis() with structural model from memory. One function call completes entire analysis."""
```

2. Model Engineer

```
system_prompt_base = """You are a model engineer.
Use run_complete_opensees_analysis() with structural model from memory.
One function call completes entire analysis."""
```

3. Verification Engineer

```
system_message = """You are a VerificationEngineer. Use get_analysis_context() to get all data,
then verify_structural_safety(capacities, demands)."""
```

Figure 9: Instruction prompts for the *Engineer Team*: including Design, Model and Verification Engineers.

Management Team Instruction Prompts**1. Project Manager**

```
system_message = """
You are a structural engineer. Decompose the racking system problem into specific inputs.

Extract and return JSON with:

    "SDA_input": "Section design: [extract column and brace specifications]",
    "LA_input": "Loading analysis: [extract location, loads, heights, dimensions]",
    "SAA_input": "Structural analysis: [extract geometry, supports, elements - MUST preserve ALL
brace coordinates exactly as given]",
    "number_of_bays": [extract number],
    "number_of_pallets": [extract number per beam]

CRITICAL: For SAA_input, you MUST preserve ALL detailed brace coordinates exactly as they
appear in the original description. Do NOT simplify or summarize brace connections - copy them
verbatim with all coordinate pairs.

Focus on extracting exact numerical values and specifications from the description.
"""
```

2. Safety Manager

```
system_message = """You are a SafetyManager. Use get_analysis_context() to get all data.
Provide final assessment: "FINAL RESULT: STRUCTURALLY ADEQUATE" or "FINAL RESULT: STRUCTURALLY
INADEQUATE"."""
```

Figure 10: Instruction prompts for the *Management Team*: (1) Project Manager decomposes racking problems into structured inputs for design, loading, and analysis while preserving brace coordinates; (2) Safety Manager performs final adequacy check and outputs the ultimate structural decision.

C MASSE Component Details

This appendix provides detailed descriptions of all core components within the MASSE framework, expanding on the high-level summaries presented in the main text. Each subsection outlines the specific roles, methodologies, and implementation strategies of the corresponding teams and agents, ensuring reproducibility and technical clarity beyond the main paper's length constraints.

C.1 Analyst Team

The Analyst Team consists of specialized agents tasked with extracting and analyzing critical engineering parameters essential for structural analysis and design. Each agent addresses a distinct aspect of data processing, ensuring comprehensive and automated preparation necessary for subsequent structural analysis and design phases.

- **Loading Analyst:** Specializes in interpreting detailed structural descriptions, converting them into structured engineering data. Leveraging natural language processing and general-purpose LLMs, the Loading Analyst systematically extracts essential building attributes such as floor elevations, applied loads, geometric dimensions, material specifications, and specific engineering criteria. The extracted information is structured into JSON schemas with standardized units, facilitating efficient downstream analyses.
- **Seismic Analyst:** Dedicated to retrieving and compiling seismic design parameters from authoritative databases and regulatory documents. Utilizing Retrieval-Augmented Generation (RAG) alongside advanced semantic search and vector-based retrieval methods such as FAISS indexing, the Seismic Analyst efficiently locates and extracts key seismic data, including spectral acceleration values, peak ground acceleration, and peak ground velocity. The agent employs PDF parsing, embedding models for semantic accuracy, and structured data storage, ensuring precise geographic-specific retrieval and rapid information availability.
- **Dynamic Analyst:** Responsible for calculating seismic loads by accurately integrating structural characteristics with seismic parameters. This agent employs established structural dynamics methods, including the equivalent lateral force procedure in line with current regulatory codes. The Dynamic Analyst systematically computes base shear forces, lateral load distributions, height-dependent amplification factors, and detailed seismic demands at the story level. Computations are rigorously validated using numerical methods provided by mathematical libraries such as NumPy, ensuring adherence to regulatory standards.
- **Structural Analyst:** Acts as an advanced structural model generation specialist. Utilizing AutoGen's ConversableAgent framework integrated with general-purpose LLMs, this agent transforms high-level engineering descriptions into detailed, JSON-formatted finite element models compatible with OpenSeesPy. Its primary function, `generate_structural_model(description)`, processes input from `StructuralMemoryManager` using prompts, validating critical load nodes for accurate force applications at specified elevations. The agent produces comprehensive structural models detailing geometric coordinates, element connectivity, material properties, section properties for structural elements (columns and braces), boundary conditions, and load vectors, integrating calculated seismic loads. Robust error handling includes JSON parsing and regular expression-based content cleaning to ensure uninterrupted analysis. The Structural Analyst ensures structural engineering com-

pliance, computational efficiency, and effective cross-agent communication through memory management tools.

Collectively, the Analyst Team synthesizes structured engineering data from various sources, including detailed textual descriptions and regulatory documentation. Their coordinated efforts provide comprehensive, reliable inputs to the Engineer Team, creating a solid foundation for precise, informed structural engineering decisions.

C.2 Engineer Team

The Engineer Team systematically conducts engineering evaluations informed by data provided by the Analyst Team. Consisting of specialized agents that focus on structural analysis, structural design, and structural adequacy verification, the team performs iterative analyses through tool utilization and structured data exchanges, evaluating structural integrity and identifying potential risks.

- **Design Engineer:** This agent calculates the structural capacities of individual elements such as beams, columns, and braces, a critical step for ensuring the structural safety and efficacy of proposed designs. Equipped with predefined Python scripts, the agent returns structural capacities and associated section properties to the system memory. These results underpin the selection of optimal yet safe combinations of structural members in comprehensive structural designs.
- **Model Engineer:** This agent synthesizes structured data from the Analyst Team, specifically load information from the Dynamic Analyst and structural model parameters from the Structural Analyst, to execute finite element analyses. Utilizing `OpenSeesPy` and `Opsvis` packages, this agent autonomously generates and runs Python-based finite element analysis, subsequently visualizing results like internal force distributions and deformation shapes for user review. It also engages dedicated Python scripts for automated post-processing, adeptly handling diverse load scenarios beyond seismic loads, such as live loads, thus ensuring automated structural analysis workflows.
- **Verification Engineer:** This agent evaluates structured results provided by the Design Engineer and Model Engineer, deploying Python scripts to systematically verify all critical structural behaviors including tension, compression, bending moments, torsion, deflection, and rotation. Should any structural element fail to meet established performance criteria, the Verification Engineer explicitly identifies the deficiency (e.g., inadequate beam strength at a specific floor), clearly documenting the structural inadequacies and underlying reasons.

Through this structured, iterative evaluation process, the Engineer Team comprehensively captures essential structural and model data. Their meticulous analysis facilitates precise identification of structural responses under specific load conditions and the structural capacity of each element, thereby providing robust support for informed decision-making by the Management Team.

C.3 Management Team

The **Management Team** oversees the operation and coordination of the entire multi-agent system, executing definitive engineering decisions guided by detailed analyses from the Analyst Team and the specialized insights provided by the Engineer Team. They critically evaluate the integrated findings, examining both quantitative outputs and intermediate analytical processes, to make authoritative and conclusive engineering judgments.

The primary responsibilities of the MASSE Management Team include:

- Coordinating the structural analysis workflow through problem decomposition and task distribution.
- Managing data integration and intermediate results processing, ensuring project timelines and quality standards are maintained.
- Conducting final structural safety evaluations and authoritative decision-making based on comprehensive analytical outcomes.
- Providing standardized safety conclusions and managing system termination protocols.

The Management Team imitates the roles of managers and administrative coordinators in consulting engineering firms, illustrating how essential managerial and coordination functions can be systematically modeled within agent-based workflows.

D Specific Case Trajectories

We further present a detailed trace of a representative case trajectory, which illustrates how the multi-agent system progresses through each step of problem-solving. Such traces not only enable us to analyze and debug the system's behavior but also provide valuable insights into potential avenues for optimization and the range of problems the framework can address. By documenting the complete trajectory, the outcomes of the system become highly interpretable. Moreover, presenting key intermediate steps for verification by human engineers significantly enhances the trustworthiness of the system in real-world deployment. An example case trajectory is shown below.

MASSE Analysis Log — Initialization
<p>Step 0: System Initialization</p> <p>Log Start: ===== MASSE ANALYSIS LOG START =====</p> <p>Logger: Initialized — all terminal output will be saved to log file.</p> <p>Database: - Loading existing vector database from [REDACTED PATH] - RAG seismic database validation successful</p> <p>Agent Registration: 15 functions successfully registered to agents and user_proxy.</p> <p>Problem Setup: Location: Nanaimo, BC System: 2 bays, 8.0 ft beam length, 3 pallets per beam. Beam: 4 in Z-section. Columns: Steel U-channels (3.079 in × 2.795 in × 0.0787 in, 16.0 ft). Braces: Steel U-channels (1.0 in × 1.0 in × 0.054 in, length 4.3 ft). Geometry: 2 beam-columns from (0,0)→(0,16.0) and (3.5,0)→(3.5,16.0). Bracing: 8 pin-ended trusses connecting specified nodes. Supports: Fixed at (0,0) and (3.5,0). Loads: Applied to left column at 4.0 ft, 8.5 ft, 13.0 ft. Units: Coordinates in ft, forces in kip, stiffness in kip/in².</p> <p>Runtime Config: - AutoGen version: 0.9.2 - LLM Config: gpt-4o, Temp = 0, Max tokens = 2000 - Runtime logging started (UUID [REDACTED]) - Database path [REDACTED PATH]</p> <p>Agents Initialized: ProjectManager, DesignEngineer, LoadingAnalyst, SeismicAnalyst, DynamicAnalyst, ModelEngineer, StructuralAnalyst, VerificationEngineer, SafetyManager</p>

System Verification: Completed — all components verified.

Log End: System Initialization finished, structural analysis can proceed.

ProjectManager — Step 1: Problem Decomposition and Setup

Step 1: Problem Decomposition and Setup

User Instruction: - Call `split_problem_description()` with the full problem text. - After splitting, call `adjust_pallet_weights()` using `number_of_bays` and `number_of_pallets` from memory.

Tool Call 1: `split_problem_description` *Input:* Problem description text. *Output:* Problem split into SAA, SDA, LA inputs. Bays = 2, Pallets = 3.

Tool Call 2: `adjust_pallet_weights` *Input:* LA_input, num_bays = 2, num_pallets = 3. *Process:* - Loads before update: None - Loads after update: [1875, 1125, 750] (lbs) *Output:* Pallet weights adjusted successfully.

Result Summary: - Problem description split into SAA, SDA, LA inputs. - Bays = 2, Pallets = 3. - Pallet weights updated to 1875, 1125, 750 lbs.

Follow-up Tool Calls: - `update_saa_input(section_data, load_data)` → SAA input updated successfully, brace coordinates preserved. - `save_analysis_results(filepath="analysis_results.json")` → Results saved.

DesignEngineer — Step 2: Section Design

Step 2: Section Design

User Instruction: Extract section info from SDA_input and calculate capacities.

Tool Call 1: `extract_section_info` *Input:* Description text from SDA_input. *Output:* Section info extracted — column, brace, beam.

Tool Call 2: `calculate_section_capacities` *Input:* Section info (from previous step). *Output:* Section capacities calculated successfully.

Result Summary: - Section info extracted from SDA_input. - Element capacities computed and stored.

LoadingAnalyst — Step 3: Building Information

Step 3: Building Information

User Instruction: Extract building info from LA_input.

Tool Call: `extract_building_info` *Input:* Description text from LA_input. *Output:* Building info extracted — racking system, 3 floors, 3 load points.

Result Summary: - Building type: Racking system - Number of floors: 3 - Number of load points: 3

SeismicAnalyst — Step 4: Seismic Parameters

Step 4: Seismic Parameters

User Instruction: Get seismic parameters for Nanaimo, BC based on location from building_info.

Tool Call: `get_seismic_parameters` *Input:* Location = "Nanaimo, BC". *Process:* Retrieved data using RAG search; 6 relevant document segments found. *Output:* Seismic parameters successfully extracted.

Result Summary: - Location: Nanaimo, BC - $Sa_{0.2} = 1.02$ - $PGA = 0.446$

DynamicAnalyst — Step 5: Load Calculations

Step 5: Load Calculations

User Instruction: Calculate seismic loads using data from memory.

Tool Call 1: `get_memory_summary` *Output:* Memory contains 14 non-null entries including SAA_input, SDA_input, LA_input, building_info, seismic_parameters, floor_elevations_ft, loads_lbs, section_data.

Tool Call 2: `get_memory_data("floor_elevations_ft")` *Output:* [4.0, 8.5, 13.0]

Tool Call 3: `get_memory_data("loads_lbs")` *Output:* [1875, 1125, 750]

Tool Call 4: `get_memory_data("seismic_parameters")` *Output:* {Sa_02: 1.02, Sa_05: 0.942, Sa_10: 0.037, Sa_20: 0.328, PGA: 0.446, PGV: 0.684}

Tool Call 5: `calculate_seismic_loads` *Input:* Floor elevations = [4.0, 8.5, 13.0], Loads = [1875, 1125, 750] *Output:* Seismic loads calculated: $F1 = 0.395$ kip, $F2 = 0.504$ kip, $F3 = 0.514$ kip

Result Summary: - Floor elevations and pallet weights retrieved. - Seismic parameters obtained from memory. - Final seismic loads: $F1 = 0.395$ kip, $F2 = 0.504$ kip, $F3 = 0.514$ kip.

ProjectManager — Step 6: Update SAA Input

Step 6: Update SAA Input

User Instruction: Call `update_saa_input()` to merge SAA, section, and load data from memory.

Tool Call 1: `get_memory_data("saa_input")` *Output:* ...

Tool Call 2: `get_memory_data("section_data")` *Output:* JSON containing section properties and capacities for posts, braces, and beams.

Tool Call 3: `get_memory_data("load_data")` *Output:* JSON containing load cases: - Seismic: $F1 = 0.395$, $F2 = 0.504$, $F3 = 0.514$ - Live: $F1 = 1.875$, $F2 = 1.125$, $F3 = 0.75$

Tool Call 4: `update_saa_input(section_data, load_data)` *Process:* - Original SAA_input length: 365 chars - Brace coordinates preserved - Updated SAA_input length: 365 chars *Output:* SAA input updated successfully with section and load data.

Result Summary: - Section and load data retrieved from memory. - SAA input merged successfully. - Structural brace coordinates preserved.

StructuralAnalyst — Step 7: Generate Structural Model

Step 7: Generate Structural Model

User Instruction: Call `generate_structural_model()` using updated SAA input.

Tool Call 1: `get_memory_data("SAA_input_update")` *Output:* Structural analysis description including: - Two elastic beam-columns from (0,0)→(0,16.0) and (3.5,0)→(3.5,16.0) - Fixed supports at (0,0) and (3.5,0) - 8 pin-ended truss braces with specified coordinates

Tool Call 2: `generate_structural_model` *Process:* - Found 8 brace coordinates in description - Floor elevations retrieved from memory: [4.0, 8.5, 13.0] - Required load nodes: (0,4.0), (0,8.5), (0,13.0) - Generated 15 nodes total - Generated 8 truss elements + 2 beam-column elements - Model coordinate ranges: $X = 0-3.5$, $Y = 0-16.0$ - Applied loads: $F1 = 0.395$ kip, $F2 = 0.504$ kip, $F3 = 0.514$ kip *Output:* Structural model generated with 15 nodes and 10 elements.

Result Summary: - Structural model created from SAA input. - Geometry, supports, and braces defined correctly. - Seismic loads applied to correct nodes. - Final model: 15 nodes, 10 elements.

ModelEngineer — Step 8: OpenSees Analysis

Step 8: Complete Structural Analysis (OpenSees)

User Instruction: Run `run_complete_opensees_analysis()` using the structural model.

Tool Call 1: `get_memory_summary` *Output:* Memory contains 16 non-null entries, including `structural_model`, `load_data`, and `section_data`.

Tool Call 2: `get_memory_data("structural_model")` *Output:* JSON model with: - Units: ft, kip, kip/in² - Materials: $E = 29000$ - Sections: column ($A = 0.705$, $I = 1.144$), brace ($A = 0.162$) - 15 nodes, 10 elements (2 `elasticBeamColumn`, 8 truss) - Supports: fixed at base nodes - Loads: seismic forces at nodes (0,4.0), (0,8.5), (0,13.0)

Tool Call 3: `run_complete_opensees_analysis(model_data)` *Process:* - Generated OpenSees script `structural_model_exec.py` - Analysis executed successfully - Internal forces saved to `internal_forces.json` - Applied load combinations: seismic = 3.340, live = 5.625, envelope = 5.625 *Output:* Internal forces extracted and stored.

Result Summary: - OpenSees analysis completed successfully. - Beams: Max tension = 3.34 kip, Max compression = 5.62 kip, Max moment = 7.72 kip-in - Trusses: Max tension = 1.35 kip, Max compression = 1.73 kip - Results saved in memory.

VerificationEngineer — Step 9: Structural Verification

Step 9: Structural Verification

User Instruction: Verify structural safety using all analysis data.

Tool Call 1: `get_analysis_context` *Output:* Context includes: - Seismic parameters ($Sa_{02} = 1.02$, $Sa_{05} = 0.942$, $Sa_{10} = 0.037$, $Sa_{20} = 0.328$, $PGA = 0.446$, $PGV = 0.684$) - Structural model JSON (geometry, sections, loads) - Processed forces: • Beams — max tension = 3.34 kip, max compression = 5.625 kip, max moment = 7.72 kip-in • Trusses — max tension = 1.35 kip, max compression = 1.73 kip - Adjusted SAA, SDA, and LA inputs

Tool Call 2: `verify_structural_safety` *Input:* - Capacities: beams and trusses from section design - Demands: seismic loads from nodes *Output:* Safety verification complete — PASS

Result Summary: - Verification executed using capacities and demands. - All elements satisfy safety requirements. - Structure passes safety verification.

SafetyManager — Step 10: Final Safety Assessment

Step 10: Final Safety Assessment

User Instruction: Use `get_analysis_context()` to obtain all data and provide the final assessment: "FINAL RESULT: STRUCTURALLY ADEQUATE" or "FINAL RESULT: STRUCTURALLY INADEQUATE".

Tool Call 1: `get_analysis_context` *Output:* Context includes: - Seismic parameters ($Sa_{02} = 1.02$, $Sa_{05} = 0.942$, $Sa_{10} = 0.037$, $Sa_{20} = 0.328$, $PGA = 0.446$, $PGV = 0.684$) - Structural model JSON with geometry, materials, sections, and loads - Processed forces: • Beams — max tension = 3.34 kip, max compression = 5.62 kip, max moment = 7.72 kip-in • Trusses — max tension = 1.35 kip, max compression = 1.73 kip - Safety evaluation: PASS • Post checks — tension ratio = 0.13, compression ratio = 0.28 • Brace checks — tension ratio = 0.18, compression ratio = 0.34

Final Assessment: FINAL RESULT: STRUCTURALLY ADEQUATE

Result Summary: - Full system context retrieved. - SafetyManager confirmed PASS. - Structure deemed structurally adequate. - MASSE analysis log closed successfully.

E Experiment Details

E.1 Simulation Setup

We selected a racking system design [13] to exemplify typical structural engineering tasks. Racking systems, extensively utilized in warehouses, optimize vertical cargo storage while ensuring structural safety. Structural engineers regularly determine allowable loads and issue certified reports to ensure operational compliance. This scenario effectively integrates core structural engineering tasks—structural analysis, design, and load transformation—aligned directly with the functionality of our proposed agents. Figure 12 illustrates the system configuration. The racking elevation shows a three-tier arrangement with specific load capacities of 1000 lbs (P1), 1250 lbs (P2), and 1750 lbs (P3), while the racking side elevation highlights diagonal bracing employed to achieve lateral stability.

We developed a comprehensive dataset consisting of one hundred distinct scenarios validated through expert-derived ground-truth solutions, facilitating robust evaluation reflective of real-world engineering challenges. Although unique in nature, particularly because loading reports for racking systems are mandated and must be carefully certified by structural engineers specializing in earthquake-prone regions such as British Columbia and California, the racking system scenario is representative of broader structural engineering tasks. The various components of this scenario—such as parameter retrieval, load transformation calculation, structural modeling, structural analysis, structural design, section property determination, section capacity calculation, and structural adequacy verification—are fundamental to structural engineering practice. Therefore, while the agents developed in this framework specifically address the racking system scenario, their design enables easy adaptation to other structural engineering contexts. With minimal modifications, these agents can be employed in alternative multi-agent systems, making them effectively plug-and-play and broadly applicable within the structural engineering domain.

E.2 Evaluation Metrics

To accurately simulate the environment of a structural engineering consulting firm, we utilize a dataset derived from real-world racking system projects located in British Columbia, Canada. Our comprehensive dataset enables benchmarking across distinct system components, including agents dedicated to structural analysis, agents focused on structural design, agents responsible for load transformation, and the agents within the whole system that execute the complete structural engineering task. Each component involves multiple agents collaboratively addressing distinct tasks. The dataset encompasses:

Structural Analysis Agent Benchmark (SAAB): Evaluates the capability of the structural analysis agents to accurately construct finite element models—including geometry, supports, and load conditions—using OpenSeesPy, based solely on natural language input and internal memory states. The assessment further examines the successful execution of finite element analyses and the accurate retrieval of specified analysis results from OpenSeesPy outputs.

Structural Design Agent Benchmark (SDAB): Assesses structural design agents’ abilities to accurately interpret dimensional information from natural language inputs, invoke appropriate computational tools to determine necessary properties and capacities, store results correctly, and transfer them effectively into memory states.

Loading Agent Benchmark (LAB): Evaluates the proficiency of loading agents in accurately extracting relevant information from natural language inputs, effectively performing Retrieval-Augmented Generation (RAG) operations on supporting documents, and invoking appropriate tools to correctly determine the required applied loads.

Multi-Agent Structural Engineering Benchmark (MASEB):

Provides a comprehensive scoring framework designed to objectively assess system performance based on different LLMs.

E.2.1 Evaluation Judge Instructions and Benchmark Rubrics

We employ GPT-5 as a LLM Judge to evaluate MASSE’s outputs against expert-verified ground-truth solutions. The system instruction is defined as follows.

System Instruction

System Instruction: MASSE Evaluation Judge (Refined Scheme)

You are an LLM Judge. Your role is to evaluate the performance of the MASSE multi-agent system by analyzing one complete analysis log. You must assign 0–100 points for each benchmark: Structural Analysis Agent Benchmark (SAAB), Structural Design Agent Benchmark (SDAB), Loading Agent Benchmark (LAB), and Multi-Agent Structural Engineering Benchmark (MASEB). Scoring must follow the detailed rubrics below. Your final output must be only a JSON object with four scores, plus total token usage and total time.

The four benchmarks are defined as follows. Structural Analysis Agent Benchmark (SAAB) consists of four components: Model Geometry Accuracy (30 pts), Integration of Section and Load Data (20 pts), OpenSees Analysis Execution (30 pts), and Result Retrieval Accuracy (20 pts). Structural Design Agent Benchmark (SDAB) is divided into Extraction Accuracy (30 pts), Capacity Computation (30 pts), Data Storage and Memory Update (20 pts), and Transfer and Availability (20 pts). Loading Agent Benchmark (LAB) covers Load Extraction (25 pts), Adjustment and Normalization (25 pts), RAG Seismic Retrieval (25 pts), and Load Calculation (25 pts). Finally, the Multi-Agent Structural Engineering Benchmark (MASEB) includes Pipeline Completion (30 pts), Consistency Across Agents (30 pts), Final Result Accuracy (20 pts), and Efficiency and Robustness (20 pts).

Figure 11: LLM Judge System Instruction

F Problem Example

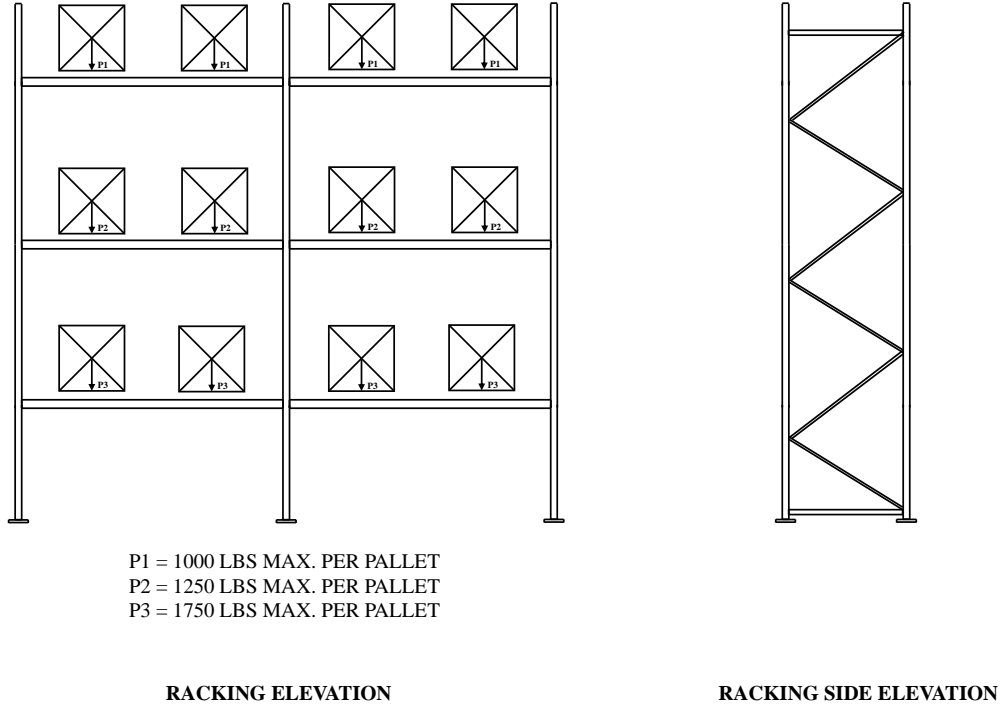


Figure 12: Racking system example.

Problem Description (for structural scheme see Figure 12)

A steel racking system frame located in Nanaimo, BC is modeled in side elevation as a 2D frame with elastic members and pin-ended truss braces. Coordinates are given in feet (1 ft = 12 in); forces are in kip; stiffness is in kip/in². 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 beam-columns with $E = 29,000$ kip/in², and the braces are steel U-channels 1.0 in × 1.0 in × 0.054 in, treated as truss elements (pin-pin) 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) → (3.5,0.5), (3.5,0.5) → (0,3), (0,3) → (3.5,5.5), (3.5,5.5) → (0,8), (0,8) → (3.5,10.5), (3.5,10.5) → (0,13), (0,13) → (3.5,15.5), and (3.5,15.5) → (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, is the structural system safe in this scenario?

Figure 13: A racking system problem description