POLITECNICO DI TORINO

Master's degree Program in
in Digital Skills for Sustainable Societal Transitions

Master's Degree Thesis

Human-in-the-Loop Decision Support and Visualization for LLM-Assisted Structural Simulations

Supervisor:

prof. Rosario Ceravolo

Candidate:

Amir Yarmohamadi

Co-Supervisors:

prof. Gianvito Urgese

prof. Gaetano Miraglia

A.A. 2024/2025

Acknowledgements

I am deeply grateful to my family for their unwavering support, patience, and belief in me throughout my academic journey. I am especially thankful for the way they endured my distance and absence with understanding and love. Their encouragement has always been a source of strength and motivation.

A special thank you goes to my wife, Saeideh, for her love, understanding, and constant support during the many challenges and long hours devoted to this research. I would also like to thank my daughter, Liana, whose playtime I too often had to shorten while working on this thesis. Their patience, love, and encouragement made this achievement possible.

Finally, I am proud and grateful to have reached this goal after a long journey of study, work, and perseverance. As I conclude this chapter, I express my hope for a better world, one where peace, knowledge, and sustainable development are shared by all nations. I also carry a sincere hope that my homeland, Iran, and its people will enjoy an increasingly bright future filled with opportunity, prosperity, and hope.

Thank you all for being part of this journey.

ABSTRACT

Finite element analysis is widely used in structural engineering, but many simulation workflows still depend on manual preparation, specialized expertise, and time-consuming result interpretation. Recent advances in large language models (LLMs) suggest that conversational systems can support engineering workflows, although their use in simulation environments raises important concerns regarding reliability, controllability, and engineer oversight.

This thesis presents a human-in-the-loop conversational framework that integrates a large language model agent with the Code_Aster finite element solver through a Python-based orchestration system. The framework allows engineers to describe structural beam analysis problems in natural language while the agent collects missing parameters, validates inputs, and confirms all simulation settings before execution. The LLM is used only for reasoning, coordination, and interaction management, whereas all numerical calculations are performed by deterministic engineering software. This separation between language reasoning and numerical computation is a central design principle of the proposed system.

The framework maintains engineer control by requiring explicit confirmation before simulation execution and by regulating how computational tools are invoked during interaction. After each analysis, the system automatically generates engineering interpretation and verification outputs, including displacement, rotation, shear force, bending moment, and equilibrium diagrams, presented through an interactive interface for result inspection and comparison. Evaluation across five structured case studies showed that the framework consistently enforced human-in-the-loop requirements, correctly rejected unsupported requests, and produced results in close agreement with analytical and OpenSees reference solutions.

Although the current implementation is restricted to the analysis of simply supported subjected to a central point load and cantilever beams subjected to a tip load, this study demonstrates that large language model-based orchestration can effectively support engineering simulation workflows while preserving transparency, reliability, and engineer oversight.

Keywords

large language model, finite element analysis, human-in-the-loop, Code_Aster, structural simulation, decision support, visualization, LangChain, beam analysis, engineering automation

Table of Contents

ABSTRACT ***7***

Chapter 1 – Introduction ***9***

1.1 Background 9

1.2 Problem Statement 12

1.3 Objectives 13

1.4 Scope of the Work 14

1.5 Thesis Structure 15

Chapter 2 – Literature Review ***16***

2.1 FEM Workflows and Challenges 16

2.2 Visualization in Structural Analysis 17

2.3 Large Language Models in Engineering Applications 18

2.4 Human-in-the-Loop AI 19

2.5 Decision Support Systems 21

2.6 Research Gaps and Motivation 22

Chapter 3 – System Architecture ***24***

3.1 Overview of the Proposed System 24

Layer 1. User interface layer 24

Layer 2. LLM agent layer 25

Layer 3. Tool calling 25

Layer 4. Workspace and file layer 25

Design principles 26

3.2 System Workflow 27

3.2.1 Workflow phases 27

3.2.2 The execution sequence within Phase 4 28

3.2.3 Safe_call() and tool calling integrity 29

3.2.4 Unit conversion at the phase boundary 29

Chapter 4 – Human-in-the-Loop Decision Support System ***31***

4.1 Overview of the Proposed System 31

Interface layer 31
Agent layer 31
Behavioral rules as design decisions 32
4.2 Interaction Design 33
4.2.1 The conversational paradigm 33
4.2.2 Session modes 34
4.2.3 Memory and state across turns 34
4.2.4 Two directions of information flow 34
4.2.5 Unit convention as interaction design 35
4.2.6 Stop and redirect mechanisms 36
4.3 Input Handling & Validation 36
4.3.1 The required parameter set 36
4.3.1 The required parameter set 36
4.3.2 Progressive collection 37
4.3.3 Engineer-initiated consultation for material and section properties 37
4.3.4 Deterministic validation 38
4.3.5 The pre-simulation confirmation gate 39
4.4 AI Interpretation and Decision Support 39
4.4.1 Displacement results 40
4.4.2 Stress results 40
4.4.3 Reaction forces and equilibrium verification 40
4.4.4 Bending moment and internal force distribution 41
4.4.5 Explanation structure and framing 41
4.4.6 Limitations of AI interpretation 41
4.5 Hybrid Approach (Rule-based LLM) 42
4.5.1 The rule-based layer 42
4.5.2 The LLM layer 43
4.5.3 Division of responsibility 43
4.6 Explanation Generation 44
4.6.1 Connection to verification 44
4.6.2 Limitations and the role of engineer judgment 44
4.7 Human–AI Interaction Loop 44
Phase 1. Information Gathering 45
Phase 2. Consultation 45
Phase 3. Confirmation 45

Phase 4. Execution 46
Phase 5. Validation 46
Phase 6. Interpretation 46
4.8 System Workflow Example 48
Scenario 49
Commentary 50
Chapter 5 – Visualization Framework 52
5.1 Role of Visualization in Decision Support 52
5.2 Implemented Visualization Tools 54
5.2.1 Tool Architecture and Integration 54
5.2.2 Deflection and Rotation visualizer (visualize_resu) 55
5.2.3 Shear Force and Bending Moment (visualize_stress) 56
5.2.4 Moment Contour on Beam Schematic (visualize_mfz_contour) 58
5.2.5 Free-Body Diagram (visualize_freebody) 59
5.2.6 Unit Conversion Layer (units.py) 60
5.3 Result Types and Engineering Interpretation 61
5.3.1 Deflection (D_Y) 61
5.3.2 Rotation (D_{RZ}) 62
5.3.3 Shear Force (V_Y) 62
5.3.4 Bending Moment (M_{FZ}) 63
5.3.5 Moment Contour 63
5.3.6 Reactions and Free-Body Diagram 63
5.4 Frontend Delivery 64
5.4.1 Architecture and Design Rationale 64
5.4.2 Results Tab 65
5.4.3 Artifacts Tab 67
5.4.4 Verification Tab 68
5.4.5 Compare Tab 69
5.4.6 Run Storage Model and Artifact Tracking 70
Chapter 6 – Case Studies & Evaluation 73
6.1 Evaluation Framework 73
6.1.1 Metric definitions 73
6.2 Experimental Setup 75
6.2.1 Software configuration 75

6.2.2 Baseline structural configuration 75
6.2.3 Analytical reference solutions 76
6.2.4 Session protocol 76
6.3 Case Studies 77
6.3.1 Case 1: Baseline Correct Input 77
6.3.2 Case 2: Missing Section 79
6.3.3 Case 3: Ambiguous Unit Input 81
6.3.4 Case 4: Out-of-Scope Boundary Condition 83
6.3.5 Case 5: Iterative Re-Analysis and Comparative Evaluation 84
6.4 Cross-Case Analysis 87
6.4.1 HITL compliance summary 87
6.4.2 Numerical accuracy summary 88
6.4.3 Interaction efficiency 88
Chapter 7 – Discussion & Conclusion 90
7.1 contributions of this Thesis 90
7.2 Limitations 92
7.3 Future Work 93
7.4 Final Remarks 93
References 95

List of Figures

Figure 3.1 System architecture and functional layers..... 26

Figure 3.2 System workflow from session start to result interpretation..... 30

Figure 4.1 Three-layer system architecture and information flow..... 36

Figure 4.2 Five-phase interaction loop..... 48

Figure 5.1 Deflection profile generated by visualize_resu..... 56

Figure 5.2 Rotation profile generated by visualize_resu..... 56

Figure 5.3 Shear force diagram generated by visualize_stress..... 57

Figure 5.4 Bending moment diagram generated by visualize_stress..... 57

Figure 5.5 MFZ contour generated by visualize_mfz_contour..... 58

Figure 5.6 Free-body diagram generated by visualize_freebody..... 60

Figure 5.7 Streamlit frontend overview..... 65

Figure 5.8 Results tab of the frontend..... 66

Figure 5.9 Artifacts tab of the frontend..... 67

Figure 5.10 Verification tab showing result comparison metrics..... 69

Figure 5.11 Compare tab displaying deflection profile comparison..... 70

List of Tables

Table 4.1 Required simulation parameters and default values..... 32

Table 4.2 Responsibilities of the LLM and rule-based layers..... 43

Table 4.3 Example traced session of the interaction loop..... 49

Table 5.1 Visualization tool registry..... 54

Table 5.2 Unit conversion rules for result visualization..... 60

Table 6.1 Evaluation metrics and case study applicability..... 74

Table 6.2 Software configuration for case studies..... 75

Table 6.3 Baseline structural configuration..... 75

Table 6.4 Case 1 interaction trace..... 77

Table 6.5 Case 1 numerical results..... 78

Table 6.6 Case 1 HITL compliance assessment..... 78

Table 6.7 Case 2 interaction trace..... 80

Table 6.8 Case 2 numerical results..... 80

Table 6.9 Case 2 HITL compliance assessment..... 80

Table 6.10 Case 3 interaction trace..... 82

Table 6.11 Case 3 numerical results..... 82

Table 6.12 Case 3 HITL compliance assessment..... 82

Table 6.13 Case 4 interaction trace..... 84

Table 6.14 Case 4 HITL compliance assessment..... 84

Table 6.15 Case 5 interaction trace..... 86

Table 6.16 Case 5 comparative results..... 86

Table 6.17 Case 5 workflow consistency assessment..... 87

Table 6.18 HITL compliance across all case studies..... 87

Table 6.19 Numerical accuracy across Cases 1–3..... 88

Table 6.20 Interaction efficiency across case studies..... 88

CHAPTER 1 – INTRODUCTION

Structural analysis software has reached a high level of maturity. Solvers can model complicated geometries under arbitrary loading conditions and deliver accurate results. Yet in practice, using this software is still slow, manual work. The engineer must prepare input files by hand, manage several tools in sequence, and then interpret raw numerical output once the solver finishes. For a trained specialist, this is routine. For a student, a junior engineer, or anyone working outside their primary area of expertise, the process is a barrier.

This thesis describes an attempt to reduce that barrier. The system built here connects a Large Language Model (LLM) to the Code_Aster finite element solver through a Python-based framework. allowing an engineer to describe a beam analysis problem in plain language and receive interpreted simulation results, figures, numerical summaries, and engineering commentary. This process happens without writing a single input file manually. The purpose of the system is remaining the engineer in control at every step. no simulation runs without confirmation, no material value is used without approval, and the session can be stopped at any moment.

This thesis was conducted in collaboration with the S.T.A. DATA Srl. and Politecnico di Torino. The work is a research with presenting a prototype, not a production tool. It is designed to explore a specific question which is can an LLM usefully coordinate a deterministic FEM workflow while continuing human control at the critical decision points? The five case studies in Chapter 6 represent an initial, transparent attempt to answer that question.

1.1 Background

Finite element analysis in structural engineering

The finite element method has been at the core of structural engineering computation for over sixty years. Liu, Li, and Park (2022) trace its history from the mathematical basis established in the early 1940s through to its use across almost every engineering domain today, describing four main stages of development [1]. The basic idea is to break a continuous structure into smaller pieces. Elements whose behavior can be described mathematically. Those descriptions are then assembled into a large system of equations and solved numerically. Problems that used to be analytically impractical became tractable once computers were fast enough to handle these systems.

Code_Aster is one of the open-source solvers that came out of this development. It has been in development at EDF, the French energy company, since 1989, and was made publicly available under an open license in 2001. It is used in French civil and nuclear engineering practice as well as in European academic research. It covers a wide range of analysis types, from static, dynamic, thermal, to nonlinear. But in this thesis, only linear static structural analysis is used, which is the standard starting point for beam design.

Zienkiewicz, Taylor, and Zhu (2013) wrote what is probably the most well-known textbook on FEM theory and practice. Published by Elsevier, it covers everything from the variation principles behind the method to pragmatic application for solid, structural, and fluid problems [2]. Alongside Liu et al. (2022), this work makes it clear that FEM itself is mature and numerically reliable. The difficulties engineers face in practice are not about the math, but rather the workflow around it.

Running a simulation in Code_Aster requires several steps. Defining the geometry, generating a mesh, writing a command file that specifies the material, the cross-section, the boundary conditions, the loading, and the output format. All in Code_Aster's own syntax and then setting up a job configuration file before anything actually runs. Each of these steps needs a different kind of knowledge, and that knowledge is separate from structural engineering competence. Tapeh and Naser (2023) reviewed over 4,000 publications on AI and machine learning in structural engineering and noted that the overhead of model preparation, not the computation itself, is consistently reported as one of the main things that slows down the use of FEM, especially for less experienced users [3].

Errors in model preparation are also hard to catch. A node group with a wrong name, an inconsistent unit, or a boundary condition that does not match the physical setup usually does not cause an obvious error. The solver just produces a result that looks plausible but is physically wrong. Recent work on automated structural analysis recognizes this kind of silent failure as the most dangerous part of working with complex FEM workflows when the user is not deeply familiar with the solver [4].

Large language models and engineering tasks

Large language models are a type of neural network trained using large amounts of text. They can generate fluent, context-aware language and have shown strong performance on tasks including reasoning, code generation, and answering domain-specific questions. OpenAI's GPT-4, the model used in this thesis, was reported to achieve human-level scores on several

professional benchmarks including performance near the top of a simulated bar exam [5]. While it was not designed for engineering specifically, it handles multi-step technical reasoning well enough to be practically useful in this context.

In this thesis, the LLM is not used as a solver. It is used as a coordinator. It reads what the engineer writes, figures out what information is still missing, asks for it, looks up material properties when needed, manages the confirmation process, and explains the results once the simulation is done. All the actual computations, including geometry generation, meshing, the finite element assembly, solving, and post-processing, are done by deterministic code. The LLM does not calculate anything.

This separation between the language model and the computation is not purely a practical workaround. It is a planned design decision. Guo et al. (2025) reviewed LLM-based agent architectures and found that the most reliable ones are those where the LLM handles planning and coordination while deterministic tools do the actual calculations [6]. The same principle appears in structural engineering research. Chen and Bao (2025) built a multi-agent LLM system for automated reinforced concrete design, published in Automation in Construction, and found that keeping the computation outside the language model layer was what made the system accurate and trustworthy [7].

The Python framework used to build the agent in this thesis is LangChain. It lets you attach Python functions to an LLM as callable tools, retains the discussion history between turns, and handles the logic of deciding which tool to call and when. Guo et al. (2025) describe LangChain as one of the most widely used frameworks for this kind of agent architecture, particularly because of LangChain's modular configuration and standardized tool interfaces [6].

A number of groups have recently explored using LLMs to automate FEM simulations. Chapter 2 reviews this work in detail. The short version is that these systems are good at generating input files and running solvers from natural language descriptions, but they tend to assume that the user provides complete and correct parameters from the start, and they do not do much to help the engineer understand or verify the results afterward.

The gap this thesis addresses is the part that happens before and after the computation. Before the simulation the system helps in collecting parameters by conversation, looking up material data with engineer approval, and confirming before anything runs. After the simulation: generating figures automatically and providing engineering commentary on what the results mean. And throughout the entire process: keeping the engineer informed, and in control

through technically enforced checkpoints. Chapter 2 documents this gap in detail through the literature review.

1.2 Problem Statement

The practical starting point for this thesis is an observation that anyone who has used Code_Aster will recognize. Running a beam analysis is not conceptually difficult. But the steps involved are tedious, and there are a lot of them. You need to know how Gmsh geometry files are structured, how to name physical groups correctly so Code_Aster can find them, how to use AFFE_CARA_ELEM to define the beam cross-section, how to write the boundary condition block so it fits the physical supports, and how to set up the output requests so you get the specific tables you need for post-processing. None of these things are particularly hard once you know them. But until you do, they take up a lot of time and attention that would otherwise go toward the actual structural problem.

When something goes wrong, and it usually does a few times before things work, diagnosing the problem means understanding both the structural model and the solver's error conventions at the same time. Tapeh and Naser (2023) point this out explicitly, identifying the learning curve of FEM tools as one of the main reasons they are underused in teaching and by practitioners who work outside their core specialty. [3].

The first part of the problem is that the workflow is fragmented. The geometry is defined in one tool. The mesh is generated in another. The solver runs in its own environment. The visualization is done somewhere else. The engineer is responsible in charge of maintaining everything uniform across all these progressions. Each handoff between tools is a chance for a naming error, a unit mistake, or a modeling assumption to get lost.

The second part of the problem is the gap between the numbers and the answer. When the solver finishes, it outputs tables of displacement values, stress values, and reaction forces. Those numbers are correct, but they do not tell the engineer whether the beam is adequate. To get from the numbers to an engineering judgment, you need to compare the deflection against a serviceability limit, check that the stress is below yield, verify that the reactions sum to the applied load, and think about whether the result makes physical sense. For an experienced engineer, this is routine. For someone earlier in their career, it is where mistakes happen. Mosqueira-Rey et al. (2023) identify exactly this kind of interpretation gap as one of the main motivations for HITL AI design. It is the step where human expertise matters most, and where AI-assisted guidance can make a real difference [12].

The third part of the problem is about trust. Any automation that hides what it is doing is worse than no automation. If the system silently fills in a default value, or runs a simulation with parameters the engineer has not explicitly agreed to, or produces a result that cannot be traced back to specific inputs, then it fails at the most basic level. Papagiannidis et al. (2025) reviewed frameworks for responsible AI governance and concluded that human control at the critical decision points is not optional when AI is deployed in professional scenarios containing real consequences [13]. In structural engineering, that not simply a governance principle. It is a safety requirement.

These three issues together, that are fragmented workflow, interpretation gap, and lack of transparency, are what this thesis tries to address. The goal is a system that reduces the overhead of running a beam analysis in Code_Aster, helps the engineer understand the results, and does both of these things in a way that keeps the engineer fully aware of and in control of every step.

1.3 Objectives

The thesis has five objectives. Writing them down precisely matters because they define the limits of what is claimed. Each objective is addressed in one or more later chapters.

O1. Build a working LLM-coordinated FEM agent. Design and implement a system where a responsive LLM agent collects beam parameters from the engineer in natural language and coordinates the full Code_Aster workflow. Creating geometry generation, meshing, command file generation, solver execution, and result extraction without the engineer needing to interact with any of the underlying tools directly. (Chapters 3 and 4)

O2. Implement human-in-the-loop safeguards at the code level. Design an interaction architecture where the engineer must explicitly confirm all parameters before any simulation runs, any material properties retrieved from the catalog require engineer approval before use. The session also can be stopped at any moment. These safeguards must be enforced in the code itself, not just through instructions to the LLM. (Chapter 4)

O3. Build an automated visualization framework and a usable frontend. Implement four visualization tools that produce six engineering figures from the solver output files without any manual post-processing step, and deliver them through an interface that includes run history, artifact inspection, and side-by-side comparison of results over multiple runs. (Chapter 5)

O4. Evaluate the system on representative case studies with defined metrics. Design and run five case studies that test the system under different conditions. that means controlling correct

input, missing material data, ambiguous units, unsupported boundary condition, and a focused interpretation query. the system should measure the results against five metrics: numerical accuracy, HITL compliance, interaction efficiency, interpretation quality, and scope behavior. (Chapter 6)

O5. Position the system in the literature and state its limitations clearly. Compare the system to related work in LLM-assisted engineering, structural automation, and decision support, and give an honest account of what the system does not handle well and what would be needed to extend it. (Chapter 7)

One thing this thesis does not claim: it does not claim to solve the general problem of integrating LLMs with FEM software. It demonstrates one working example of that integration, for a defined structural typology, evaluates it on five case studies, and reports the results honestly including the parts that did not work as expected.

1.4 Scope of the Work

This section says what the thesis covers and what it does not. Some boundaries were chosen deliberately, others came from time and tool constraints. Both types are stated here because knowing what was left out is as important as knowing what was included.

The structural problems used throughout the thesis are simply-supported beams under a central point load and cantilever beams under a tip load. They are modeled as a one-dimensional Euler-Bernoulli beam element in Code_Aster. This was chosen because it has a well-known analytic solution. maximum deflection, maximum moment, reactions, which makes it easy to verify the FEM results without any ambiguity. Sabat and Kundu (2021) note that the simply-supported beam under a central load is the standard verification problems in structural FEM, used consistently in the literature for exactly this reason. [14] It is also familiar enough that any structural engineer can tell whether the system is behaving correctly, without needing deep FEM expertise.

The analysis is linear static. The material model is linear elastic. The cross-section can be rectangular, defined by height H_Y and width H_Z or can be defined as general section by giving A, I_y, I_z, J_x, A_y, and A_z. These choices are consistent across all five case studies and all visualization outputs.

The system runs locally on a single machine. The frontend interface, the agent, the tools, and the solver all run in the same Python environment. The only external dependency is the OpenAI API for LLM inference.

The LLM used in all evaluations is GPT-4.o, accessed via the OpenAI API with temperature set to zero. Setting temperature to zero makes the model's responses deterministic for a given input, which is important for the reproducibility of the case study results.

The system scope is limited to simply supported and cantilever one-dimensional beam models and does not support continuous, or frame structures, nor two or three-dimensional shell and solid elements. Dynamic and nonlinear analyses, including seismic response, vibration, plasticity, and large deformations, are outside the scope, and material yield stress is used only for utilization ratio calculations rather than nonlinear simulation. The framework also does not perform formal code compliance checks with standards such as Eurocode 3 or AISC 360, instead providing only general engineering interpretation and verification metrics. In addition, the prototype is designed as a single-user local research system without multi-user, authentication, or security features. Although the architecture allows replacement of the LLM backend, all case studies in Chapter 6 were performed using GPT-4.

1.5 Thesis Structure

The thesis has seven chapters. The order follows a natural logic. Chapter 2 establishes what the field already knows, Chapters 3 to 5 describe what was built, Chapter 6 evaluates it, and Chapter 7 draws conclusions.

Chapter 2 reviews the literature on FEM workflows, visualization, LLMs in engineering, human-in-the-loop AI, and decision support systems, and identifies the research gap addressed by this thesis. Chapter 3 presents the overall four-layer system architecture and workflow. Chapter 4 focuses on the human-in-the-loop interaction framework, including the interaction logic, behavioral rules, confirmation mechanisms, and AI interpretation process. Chapter 5 explains the visualization framework, frontend, and data management structure. Chapter 6 evaluates the system through five case studies using defined performance measures, while Chapter 7 summarizes the findings, compares them with existing research, discusses limitations and future improvements, and concludes with the thesis contributions.

CHAPTER 2 – LITERATURE REVIEW

2.1 FEM Workflows and Challenges

The finite element method (FEM) is one of the most widely user numerical tools in structural engineering. It converts a continuous mechanical problem within a discrete set of equations that can be solved computationally. Its application covers static and dynamic analysis, linear and nonlinear behavior, and a wide range of structural typologies. Despite its numerical power, FEM is not a single operation. It is a multi-stage workflow, and the pragmatic challenges that arise at its stages are attracting growing research attention.

The workflow starts with defining the model. Engineers choose how to idealize the structure, select element types, set boundary conditions, and decide on load cases. These choices are important and affect the accuracy of the results, but they are not usually made explicit or guided by strict rules. Instead, engineers rely on their own experience and judgment. This makes the setup phase time-consuming and increases the risk of mistakes. A scientometric review of artificial intelligence, machine learning, and deep learning in structural engineering notes that model preparation is a major source of inefficiency, with much of the engineering effort spent on input preparation rather than analysis [3].

Another challenge is that finite element method (FEM) workflows are often split across different software. Geometry is usually created in CAD or BIM tools, analysis happens in a separate solver, and results are checked in different post-processing software. Moving between these steps can lead to inconsistencies. Data often need to be reformatted, assumptions restated, and it becomes harder to track changes. Reviews of AI-assisted structural design show that the lack of smooth integration between design, analysis, and verification is still a major problem [15].

Even after a simulation runs successfully, making sense of the results can be difficult. Finite Element Method (FEM) outputs are usually large and complex. One analysis might yield displacement maps, stress distributions, reaction forces, and modal shapes simultaneously. To understand what these results mean for engineering, you need to consider the context. For example, a high-stress area could indicate a design problem, a modeling error, or a stress concentration at a support. Telling the difference between these cases takes experience. Studies on structural identification show that modeling uncertainty can strongly affect how results are interpreted, and simply looking at visuals may not give the full picture [16].

Collectively, these issues indicate that the primary challenge of implementing the finite element method (FEM) in practice is not computational. Modern solvers demonstrate both speed and numerical reliability. Instead, bottlenecks arise within the broader workflow, including model setup, meshing, data exchange, and result interpretation. Addressing these challenges requires solutions that assist engineers throughout the entire process rather than focusing solely on the solve step.

2.2 Visualization in Structural Analysis

Visualization is the main way engineers access and understand simulation results. Since Finite Element Method (FEM) outputs are numerical, they need to be turned into graphics to be useful. Common tools like contour plots, deformed shapes, and vector fields help with this. Without these visuals, FEM results are abstract and hard to use. So, visualization is not just for looks—it is an essential part of the analysis process.

In the last ten years, visualization tools have improved a lot. Early post-processing software only showed static images. Now, engineers can interact with results by rotating models, using clipping planes, checking nodal values, and switching between load cases in real time. Augmented reality (AR) has added even more features. Huang et al. (2017) created a system that overlays FEM results onto real structures with AR, so engineers can see simulation outputs in the real-world context [17]. Other studies show that AR-based visualization helps engineers find critical areas and better understand how structures behave [18].

Despite these advances, a key limitation remains. Visualization tools show data, but they do not explain it. For example, a contour plot of von Mises stress highlights areas of high stress, but it does not indicate whether those values are acceptable, whether they result from a modeling assumption, or how they compare to a design limit or another setup. The engineer has to interpret these results. The quality of this interpretation depends on the individual's expertise, which brings subjectivity into the analysis. As a result, two engineers might look at the same data and reach different conclusions [17].

Another limitation is the mental effort required. FEM studies often include several analyses, such as different load combinations, design options, and sensitivity checks. Comparing results from these cases is not easy. Visualization tools often simplify data into scalar fields or basic images, which can obscure local effects or lead to confusion. Uncertainty in model inputs, mesh resolution, and discretization methods further complicates visual interpretation. Studies on

structural identification have found that relying only on visual inspection, without systematic comparison, can lead to wrong conclusions about how structures behave [16].

A third limitation is the gap between looking at visual data and making decisions. Visualization helps engineers spot patterns and find unusual results, but making decisions takes more than just seeing patterns. It needs context, comparisons to reference values, and judgment about possible outcomes. Standard visualization tools do not help with this higher-level thinking. They present data but do not interpret it. Because of this gap, there is a need for extra support, such as AI-assisted interpretation, which will be discussed in the next sections.

2.3 Large Language Models in Engineering Applications

Large language models (LLMs) are neural networks trained on large collections of text. They can produce fluent, context-aware language and have achieved strong results in tasks such as question answering, code generation, summarization, and multi-step reasoning. Because of their flexibility, LLMs are drawing more attention in science and engineering. A review in Automation in Construction notes that AI techniques, including LLMs, are being rapidly adopted in engineering, leading to more efficient, code-compliant design workflows [15].

One of the most practically relevant applications of LLMs in engineering is as natural language interfaces for complex computational tools. In the past, running simulations meant knowing specific software commands, file formats, and parameter rules, which could be a big hurdle, especially for beginners. LLMs help by letting engineers describe what they need in plain language, and the system converts these requests into technical actions. In structural engineering, this has been used for tasks like creating geometry definitions, setting analysis parameters, and understanding building codes. A review of LLMs in the architecture, engineering, and construction fields shows that natural language interaction is now a main use case throughout the construction process [8].

These architectures, different agents handle different subtasks: one may interpret user intent, another may generate input files, and a third may verify results. The LLM acts as a coordinator and reasoning engine, while deterministic tools perform the actual computations. Chen and Bao (2025) demonstrated this approach in the context of reinforced concrete design, developing a multi-agent LLM framework suited of automating code-compliant structural design while maintaining explainability and verifiability [19]. A similar framework for automated structural analysis using LLMs was proposed by Jang et al. (2024), further illustrating the breadth of natural-language-driven engineering automation [20].

LLMs also have some clear limitations. The most serious is hallucination, which means creating content that sounds correct but is actually wrong. While this can be annoying in daily use, it can be dangerous in engineering. For example, if a made-up material property or load value is used in a simulation without verification, it could yield results that appear correct but are actually incorrect. Studies show that hallucinations happen more often with technical or numerical information, making it a big challenge for reliable use in high-risk situations [21].

Another limitation is numerical precision. LLMs are probabilistic models, so they do not do math in the traditional sense. Instead, they predict likely answers based on patterns they have learned. When a task requires exact calculations, correct units, or adherence to physical laws, this randomness is a problem. LLMs cannot replace numerical solvers. They can help set up inputs, read outputs, and explain results, but the actual calculations must be done by reliable, rule-based tools. This idea, known as the neuro-symbolic approach, is key to the system designs described by Chen and Bao (2025) and Xie et al. (2025) [15], [19].

A third challenge is trust and transparency. LLMs work by using learned statistical patterns, not clear step-by-step reasoning. This makes their answers hard to trace and check without outside validation. In engineering, decisions need to be justified and repeatable. If a system's reasoning is unclear, it is hard to meet these standards. Studies on responsible AI emphasize that using AI in professional fields requires ensuring it is reliable, understandable, and under human control [13]. These needs are the foundation for the HITL approach, which will be discussed next.

2.4 Human-in-the-Loop AI

Human-in-the-loop (HITL) is a design approach where human decisions are built into an AI-driven process instead of being replaced. In this setup, people do more than just receive automated results. They play an active role by checking assumptions, guiding analysis, and making final decisions. This difference matters because it sets HITL apart from both fully automated systems and purely manual processes. The outcome is a collaborative workflow where human expertise and computer power support each other.

The theory behind HITL comes from responsible AI governance, socio-technical systems theory, and human factors research. In AI governance, HITL helps maintain accountability. Papagiannidis et al. (2025) call it a type of assisted intelligence, where AI gives recommendations but humans keep control over important decisions [13]. This is especially important in structural engineering, where mistakes can directly affect safety.

From a socio-technical perspective, Herrmann and Pfeiffer (2023) argue that AI systems should not be judged without considering their organizational and human contexts [22]. How well these systems work depends on how human expertise fits into the decision process. Even the best algorithm can perform poorly if it does not match human workflows, while a simpler system that fits well with engineering practice can do better. So, HITL design needs to focus not just on the AI itself, but also on how people interact with it, how information moves, and what mental effort is required from users.

One known risk with high automation is the out-of-the-loop problem. When AI systems do most of the work, users often stop paying close attention and just watch the results. This lowers their awareness of what is happening. If something goes wrong, they might not have the background they need to step in and fix it. Jiang et al. (2023) study this issue by looking at how users lose the ability to properly oversee AI outputs [23]. HITL design helps prevent this by making sure people stay involved when important decisions are made.

Research by Mosqueira-Rey et al. (2023) provides a comprehensive state-of-the-art review of HITL machine learning, covering active learning, interactive machine learning, and machine teaching as the main modalities of human involvement [12]. The review concludes that effective HITL systems entail careful design of interaction strategies: when to involve the human, what information to present, and how to process feedback. These design choices directly affect both system efficiency and user interaction.

In real-world use, HITL systems split responsibilities in a balanced way. AI takes care of tasks that need speed, scale, and consistency, like reading inputs, creating configuration files, and running simulations. People focus on tasks that need judgment and context, such as checking if results make sense, deciding whether to continue with a simulation, and analyzing unclear outputs. In structural analysis, this split makes sense. Algorithms are good for routine tasks, while human expertise is still needed for interpretation.

Building effective HITL systems can be challenging. Finding the right level of human involvement is a constant design issue. If people are not involved enough, oversight drops. If they are involved too much, it adds mental effort without much benefit. The way people interact with the system should make sure their input is thoughtful, not just a quick approval. These challenges are well known in engineering AI research and have shaped the design choices in this thesis.

2.5 Decision Support Systems

Decision support systems (DSS) are computer tools that help users make decisions when things are complex, uncertain, or when information is missing. Unlike systems that just process transactions or retrieve data, DSS are meant to support reasoning and evaluation. They bring together data management, analytical models, and user interfaces so decision-makers can explore options, weigh consequences, and choose between alternatives [24].

The application of DSS in engineering is well established. Structural design requires making many decisions that must meet codes, standards, and client needs. Traditional DSS help by including engineering rules, offering tools to refine parameters, and allowing users to compare scenarios. Morrison et al. (2023) found that engineers see a DSS as useful only if it matches how they actually make decisions [25]. A system that is technically capable but poorly aligned with engineering workflows will not be adopted in practice.

One major advantage of modern DSS is that they can bring together different types of information. Structural decisions rely on factors such as material properties, geometry, loads, safety factors, and costs. A DSS can collect these details from various sources and show them in one place. This makes it easier for engineers to gather information and compare design options. In engineering, this is especially helpful when decisions depend on both numbers and expert judgment.

Traditional DSS have some key drawbacks. The first is that they are often rigid. Many use fixed rules and require structured inputs, which works for routine problems with clear parameters. But when a problem is new, when information is missing or unclear, or when it is not obvious how to decide, these systems fall short. In structural analysis, engineers often face problems that do not fit standard templates, and a rigid DSS cannot handle these situations.

Another limitation is that DSS are not always easy to interpret. They can give recommendations, but do not always explain how they reached them. If an engineer receives a design suggestion from a DSS but does not understand the reasoning behind it, they cannot properly evaluate it. This is especially important if the DSS advice conflicts with the engineer's judgment or if the situation involves assumptions that the DSS cannot handle. Miller et al. (2017) found that understanding DSS outputs can be just as demanding as making the decision itself [26].

A third drawback is that DSS and simulation tools are often separate. In structural analysis, the DSS and the FEM solver are usually different systems. Engineers have to move parameters by

hand, look at results in one place, and enter data again in another. This separation makes the process less smooth and stops the DSS from giving real-time feedback based on simulation results.

Combining LLMs and HITL design, as discussed in Sections 2.3 and 2.4, can help solve these problems. LLMs offer flexible, language-based interaction that adapts to context. HITL design makes sure engineers stay in control of important decisions. Together, these features can make DSS more adaptable, easier to understand, and better connected to simulations. This integration is the main idea behind the system proposed in this thesis.

2.6 Research Gaps and Motivation

This chapter reviews five related topics: FEM workflows and their challenges, visualization as the main way to interpret results, LLMs as new AI tools for engineering, HITL design for responsible AI use, and DSS as the framework for decision support. While each area has made significant progress, there is still a major gap in how these advances are brought together to help engineers throughout the analysis process.

The first gap is the imbalance between automation and interpretability. Recent research on AI in structural engineering mostly aims to automate result generation. For example, Automation in Construction has published several studies showing that LLM-based systems can create geometry definitions, set up solvers, and check code compliance accurately [15], [19]. These advances are important, but automation alone can cause problems. Engineers may receive results they did not create and might not fully understand. Human verification is still necessary, but current frameworks do not provide enough support for this step [19].

The second gap is the weak connection between visualization and decision-making. As discussed in Section 2.2, visualization tools have improved a lot. They let users explore results interactively and help with spatial reasoning. However, these tools do not offer context or structured ways to compare results. Engineers still have to decide for themselves whether a result is acceptable, meets expectations, or meets design requirements. No current system addresses this gap systematically.

The third gap is the limited flexibility of DSS for unstructured engineering problems. As mentioned in Section 2.5, traditional DSSs work well for routine, clearly defined tasks. They are less effective when problems are new, inputs are missing, or the decision process depends on the context. Studies in socio-technical systems show that good decision support needs better

integration between human thinking and computer tools [22]. Current DSS architectures do not achieve this level of integration.

The fourth gap is the lack of a integrated framework that combines the three components above. Present approaches are partial. Some provide automation without explanation. Others provide visualization without decision support. Others provide AI interaction without engineering validation. What is missing is a system that integrates visualization, AI-assisted interpretation, and human control within a coherent workflow specifically designed for structural simulation environments.

This research gap can be stated precisely: To put it simply, there is no widely accepted framework that combines visualization, AI-assisted interpretation, and human-in-the-loop decision support in one simulation-integrated system for structural analysis. Current methods handle only parts of the problem. None covers the decision-support process from model confirmation to result interpretation and engineering judgment. This gap is the main reason for this thesis. The system proposed here combines visualization and LLM-assisted interpretation within a HITL framework. It keeps the engineer as the main decision-maker and uses AI to help explain and understand simulation results. This approach matches the growing agreement in human-centered AI research that AI should support, not replace, human expertise[13], [22].

CHAPTER 3 – SYSTEM ARCHITECTURE

3.1 Overview of the Proposed System

Structural simulation using Code_Aster involves a multi-stage workflow that extends well beyond the solver itself. Before any computation can begin, the engineer must define a geometry, generate a finite element mesh, configure material properties and boundary conditions, specify the load case, and prepare these set of inputs in at least four software-specific input files. After computation, the raw numerical results must be extracted, then should visualized, and interpreted in engineering terms. In conventional practice, these stages are performed manually, or using intermediary software with a graphical interface like Salome_Meca, and require the engineer to manage data exchange between them. It should be noted that the input files of Code Aster must strictly adhere to its syntax and a typo or blank space can completely stop the solution process. It is also worth noting that the language of definition of all variables is in French. This situation makes it difficult to work with Code-Aster text files. so that if an engineer wants to use the LLM provider's services directly to create input files without using our system, they have a very low chance of being able to run Code Aster.

The system described in this chapter replaces this fragmented, manual process with a coordinated, LLM-driven workflow in which each stage is run by the AI-agent, it is responsible for validating each steps result, and connect each step to the next.

The proposed system is not a chatbot wrapped around a solver. It is a layered architecture in which each layer has a defined responsibility and communicates with adjacent layers through well-specified interfaces. The layers are described below and illustrated in Figure 3.1.

Layer 1. User interface layer

The engineer communicates with the system in natural language. The interface is conversational. the engineer describes their structural problem, and the system extracts the necessary parameters from that description. The user interface is neutral and neutral in design. The architecture supports any conversational front-end, including web-based or graphical interfaces, without changes to the underlying system. This layer is where the human-in-the-loop mechanisms are expressed. the engineer's inputs, approvals, and stop commands all originate here.

Layer 2. LLM agent layer

The LangChain-based agent, powered by GPT-4.o, acts as the system's coordinator and reasoning engine. it parses natural language input to extract simulation parameters, identifies what is missing and asks follow-up questions, retrieves engineering knowledge from external sources when needed. enforces the eight behavioral rules defined in the system prompt, and generates natural language interpretations of simulation results. The agent does not compute and does not directly manipulate files. It reasons about what should happen next. dispatches tool calls, and manages the conversation memory across turns.

Layer 3. Tool calling

Seven deterministic Python-based tools implement. Each tool has a defined input parameter set. a defined output file, and is wrapped by the safe_call() validation mechanism. agent calls tools only after the parameters were collected. The tools are: a geometry generator, a Gmsh mesher, a Code_Aster command file generator, an export file generator, the Code_Aster solver invocation, validator, and four visualization tools.

Layer 4. Workspace and file layer

All intermediate and final outputs are written to a workspace directory. every tool reads from and writes to files in this directory. This design makes every intermediate result inspectable and auditable, and makes the system resumable from any intermediate step.

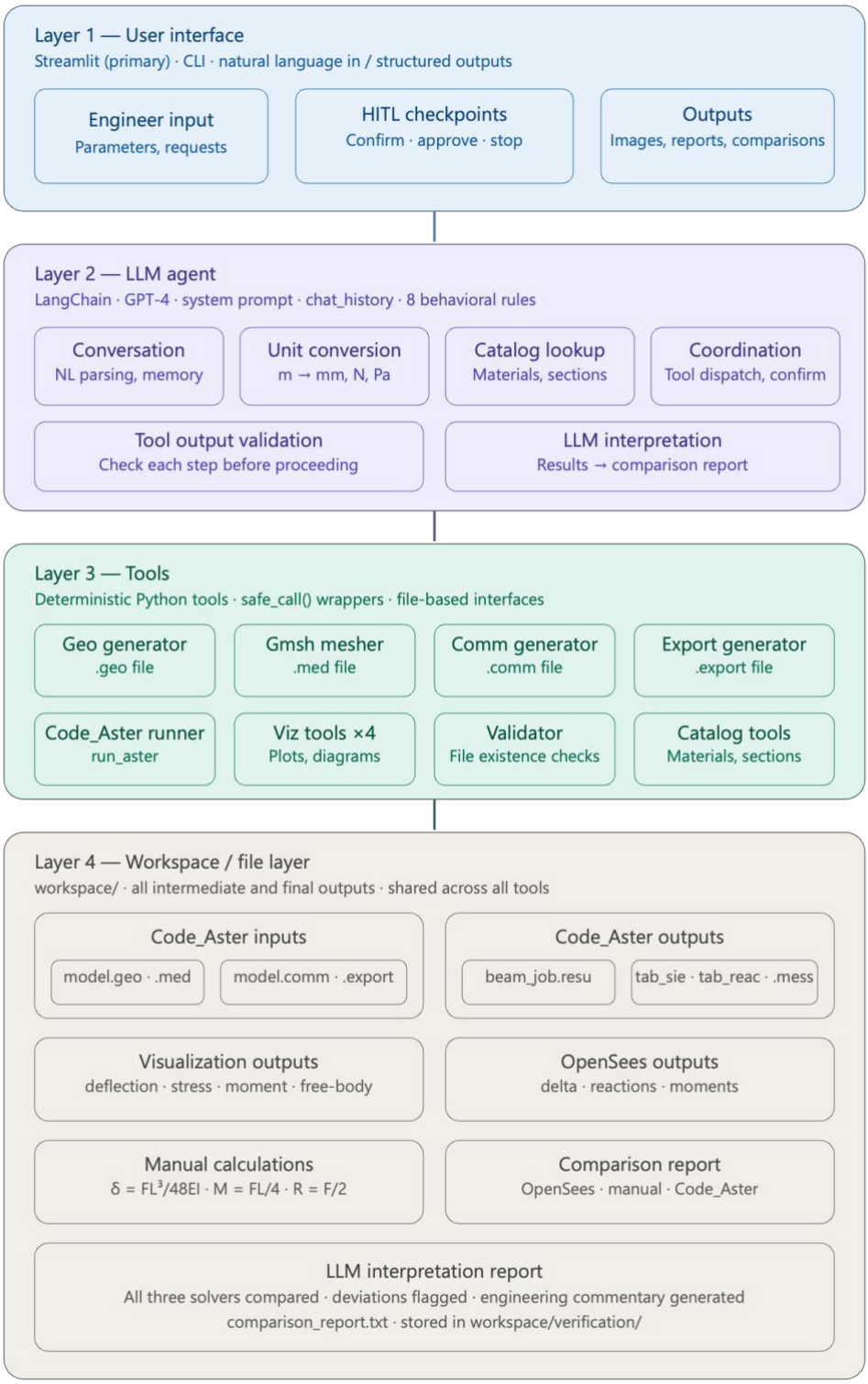

Figure 3.1 System architecture: four functional layers with their components.

Design principles

Four design principles govern the system architecture. The first is separation of concerns. the LLM handles reasoning and communication. Deterministic code handles computation and

validation. These two responsibilities are never mixed. The second is fail-fast execution: any tools calling that fails halts the entire process immediately rather than allowing corrupted or missing outputs to propagate to subsequent steps. The third is file-based traceability: every intermediate result exists as a named file in the workspace. it maks the process's history auditable. The fourth is human authority. that means no simulation executes without an explicit engineer confirmation, and the engineer can terminate the session at any point.

3.2 System Workflow

The workflow of the system is the sequence of events from the first engineer message to the final interpreted output. It is not a simple linear execution. it includes human decision points, iterative parameter collection, and optional consultation steps. Figure 3.2 illustrates the complete workflow. The workflow is divided into five phases, using the same phase names as chapter 4, where each phase is described in greater interaction-design detail.

3.2.1 Workflow phases

Phase 1. Information Gathering: The agent identifies which of the eight required simulation parameters are present in the engineer's initial message and which remain unknown. It asks targeted follow up questions across successive turns until all parameters are known or the engineer requests default values. This phase cannot be skipped. The agent will not execute with incomplete parameters.

Phase 2. Consultation: When the engineer does not know one or more parameter values (material properties or standard sections) the agent retrieves them from engineering references via the catalog, presents them with citations, and waits for the engineer's approval before locking them as simulation inputs. It may be skipped entirely if all parameters are provided directly.

Phase 3. Confirmation: The agent presents a complete summary of all eight parameters in user facing units and requests explicit confirmation. This is the primary human-in-the-loop checkpoint of the system. The confirmation gate is enforced at two levels. as a behavioral rule in the system prompt, and as a code-level default (confirm=*False*) in the running tools.

Phase 4. Execution: The AI_agent executes the simulation. The engineer is not in the loop during this phase. The six tools run step by step, each validated by *safe_call()* before the next begins. This is described in detail in Section 3.2.2.

Phase 5. Interpretation: The agent generates natural language explanations of the simulation verification. The system provides a report to analyse the accuracy of the analysis. This report is a comparison between the results obtained from the Code-Aster, OpenSees, and automated manual calculations. The engineer evaluates the results and may accept the results, or request a new simulation. If a new simulation is requested, the loop returns to Phase 1 with session memory intact.

3.2.2 The execution sequence within Phase 4

The tools calling internal execution within Phase 4 is a not fixed, consists of chain of six tools. Each step must succeed before the next begins.

Step 1. Geometry generation: The generate_geo_tool receives the validated parameters and writes a Gmsh-format .geo script defining the beam geometry as a one-dimensional line entity with physical groups for the nodes and element sets. The geometry is parameterized by beam length and element count, both of which are converted from meters to millimeters at the start of tool execution before this step executes.

Step 2. Meshing: The *mesh_with_gmsh_tool* invokes the Gmsh binary with the *.geo* file as input and produce a *.med* mesh file in MED/HDF5 format, which is the mesh format required by Code_Aster. The mesh contains the discretized beam geometry and the element connectivity data. this file is not readable. it is a binary file.

Step 3. Command file generation: The *generate_comm_tool* writes the Code_Aster *.comm* input script. This file defines the complete finite element problem: it assigns the material (DEFI_MATERIAU). it defines the cross-section that can be defined as a general section (AFFE_CARA_ELEM with A, I_y, I_z, I_T, A_{vy}, A_{vz}), or as a rectangular Euler-Bernoulli beam element (AFFE_CARA_ELEM with HY and HZ). It applies the boundary conditions (AFFE_CHAR_MECA) that can be simple supported or cantilever. Here the transverse load F_Y assigns at the midspan node in simple supported or on tip for cantilever, specifies linear static analysis (MECA_STATIQUE). it also requests displacement, stress, and reaction force output.

Step 4. Export file generation: The *generate_export_tool* writes the *.export* job configuration file. This file tells Code_Aster where to find: its input files (.comm and .med), outputs, the job name, and solver settings. It is a lightweight configuration step but is critical. A malformed export file causes the solver to fail before any computation begins.

Step 5. Solver execution: The *run_code_aster_tool* invokes the Code_Aster executable (as_run) with the export file as its argument. Code_Aster reads the *.comm* and *.med* files, executes the linear static finite element analysis, and writes four output files to the workspace consist of the *.resu* nodal result file containing displacement values, the *.mess* solver log, the *tab_sie.txt* stress table, and the *tab_reac.txt* reaction force table.

Step 6. Visualization: Five visualization tools run sequentially, each reading one or more of the result files and producing a plot. the displacement profile (from .resu), the von Mises stress distribution (from tab_sie.txt), the bending moment and internal force contour (from tab_sie.txt), and the free-body diagram with support reactions (from tab_reac.txt). These tools are one of the direct contribution of this study and are described in detail in Section 3.3.7.

3.2.3 Safe_call() and tool calling integrity

Every tool invocation in the running process is wrapped by the safe_call() function. This wrapper performs three operations for each step. it logs the tool name and input arguments. it catches any exception raised by the tool and reports it without crashing the calling process. and it checks for the presence of the expected output file after the tool returns. If any of these checks fails or if the tool raises an exception, returns an error string, or fails to produce its expected output, safe_call() returns False and the run halts immediately. The subsequent step never executes on a missing or corrupted input. By halting immediately at the point of failure and logging the error, the system ensures that the engineer is always informed of the actual problem and never presented with results derived from a failed computation.

3.2.4 Unit conversion at the phase boundary

All simulation parameters are stored in meters at the agent level throughout the session. The conversion from meters to millimeters is performed as a single explicit operation at the start of run before any tool is called. This conversion affects three parameters: length_m (beam length, m to mm), H_Y (section height, m to mm), and H_Z (section width, m to mm). The remaining parameters like E, NU, RHO, F_Y, n_elems are dimensionally consistent with the millimeter unit system and require no conversion.

Performing the conversion at the phase boundary rather than inside individual tools is a design choice with two benefits. First, it creates a single, auditable location for the unit transformation. if a dimensional error occurs, it can only have originated in one place. Second, It allows the tools in Layer 3 to work exclusively with millimeters. matching the native conventions of

Code_Aster and Gmsh, while the agent layer and the engineer interface work exclusively with meters, matching the engineer's natural unit system.

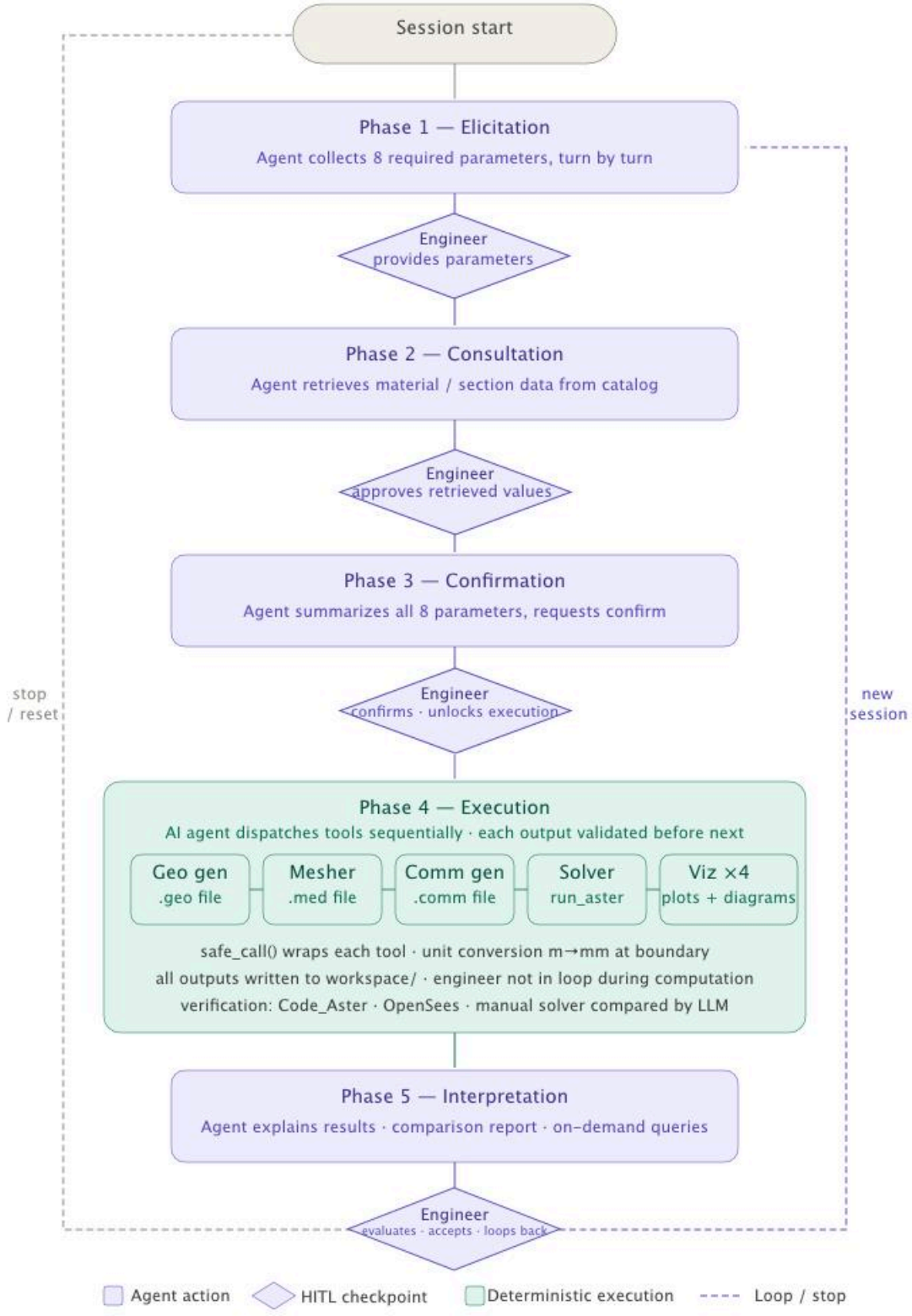

Figure 3.2 System workflow from session start to result interpretation.

CHAPTER 4 – HUMAN-IN-THE-LOOP DECISION SUPPORT SYSTEM

4.1 Overview of the Proposed System

The system in this chapter is based on two key ideas. The first is separation of roles. Which means the LLM handles reasoning, communication, and coordination, while the computational chained tools performs all numerical calculations. The second idea is human control. It means that the engineer approves every step. No simulation input is used without approval, and no step runs without confirmation. These two principles shape the system and guide all design decisions described next.

The system operates also across three functional layers. Together, they form a complete decision support workflow from natural language input to interpreted simulation output.

Interface layer

The engineer communicates with the system in natural language. There are no forms to fill, no parameter schemas to navigate, and no software-specific commands to learn. The engineer describes their structural problem as they would to a colleague. The system extracts structure from that description. In this thesis, the interface system is designed to work is the interaction style. it proceeds the work step-by-step with conversation in natural language, with memory across turns.

Agent layer

The LLM acts as a coordinator and reasoning engine. It parses natural language input to extract simulation parameters, identifies what is missing and asks follow-up questions, retrieves engineering knowledge from external sources when needed. Enforces behavioral rules defined in the system prompt, and generates plain-language explanations of simulation results. Critically, the LLM does not compute. It does not directly execute tools. It reasons about what should happen and coordinates the actions across excursion of tools. It is the engineer agent, not the one responsible for the engineering request. It does its duty only after the engineer has authorized each transition. It orchestrates the using of tools, only with the engineers permission in critical point.

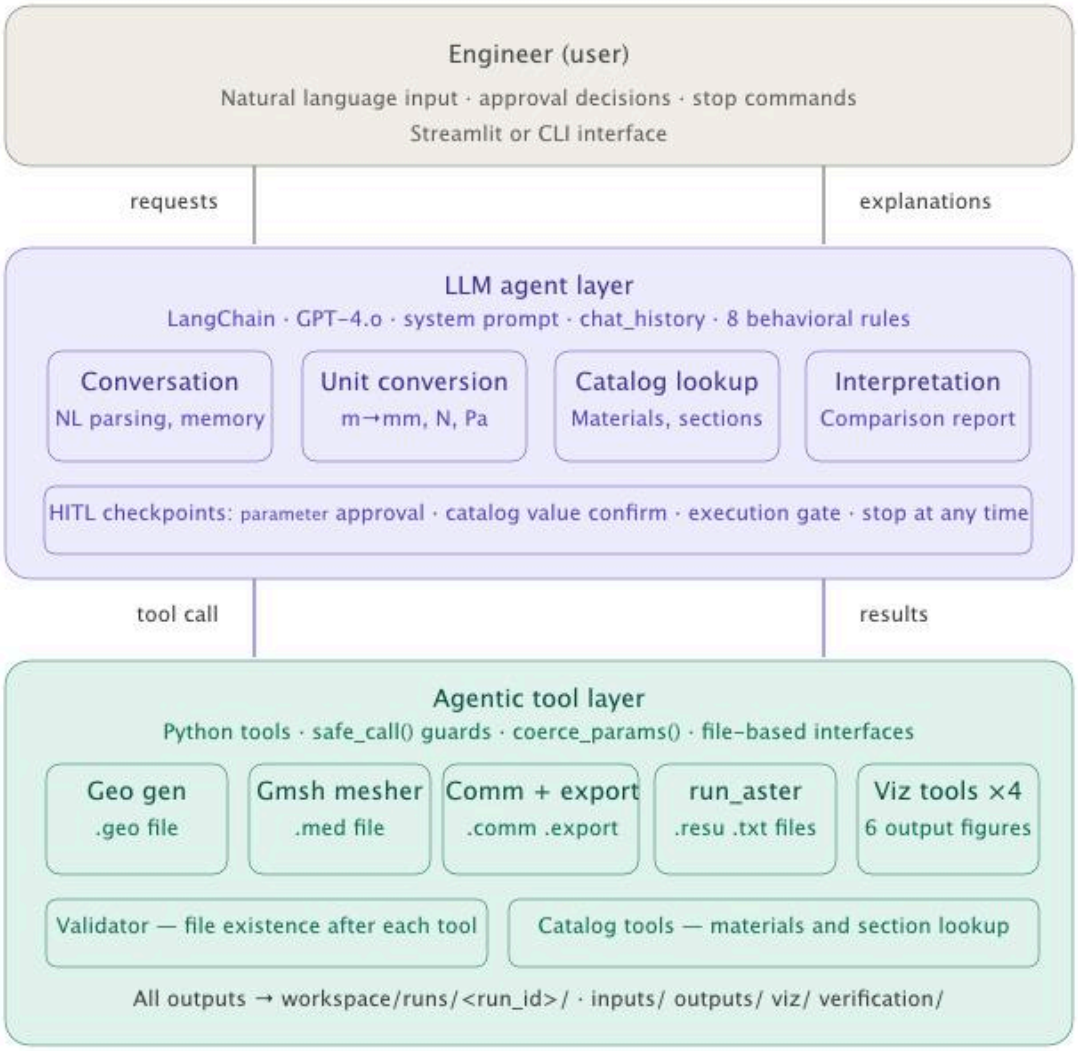

Figure 4.1 System architecture showing the three functional layers and the bidirectional flow of information between the engineer, and the LLM agent.

Behavioral rules as design decisions

The LLM agent layer is governed by eight behavioral rules embedded in the system prompt. These are not configuration options. They are engineering design decisions that determine how the system behaves in every session. They are summarized here as design principles:

Rule 1. No tool may be called until all eight required simulation parameters are known. This prevents the system from silently substituting default values for inputs the engineer has not yet provided.

Rule 2. Default values are never assumed unless the engineer explicitly requests them. This ensures that every parameter in the simulation reflects an active decision.

Rule 3. When material properties or sections are unknown, the system suggests its own library providing its references to European codes. the user is able to enter their data or choose from the suggested list.

Rule 4. All parameters are summarized in user-facing units and explicit confirmation is required before the tools executes. This is the primary human-in-the-loop checkpoint.

Rule 5. If the engineer responds negatively or requests a change, no tools are called. The agent asks what should be modified and waits.

Rule 6. Stop commands are honored immediately. The agent acknowledges and halts all workflow without exception.

Rule 7. The simulation may run only once per explicit confirmation. Automatic reruns are prohibited.

Rule 8. Conversation memory is maintained across all turns. The engineer may build up a problem description incrementally without repeating prior information.

These rules collectively define a system that is collaborative by design. The LLM is a capable and knowledgeable assistant. The engineer is the decision-maker. The remainder of this chapter explains how each aspect of this architecture is implemented.

4.2 Interaction Design

Interaction design in this system refers to the rules, conventions, and mechanisms that govern the exchange of information between the engineer and the agent. These choices are as consequential as the computational architecture. Interaction design shows whether the system is easy to use, fits the engineer's way of thinking, and whether the human-in-the-loop steps are real decision points rather than just simple confirmations.

4.2.1 The conversational paradigm

The choice of natural language as the primary interaction medium is deliberate. Engineers mostly do not think about structural problems in terms of parameter schemas. They think in terms of physical descriptions. Like a beam with a certain length, made of a certain material, carrying a certain load. Although working with engineering software over the years has structured their minds. A form-based interface imposes a rigid structure on this description and forces the engineer to translate their problem into a predefined format before the system can understand it. A conversational interface reverses this. the system takes on the duty of

extracting structure from natural language, and the engineer describes their problem in whatever order and level of detail feels natural.

This approach also handles incompleteness. The engineer may not know all eight required parameters at the start of the session. They may know the geometry but not the material, or the load but not the cross-section dimensions. A conversational system can identify gaps and ask targeted follow-up questions. Additionally building up the parameter set incrementally across multiple turns. A form-based system would require the engineer to have all information ready before submission or would silently substitute default values. It is a significant source of simulation error.

4.2.2 Session modes

The chat mode is the primary mode for this thesis. It supports multi-turn sessions with persistent memory. The engineer and agent exchange messages across as many turns as needed, and the agent retains all information provided in earlier turns. The engineer may revise, clarify, or extend their problem description at any point without repeating what has already been established.

4.2.3 Memory and state across turns

Conversation memory is implemented as a list of message objects. alternating human and agent messages. Which means that is appended after each exchange and passed to the agent with every new request. This gives the agent access to the full history of the session at each turn. The engineer may refer to earlier statements without ambiguity. For example said that 'use the same beam length as before' or refine previously given values 'change the load to 3000 N'. It is applicable because the agent has the complete context.

This is not a trivial capability. It is what makes incremental specification possible and what allows the confirmation step in Phase 3 of the interaction loop to be meaningful. the parameter summary the agent presents at that point reflects everything established across all prior turns, not just the most recent message. It is a result of several questions and Answers between the engineer and LLM.

4.2.4 Two directions of information flow

A key design feature of this system is the information flows in both directions, between the engineer and the agent. Most human-in-the-loop systems are described as one-directional: the

AI system generates output and the human validates it. This system also supports the reverse, and both directions are treated as first-class interactions.

In system-initiated elicitation, the agent identifies missing parameters and asks the engineer for them. This is the standard elicitation mode: the agent drives the information collection process and the engineer responds.

When the engineer needs more information from the agent, the engineer actively queries the agent for information before deciding what inputs to use. For example, the engineer may ask: 'What are the standard material properties for S275 structural steel?' or 'What cross-section dimensions correspond to an HEB 200 profile?' The agent retrieves this information with references and presents it for the engineer's review and approval. The engineer then decides whether to accept the suggested values, modify individual parameters, or reject the proposal entirely. Only after approval do these values enter the simulation as inputs.

This bi-directional model is significant because it means the system supports engineers who are not certain about their inputs. which is a common situation in practice, particularly for students, junior engineers, or professionals working outside their primary specialization. The system does not demand expertise as a precondition. It provides it as a service.

4.2.5 Unit convention as interaction design

All geometry is presented to the engineer in meters and all forces in Newtons throughout the session. Internally, the tools operates in millimeters, as required by the Code_Aster and Gmsh tools. The conversion from meters to millimeters is performed silently as a tool at the boundary between the main work of the agent layer before it starts using geometry designing and analyzing tools. In other words, it means that the agent uses unit convention immediately before tool execution.

This is an interaction design decision, not a technical convenience. The meter-based convention matches the engineer's mental model of structural dimensions. A 3-meter beam is described as 3.0 m, not as 3000 mm. Using millimeters at the conversation level would introduce a potential source of input error. Engineers who are used to working in meters may accidentally enter millimeter values in the wrong field or miscalculate by a factor of 1000. By fixing the interface convention, this class of error is eliminated.

4.2.6 Stop and redirect mechanisms

The engineer always has control of the session. If they use a stop bottom, the agent immediately stops the workflow. then, it records the session, and does not perform any further actions. This ensures that, at any point in the process, the engineer can exit whenever they choose.

4.3 Input Handling & Validation

The simulation tool requires eight parameters before it can execute. These parameters define the complete structural specification of the beam analysis. These characters are categoriced in: type of the beam (simple or cantilever), its geometry, number of elements for mesh, material properties, cross-section dimensions, and applied load. This section explains how these parameters are collected, how unknown values are handled, and how the system ensures that every parameter entering the system has been explicitly approved by the engineer.

4.3.1 The required parameter set

The eight required parameters are listed in Table 4.1 with their physical meaning, the unit used, and the default value that applies only if the engineer explicitly requests defaults. The units shown are those the engineer works with. the agent performs all necessary conversions internally before starting.

These parameters define the complete structural specification of the beam analysis. its geometry, mesh elemts, material properties, cross-section dimensions, and applied load. This section explains how these parameters are collected, how unknown values are handled, and how the system ensures that every parameter entering the process has been explicitly approved by the engineer.

4.3.1 The required parameter set

The eight required parameters are listed in Table 4.1 with their physical meaning. the unit used in the engineer-facing interface, and the default value that applies only if the engineer explicitly requests defaults are listed. The units shown are those the engineer works with. the AI_agent performs all necessary conversions internally and initially.

Table 4.1 Required simulation parameters with physical meaning, user-facing unit, and default value.

Parameter	Physical meaning	User-facing unit	Default value
length_m	Beam length	m	3
n_elems	Number of finite elements	—	15

E	Young's modulus	Pa	2.10×10^{11}
NU	Poisson's ratio	—	0.3
RHO	Mass density	kg/m³	7850
HY	Cross-section height	m	0.12
HZ	Cross-section width	m	0.1
FY	Applied transverse load	N	−1500

No simulation can begin without all eight being known. This constraint is enforced both at the behavioral level. Through the system prompt rule that prohibits tool calls until all parameters are present. It is also implemented at the code level, where the agent checks for the presence of all eight before proceeding and returns an error message if any are missing.

4.3.2 Progressive collection

The agent collects parameters progressively across the turns of a session. It does not require the engineer to provide all eight values at once. Instead, it identifies which parameters have been established in prior turns, determines which remain unknown, and asks specifically for what is still needed. This approach follows how engineers define problems step by step. We think, check other necessities and complete the task. The system is designed to follow this natural process.

A critical rule governs this process. The agent never assumes default values without explicit authorization. This rule exists because silent defaults are one of the most common and least visible sources of simulation error. An engineer may think they are using their own material and geometry, but the simulation might be using default values that don’t match their design. By refusing to assume defaults, the system ensures that every parameter in the simulation corresponds to an active decision by the engineer.

The only exception is when the engineer explicitly says something equivalent to 'use default values' or 'use standard steel properties'. In this case, the agent applies the defaults listed in Table 4.1, presents them to the engineer for review, and requests confirmation before proceeding.

4.3.3 Engineer-initiated consultation for material and section properties

One of the most operationally significant features of the system is the support for engineer-initiated consultation. When the engineer does not know the material properties or section

dimensions they need, they can ask the system directly rather than looking up values externally and re-entering them. This three-step process is a core human-in-the-loop mechanism.

Step 1) Query. The engineer asks the agent for specific engineering data. For example: 'What are the material properties for S355 structural steel?' or 'What are the cross-section dimensions of an HEB 200 profile?' The query may be posed at any point in the session before, during, or after providing other parameters.

Step 2) Retrieval and presentation. The agent first searches predefined catalogs. In this application, material properties for concrete, steel, aluminum, and wood are stored in JSON files. For each material, a set of approved sections based on European and American standards is also included. JSON is used to avoid hardcoding and to support future system expansion. A catalog.py module in the library folder is responsible for reading these files. The agent checks this internal library first. If the requested data is not found, it then performs a web search. In all cases, the agent does not invent or approximate values. it retrieves them from authoritative sources and discloses those sources to the engineer.

Step 3) Approval gate. The agent asks: 'Do you accept these values? You may also modify any individual parameter before proceeding.' The engineer may accept the suggested values as presented, modify one or more parameters before accepting. or reject the proposal entirely and provide their own values. Only after the engineer approves does the agent lock these values as simulation inputs. Rejected or unconfirmed values are not used.

This process applies identically to material properties (E, NU, RHO) and cross-section dimensions. Engineers have several options for defining the section. They can select a rectangular section and specify Hy and Hz, choose a circular section, or define a general section by entering properties such as A, I_y, I_z, J_x, A_y, and A_z. This flexibility allows even engineers who are not familiar with standard section catalogs or material data to use the system effectively, while still maintaining full control over the values used in the simulation.

4.3.4 Deterministic validation

After all eight parameters have been collected and approved, the coerce_params() function is applied before any tool call is made. This function performs three operations. It casts all values to their expected numeric types. Floating point for continuous quantities, integer for the element count. It fills any remaining gaps from the default values listed in Table 4.1 only if the engineer has authorized default use, and it ensures type consistency throughout the parameter dictionary.

This function is a deterministic safety net that operates below the LLM layer. It does not depend on the agent's behavior and cannot be bypassed by an unexpected LLM response. Its purpose is to guarantee that the parameter set entering the pipeline is numerically valid regardless of how the collection conversation unfolded.

4.3.5 The pre-simulation confirmation gate

Before any tool in the system is called, the agent presents a complete summary of all eight parameters in user-facing units. meters for geometry, Newtons for load, Pascals for Young's modulus, and asks the engineer to confirm. The summary is explicit and complete. The engineer sees exactly what will enter the simulation before it runs.

This confirmation gate is enforced at two independent levels. At the behavioral level, it is a rule in the system prompt. the agent is instructed never to call run without presenting the summary and receiving an affirmative response. At the code level, the run function has a confirm parameter that defaults to *False*. The function cannot execute unless confirm=*True* is explicitly passed. Even if the LLM were to call the function without completing the confirmation dialogue, the tool-level guard would prevent execution and return a message instructing the agent to request confirmation from the engineer.

This dual enforcement is a defense in depth design. The behavioral rule governs the agent's normal operation, while the code-level guard provides a backstop that is independent of the agent's reasoning. the two layers together ensure that the confirmation gate is robust against both expected and unexpected agent behavior.

4.4 AI Interpretation and Decision Support

The simulation tool produces numerical outputs. It Includes displacement values, stress fields, reaction forces, and bending moment distributions. These numbers are necessary for engineering judgment but are not sufficient for it. An engineer looking at a maximum displacement of 4.2 mm cannot immediately determine whether this value is acceptable without knowing the span, the load type, the material, the intended application, and the applicable serviceability limits. the gap between a raw numerical result and an engineering judgment is the space this section addresses.

The AI interpretation component of the system bridges this gap by contextualizing simulation outputs. It should be noted that, it does not replace engineering judgment. it provides the information and framing that makes judgment easier and more reliable. The LLM reasons over

the result files generated by the agent and produces natural language explanations that connect numerical values to their physical and engineering meaning.

4.4.1 Displacement results

The primary displacement output of the simulation is stored in the result file produced by Code_Aster. The agent reads this file after the pipeline completes and performs several interpretive operations. it identifies the location and magnitude of the maximum displacement.

The agent then contextualizes the deflection value in two ways. First, it computes the deflection ratio $\frac{L}{\delta}$, where L is the beam span and δ is the maximum displacement, and compares this to commonly accepted serviceability limits. which is typically $\frac{L}{300}$ to $\frac{L}{500}$ for floors and beams in building structures, depending on the application. Second, the agent computes the theoretical value and compares it to the simulation result, providing a direct verification signal. A close match between the analytical and numerical values indicates that the model is behaving correctly.

4.4.2 Stress results

The stress output is extracted from the tabular stress file generated during post-processing. The agent identifies the maximum von Mises stress and its location along the beam. it interprets this value in relation to the material yield stress. For standard structural steels, the yield stress is a known quantity 235 MPa for S235, 275 MPa for S275, 355 MPa for S355, as specified in EN 1993-1-1. and the ratio of computed stress to yield stress (the utilization ratio) is a standard engineering metric for assessing whether a cross-section is within the elastic range.

4.4.3 Reaction forces and equilibrium verification

The reaction force output is stored in the tabular reaction file. It provides support reactions at the beam ends. The agent reads these values and performs a global equilibrium check. the sum of all support reactions should equal the total applied load. A deviation from this value beyond numerical rounding is a signal of a modeling error, and the agent reports it explicitly.

This equilibrium check is a simple but powerful verification step. It costs nothing computationally and provides immediate confidence that the model boundary conditions are correctly defined. It is one of the analytical verification checks that connects the AI interpretation to the broader verification and validation methodology of the project.

4.4.4 Bending moment and internal force distribution

The bending moment and internal force contour provideس a spatial view of the structural response. The agent interprets the shape of the distribution and connects it to the load case and boundary conditions.

4.4.5 Explanation structure and framing

Across all four output types, the agent follows a consistent framing principle. explanations are structured as engineering interpretations rather than raw numerical listings. The distinction is important. a raw numerical report only presents values and percentage differences between solvers. In contrast the agent contextualizes the comparison by explaining the level of agreement, identifying whether discrepancies are acceptable, and summarizing the overall structural consistency between Code_Aster, OpenSees, and manual calculations.

This framing transforms comparison outputs from simple tables of numbers into engineering validation statements that support decision-making. The role of the agent is therefore not only to display numerical differences, but also to interpret the significance of those differences in terms of model reliability, solver consistency, and expected structural behavior.

4.4.6 Limitations of AI interpretation

The AI interpretation component has important limitations that must be stated clearly. The LLM generates explanations by probabilistic reasoning over patterns in its training data, not by solving differential equations or applying structural engineering theory from first principles. Its physical reasoning may be incorrect or incomplete in cases that lie outside its training distribution. It has no access to structural design codes and cannot formally verify compliance with Eurocode, AISC, or any other standard.

For these reasons, all AI-generated interpretations are presented as assistive outputs rather than authoritative conclusions. second for reports a system designed to just fill necessary parts. which means that a framework for reports has been created and LLM is just able to fill some necessary parts. In general, the system explicitly labels them as AI-assisted interpretations and reminds the engineer that final engineering judgment remains their responsibility.

4.5 Hybrid Approach (Rule-based LLM)

The system is neither a purely LLM-driven agent nor a purely rule-based automation. It is a hybrid architecture in which each component is responsible for the tasks it handles reliably, and neither is asked to do what it cannot.

A purely LLM-driven system would be unreliable for engineering applications. Language models are probabilistic. they can generate plausible but physically incorrect interpretations. They fail to enforce behavioral constraints consistently, and cannot guarantee numerical precision. Deploying such a system for structural simulation without additional safeguards would pose unacceptable risks to result reliability. A purely rule-based system, on the other hand, would be rigid and inaccessible. it could not accept natural language input, could not retrieve engineering knowledge dynamically, and could not adapt to the varied and incomplete ways engineers describe their problems. The hybrid approach resolves this tension by assigning each type of task to the component suited to it.

4.5.1 The rule-based layer

The rule-based layer operates at two levels. The first is the behavioral level, implemented through the eight rules in the system prompt. These rules govern the agent's conversational behavior and decision-making process. They specify when tools may be called, What must happen before execution, how to handle missing information, and how to respond to stop commands. These rules are instructions to the LLM. it is expected to follow them, and in typical operation it does. However, because they are implemented as natural language instructions rather than code, they are not infallible.

The second level is the code enforcement level, implemented through deterministic Python functions that operate independently of the LLM. these cannot be bypassed by unexpected agent behavior.

confirm=*False* default. As described before, The run function requires confirm=*True* to execute the run. Without this flag, the function returns a message instructing the agent to request confirmation from the engineer. This is a hard constraint at the code level.

coerce_params(). Applies numeric type casting and range normalization to all parameters before they enter the analytic tools. it works regardless of how they were collected in the conversation.

safe_call(). Wraps every tool invocation with error handling, output logging, and output file verification. if a tool fails or its expected output file is not present, the system halts and the error is reported.

file_exists(). Validates that the expected output of each tool called step is present before the next step begins. The pipeline cannot advance past a failed step.

4.5.2 The LLM layer

The LLM layer is responsible for all tasks that require natural language understanding, contextual reasoning, and dynamic knowledge retrieval. These include parsing natural language input to extract simulation parameters, managing conversation memory and tracking which parameters are known, identifying missing inputs and formulating targeted follow-up questions, performing catalog or web searches to retrieve material properties and section data with references, generating pre-simulation parameter summaries, interpreting simulation results across four output streams, and producing structured natural Language explanations.

These tasks share a common characteristic which is they require flexibility. The space of ways an engineer might describe a beam, express uncertainty about a material, or ask about a result is too large to be encoded in fixed rules. The LLM handles this flexibility naturally. And a rule-based system would not.

4.5.3 Division of responsibility

Table 4.2 summarizes the division of responsibility between the two components. The principle is the LLM decides what to do and how to explain it. And the deterministic layer decides whether it is safe to proceed and whether the output is valid.

Table 4.2 Division of responsibility between the LLM reasoning layer and the rule-based enforcement layer.

LLM layer (reasoning)	Rule-based layer (enforcement)
Extract parameters from natural language	Type-cast and range-validate parameters
Identify missing inputs, ask follow-up questions	Enforce the 8-parameter completeness requirement
Retrieve material and section data via web search	Enforce the confirmation gate (confirm=False default)
Summarize parameters and request confirmation	Verify output file existence after each tool step
Interpret simulation results and generate explanations	Halt system on any tool failure and report error
Manage session memory across turns	Log all tool calls and their outcomes

No computation happens without both layers agreeing. The LLM must have completed the confirmation dialogue (behavioral layer) and the function must receive confirm=*True* (code layer) before any simulation step executes. this dual-layer architecture is the principal mechanism by which the system achieves reliability without sacrificing the flexibility of natural language interaction.

4.6 Explanation Generation

Explanation generation is the process by which the system transforms numerical simulation outputs into natural language statements that support engineering judgment. It is distinct from result reporting, which merely presents numerical values. and from visualization. which provides graphical representations of data. Explanation generation adds a third layer: it connects numbers and images to physical meaning, engineering context, and decision-relevant questions.

4.6.1 Connection to verification

The explanation generation component is directly connected to the analytical verification methodology described in Section 4.4. when the agent explains a displacement result, it computes the analytical prediction and compares it to the simulation value. the verification steps are designed to provide the engineer with evidence that the model is correctly specified and the simulation is producing physically consistent results. This connection between explanation and verification is one of the key contributions of the system.

4.6.2 Limitations and the role of engineer judgment

The explanation generation component has two important limitations. First, the LLM cannot apply specific structural design codes.Second, the LLM's explanations are generated probabilistically and may contain errors. The explanation of a result is not a guarantee of its correctness. Although they are generated by the results of Code-Aster, Opensees, and automated manual calculations and their comparison table. For this reason, the system consistently frames explanations as AI-assisted interpretations and prompts the engineer to apply their own judgment. The engineer remains the final authority.

4.7 Human–AI Interaction Loop

The interaction between the engineer and the system follows a structured loop with five distinct phases. each phase has a defined purpose, A defined set of actors, and one or more transition

points that require a human decision. The loop is not merely a description of how the software works. it is a designed artifact that determines when and how the engineer is engaged, and when the system is permitted to act autonomously.

Phase 1. Information Gathering

The session begins with the agent identifying what it essential to know. It parses the engineer's initial request, extracts any parameters already present, and systematically identifies what remains unknown. It then asks targeted follow up questions. one or a few at a time, to collect the missing values. The engineer responds at their own pace. providing information in whatever order and level of detail they choose. This phase continues until all parameters are either provided by the engineer or obtained through the consultation process described in Phase 2.

This phase cannot be skipped. The agent cannot execute tools with incomplete parameters. As described, the agent is instructed not to assume defaults without authorization. The elicitation phase is therefore a necessary engagement. It ensures that the engineer is actively involved in defining the problem before any computation begins.

Phase 2. Consultation

When the engineer does not know one or more parameter values, Phase 2 provides a resolution path. The engineer may query the system directly for engineering data, material of section, or the agent may identify the gap and offer to retrieve the relevant defult values. In all cases, the agent waits for the engineer's approval before any action for running the analysis.

This phase may also be skipped entirely if the engineer provides all parameters directly in Phase 1. The consultation mechanism exists to support engineers who need it. It does not impose additional steps on those who do not.

Phase 3. Confirmation

When all parameters are known, the agent presents a complete summary and requests explicit confirmation. This is the primary human-in-the-loop checkpoint of the entire system. The summary lists all parameters in user facing units. It makes the possiblity for the engineer to review the full specification at a glance before the simulation runs.

If the engineer does not confirm. if they identify an error, request a change, or are unsatisfied with the values the agent asks what should be modified, updates the relevant parameters, and

presents a revised summary. The loop can cycle through Phase 3 multiple times until the engineer is satisfied.

The transition from Phase 3 to Phase 4 is enforced at the code level by the confirm=*False* default. This means the confirmation gate is active regardless of the LLM's behavior. The agent cannot execute the run without a confirmed engineer decision.

Phase 4. Execution

Once the engineer has confirmed, the agent start using analyze tools and executing them. This phase is the only phase in the interaction loop where the engineer is not directly engaged. The computation runs sequentially through geometry generation, meshing, solver execution, and visualization. It this phase the safe_call() will use that wrapper monitoring each step and halting the system if any step fails.

The engineer's absence from this phase is intentional. numerical computation is the one stage of the workflow where human intervention adds no value and could only introduce error. There is nothing for the engineer to judge until the results are ready. Phase 4 is Therefore the only phase that is fully automated, and it is bounded on both sides: the confirmation in Phase 3 and the interpretation in Phase 5 by human decision points.

Phase 5. Validation

Before the results are interpreted, the system performs a validation stage to verify that the numerical workflow has produced consistent and reliable outputs. This phase acts as a control layer between computation and engineering interpretation. the agent compares key response quantities such as support reactions, internal forces, and when available displacements across Code_Aster, OpenSees, and manual analytical calculations. Percentage differences are computed and classified into agreement levels such as excellent, acceptable, or poor agreement.

This phase is partially automated. The consistency checks are performed by the system. But the engineer remains responsible for deciding whether the validated results are acceptable for the intended engineering purpose. Only after successful validation does the workflow proceed to Phase 6.

Phase 6. Interpretation

When the agent do the analyze and complete it, it presents an initial interpretation of all four output streams. The engineer receives displacement, stress, reaction force, and bending

moment diagrams, each following the three-component structure described in Section 4.6. The engineer then evaluates the results and decides what to do next.

Three continuation paths are available. The engineer may accept the results and end the session. They may ask follow up questions to do another analyze. They may request a new simulation with different parameters It causes the loop to return to Phase 1 with the session memory intact. The new simulation may reuse some parameters from the previous run and modify others. it allows iterative design exploration without starting over.

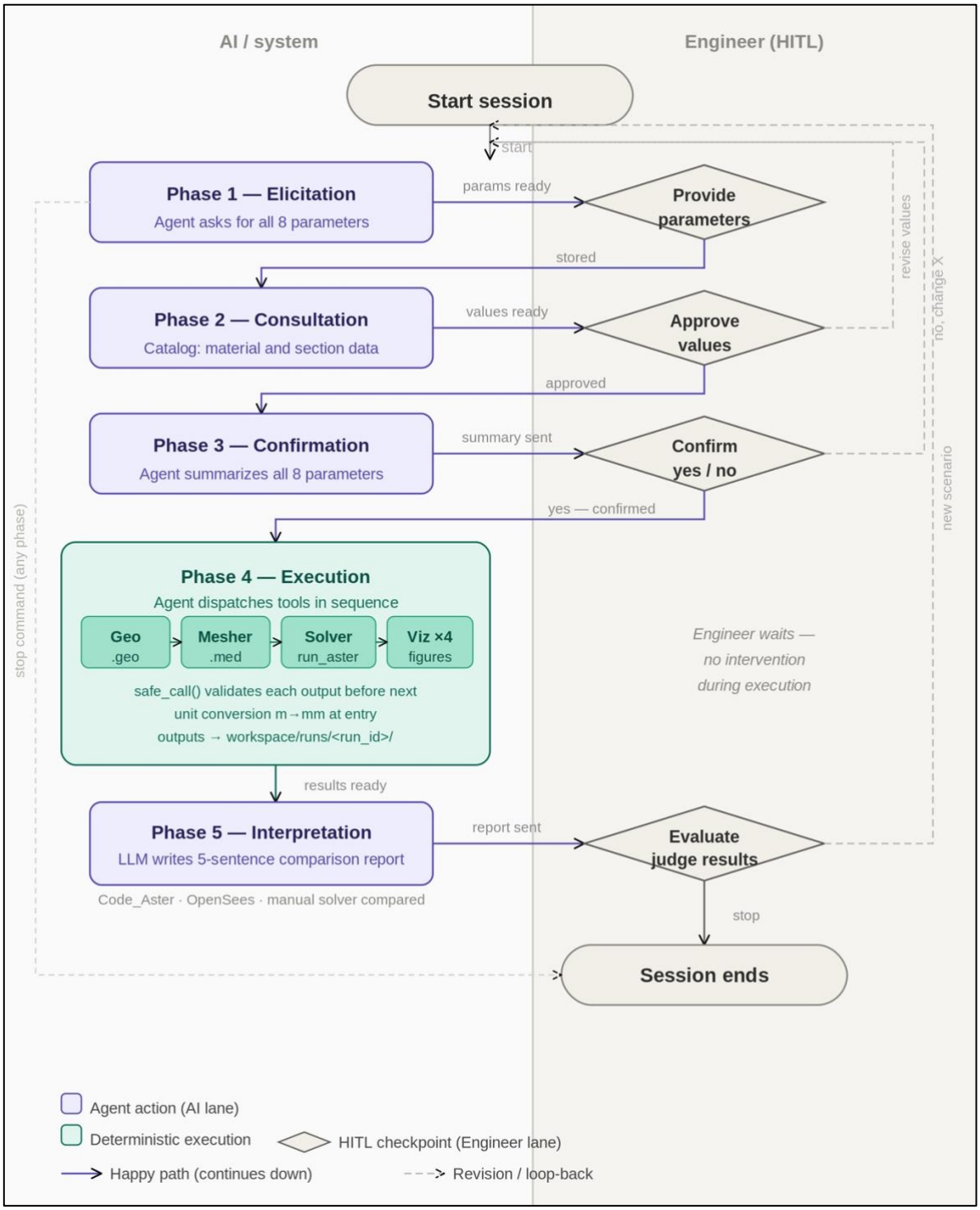

Figure 4.2 The five-phase interaction loop.

4.8 System Workflow Example

This section demonstrates the complete interaction loop through a concrete example session. The scenario is representative of a realistic engineering use case: a structural engineer wishes to analyze a simply-supported Steel beam with a known geometry and load, but is uncertain about the appropriate material properties and requests them from the system. This example

traces the session from the first message to the final interpretation, identifying the mechanism activated at each step.

Scenario

An engineer wants to analyze a simply-supported steel beam with a span of 3.0 meters, an HEB 160 cross-section, and a downward load of 2000 N applied at midspan. They know the geometry and the load. they are uncertain about the material properties for structural steel and request them from the system. Additionally, they use 15 finite elements for the discretization.

Table 4.3 Traced session demonstrating the complete interaction loop for a simply-supported steel beam analysis.

Turn	Role	Message	Mechanism activated
1	***Engineer***	I want to analyze a simply-supported steel beam. The span is 3 meters, the cross-section is HEB 160, and the load is 2000 N downward at midspan.	LLM extracts: length_m = 3.0, HY = 0.16 m, HZ = 0.16 m (HEB 160 nominal), FY = −2000 N. Missing: E, NU, RHO, n_elems. Phase 1 begins.
2	***Agent***	I have the beam length, cross-section dimensions, and load. To complete the setup, I need the material properties (Young's modulus, Poisson's ratio, density) and the number of finite elements. Do you know the steel grade, or would you like me to look up standard values?	System-initiated elicitation for 4 remaining parameters.
3	***Engineer***	It's S275 structural steel. Please look up the properties.	Engineer-initiated consultation trigger. Phase 2 begins.
4	***Agent***	For S275 structural steel (EN 1993-1-1): E = 210,000 MPa (2.10 × 10^{11} Pa), NU = 0.3, RHO = 7850 kg/m^3. These are the standard values for hot-rolled structural steel. Do you accept these values?	Web search performed. Values retrieved with reference to EN 1993-1-1. Consultation approval gate active.
5	***Engineer***	Yes, accept those. Use 15 elements.	Material values approved and locked. n_elems = 15. All 8 parameters now known. Phase 2 complete.
6	***Agent***	All parameters confirmed: • Beam length: 3.0 m • Cross-section: HY = 0.16 m, HZ = 0.16 m • Load: FY = −2000 N (midspan, downward) • Material: E = 2.10 × 10^{11} Pa, NU = 0.3, RHO = 7850 kg/m^3 • Elements: 15 Shall I run the simulation?	Phase 3: parameter summary presented. confirm=False guard active. Awaiting explicit confirmation.
7	***Engineer***	Yes, run it.	confirm=True passed. Phase 4 begins. executes: geo → mesh → comm → export → Code_Aster → visualize.
8	***Agent***	Simulation complete. Results summary: all visualizations displayed. do you want to do validation the output results?	Phase 5 begins. Automatic visualization across all 4 output streams.
9	***Engineer***	yes	Engineer ask for validation. Phase 5 begins.
10	***Agent***	the automated manual calculation done. the opensees run. the reports presented. a summary human readable report is prepared. A comparison summary that compairs all analaysis	Phase 5 validation is complete. Phase 6: Create a short summary based on all results.

		consist of Code-Aster, Opensees, and automated manual calculation is creted. An AI iterpretation is based on the results is: LLM Short Summary: The comparison report shows that Code_Aster, OpenSees, and Manual Analysis exhibit excellent agreement across all evaluated metrics, with 100% of comparisons falling within 1% of each other. Support reactions, internal force envelopes, and displacements were all consistent among the methods, indicating reliable results. No discrepancies greater than 1% were observed, confirming the accuracy of the analyses.	

Commentary

The session in Table 4.3 demonstrates all five phases of the interaction loop within a ten-turn exchange. Phase 1. Information Gathering runs through turns 1–2, where the agent extracts the available parameters and identifies what remains unknown. Phase 2 consultation: occupies turns 3–5. the engineer triggers the consultation process in turn 3, the agent retrieves and presents referenced material values in turn 4, and the engineer approves in turn 5. The consultation approval gate in turn 4 is the first human-in-the-loop checkpoint of the session. The agent presents information it has retrieved but does not act on it until the engineer confirms.

Phase 3 confirmation: is turn 6. The agent presents the full parameter summary and activates the confirmation gate. The confirm=*False* default in the run function is active throughout this turn. Phase 4 execution: It is triggered by the engineer's confirmation in turn 7, at which point the deterministic tools executes without further human input. Phase 7 interpretation: begins in turn 8 with the automatic result summary and continues through turns 9–10 with on-demand explanation.

Several specific mechanisms are worth noting. In turn 4, the agent cites EN 1993-1-1 as its source for the retrieved material values. This reference traceability is a direct product of the engineer initiated consultation workflow. the agent does not present values without sources, ensuring that the engineer knows the provenance of every parameter in their simulation. In turn 8, the automatic interpretation includes the comparison of all results. they are built-in verification steps that confirm the model is behaving correctly before the engineer evaluates the results.

The example also illustrates what does not happen in the session. The agent never assumes a material without asking. it never executes before turn 7. No default value is used without the engineer's knowledge. The session proceeds entirely at the engineer's pace. the process has

been done with the agent providing capability and the engineer retaining authority at every critical transition.

CHAPTER 5 – VISUALIZATION FRAMEWORK

This chapter describes the visualization framework developed as one of a core contribution of this thesis. The framework consists of two connected layers: a set of automated post-processing visualization tools that transform Code_Aster solver text output into six engineering figures. There is also a frontend interface that delivers those figures to the engineer. Streamlit-based frontend has been employed for this purpose within a stateful run-management environment. Together, these two layers, visualization tools and frontend, constitute the output side of the human-in-the-loop workflow described in Chapter 4. The mechanisms by which the system returns engineering understanding to the engineer after each simulation.

The chapter is organized as follows. Section 5.1 focus on the role of visualization in decision support and motivates the design choices. Section 5.2 describes the four implemented visualization tools, their inputs, outputs, and architecture. Section 5.3 explains the engineering significance of each of the six result types. To answer why these six figures has been selected to visualize the text based outputs of Code_Aster. Section 5.4 documents the Streamlit frontend as a designed system, covering its four result panels, its run storage model, and its comparison feature. The comparison feature also has been considered as the main interactive contribution of the frontend layer.

5.1 Role of Visualization in Decision Support

When the Code_Aster solver completes a simulation, the workspace will receive three primary result files: *beam_job.resu*, which holds nodal displacements and rotations at each finite element node (beam_job is our naming for the beam run. The important thing is the format of this file whitch is *.resu*); *tab_sie.txt*, which have element-level internal forces and moments. and *tab_reac.txt*, that holds support reaction forces at the boundary condition nodes. These files are numerically complete. they contain all information produced by the finite element analysis considering the analyse type and predefined outputs. as degaussed in chapter 3, user should declare the type of results that they want. However, these outputs are not in a way that can be extracted in the short time.

To explain the gap more clearly consider a *beam_job.resu* file for a 15-element meshed beam model. it will contain 16 rows of nodal data, each with six displacement and rotation components. For reading this table, an engineer cannot immediately determine whether the deflection profile is symmetric, whether the maximum deflection occurs at the expected

location, whether its magnitude is within an allowed limit, or whether the boundary conditions were correctly applied. A figure that plots the transverse displacement profile can communicate all of these things at a glance. This gap between structured numerical data and actionable engineering judgment is the problem the visualization layer is designed to solve.

The visualization framework in this system is different from general-purpose FEM post-processors. Tools such as ParaView, Salome-Meca, or Code_Aster's STANLEY module are designed for detailed interactive exploration of three-dimensional simulation results by experienced FEM practitioners. They are not suited for the context of this system, where the engineer is conducting a conversational analysis, results must be available immediately after the run completes. These figures should be visualized without any post-processing setup, and importantly, they must be readable by engineers who are not FEM specialists. The visualization tools implemented here are narrow in scope. each produces one or two specific figures for a specific result type. But targeted in their purpose: to give the engineer exactly the information needed to evaluate the simulation result and proceed with the next decision.

The design of the visualization layer is shaped by four requirements derived from the HITL architecture described in Chapter 4. First, automation: figures must be generated without any manual step, as part of the system's final phase, therefore the Results tab is generated automatically when the engineer enters Phase 5. Second, completeness: the six figures together must cover most structural response quantities that a structural engineer typically evaluates for a beam like: deflection, rotation, shear, moment, and reactions. The purpose is to leave nothing that requires the engineer to consult the raw files for routine assessment. Third, verifiability: each figure must include at least one element that serves as a self-check. That can be an equilibrium annotation, an antisymmetric signature, or an analytical comparison reference. Therefore, the engineer can detect modeling errors directly from the figure without consulting the agent. Fourth, comparability: the figures must be stored with consistent meaningful naming, formatting, and axis conventions across runs in a way that they can be placed side by side in the Compare tab without visual inconsistency.

The six figures produced by the framework are: beam_job_def.png (deflection profile), beam_job_rot.png (rotation profile), shear_diagram.png (shear force diagram), moment_diagram.png (bending moment diagram), mfz_contour.png (moment contour on beam schematic), and freebody.png (free-body diagram). They are described one by one in Section 5.2 and their engineering significance is discussed in Section 5.3. Their delivery to the engineer through the Streamlit frontend is described in Section 5.4.

5.2 Implemented Visualization Tools

The visualization layer is implemented in tools/visualize_tool.py. matplotlibis used for figure rendering and numpy for numerical processing. Four tool functions are defined, each decorated with @tool from LangChain, which gives them the same callable interface as the simulation tools registered in build_agent(). This registration has a practical consequence. The AI agent is responsible to use them in a proper sequence. Any individual visualization tool can be called by the agent on demand. for example, if the engineer asks the agent to regenerate a specific figure after modifying parameters, or if a visualization step failed during the pipeline run and needs to be retried. AI Agent manages to use them again, based on the user request.

5.2.1 Tool Architecture and Integration

The four visualization tools generate six figures. These visualization tools execute as the final phase of run after the Code_Aster solver has completed and its output files are confirmed present in the workspace with different file fomats. The system is designed to handle errors safely. Each tool is executed using the same `safe_call()` function. this calling system catches errors, logs them, and checks if the expected output file is created. However, unlike simulation steps, a failure in visualization does not stop the whole process. If the solver works correctly but a visualization tool fails the process it is still considered completed, but with warnings (e.g., due to unexpected formatting in `tab_sie.txt`). In this case, the engineer is free to access the raw output files in the Artifacts tab and can also ask the agent to regenerate specific figures if needed.

All output figures are written to: "*workspace/runs/<run_id>/viz/*". The file paths are registered in the SQLite artifacts table at generation time, that makes them discoverable by the frontend without any file-system scanning. The tool-to-output mapping is summarized in Table 5.1.

Table 5.1 Visualization tool registry: inputs, outputs, and result type for each of the four tools.

Tool	Input file(s)	Output file(s)	Result type
visualize_resu	beam_job.resu, tab_disp.txt	beam_job_def.png, beam_job_rot.png	Displacement D_Y, rotation D_{RZ}
visualize_stress	tab_sie.txt	shear_diagram.png, moment_diagram.png	Shear force V_Y, bending moment M_{FZ}
visualize_mfz_contour	tab_sie.txt + geometry/support meta	mfz_contour.png	Moment contour on beam schematic

visualize_freebody	tab_reac.txt, model.geo, model.comm	freebody.png	Support reactions and applied loads

5.2.2 Deflection and Rotation visualizer (visualize_resu)

The visualize_resu tool reads nodal displacement data from *beam_job.resu* and *tab_disp.txt.* The primary data of interest are the D_Y component and the D_{RZ} component that is rotation about the beam axis. The D_Y component is transverse displacement, that constitutes the deflection profile of the beam. the D_{RZ} component is rotation about the beam axis, and that is the slope of the deflection curve. The tool produces two separate figures.

beam_job_def.png is a line plot of D_Y versus position along the beam axis. The x-axis represents the position of beam in meters, running from the left support (x= 0) to the right support (x= L). The y-axis represents transverse deflection in millimeters, converted from the solver's output in meters by the unit conversion layer described in Section 5.2.6. It should be noted that, in some cases, to better show the details, the axis has been extended a little. The deflection is plotted as negative downward, consistent with the sign convention used in the *.comm* file. The maximum deflection value, its position along the beam, and the corresponding deflection ratio L/δ are annotated directly on the figure. Support symbols are drawn at x=0 and x=L to indicate the boundary condition locations. The combination of annotated maximum value and deflection ratio allows the engineer to immediately assess whether the serviceability limits is satisfied without any manual calculation.

beam_job_rot.png is again a line plot of D_{RZ} versus beam position in the same axis convention. Rotations are displayed in milliradians for better representation of small numerics. For a simple supported beam under a symmetric load, the expected D_{RZ} profile is antisymmetric about the midspan: positive (clockwise) at the left support, zero at midspan, and negative (counter clockwise) at the right support. This antisymmetry is a geometric consequence of the problem of symmetry and is independent of the material or section properties. The figure therefore checks and verified by system if that does not depend on knowing the correct numerical magnitude. the verification checks if the profile is out of symmetry, the boundary conditions or loading location have been specified incorrectly. Figure 5.1 shows the deflection of a cantilever beam with tip load.

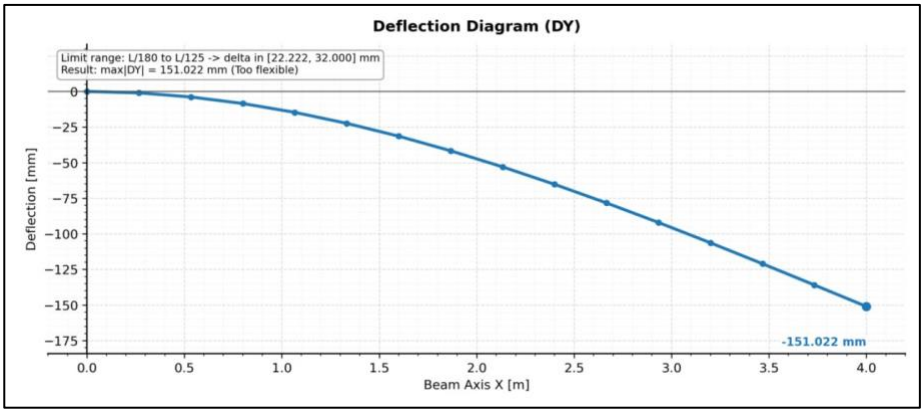

Figure 5.1 Deflection profile output of visualize_resu. The maximum is annotated directly on the figure. (cantilever with tip load, length 4 m, 15 elements, steel S275, section IPE180, load 15000 N)

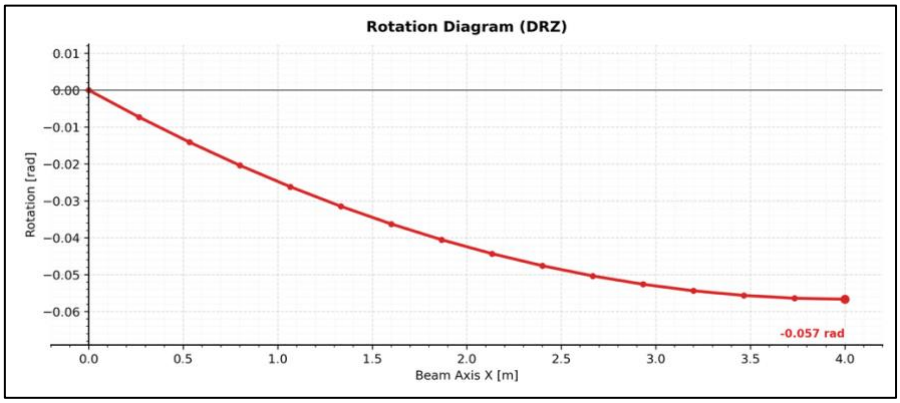

Figure 5.2 Rotation profile output of visualize_resu. (cantilever with tip load, length 4 m, 15 elements, steel S275, section IPE180, load 15000 N)

5.2.3 Shear Force and Bending Moment (visualize_stress)

The *visualize_stress* tool reads element-level force and moment outputs of Code_Aster run from *tab_sie.txt*. It is a tabular output file written by Code_Aster's IMPR_TABLE post-processing command. The file contains the generalized forces VY and MFZ at each integration point or node along the beam which are transverse shear force and bending moment about the z-axis, respectively. The tool produces two figures.

shear_diagram.png is a shear force diagram produced as a step plot. For a simple supported beam under a central point load, the shear force is constant between load points. +F/2 from the left support to the load application point. Then −F/2 from the load point to the right support. The tool draws the step function, annotates the shear values at the supports and at the load point, and marks the sign convention with arrows. The shear values at the supports are the

negative of the support reactions. This relationship connects the shear diagram to the free-body diagram and provides a consistency check between two independent outputs.

moment_diagram.png is a bending moment diagram extracted as a line plot. For a central point load on a simple supported beam, the moment distribution is triangular. zero at both supports and maximum FL/4 at midspan. The tool annotates the maximum moment value, its position, and the zero-crossing points. The analytical reference value $M_{max} = FL/4$ is printed on the figure for immediate comparison with the computed value. it gives the engineer a verification reference in the figure itself, without requiring consultation with the agent or external calculation.

The two figures are designed to be read in combination. The relationship $dM/dx = -V$ (in the sign convention used by Code_Aster) means that the slope of the moment diagram at any point should equal minus the shear force at that point. For the step shear force and triangular moment profile of the baseline load case this relationship is visually verifiable. The moment diagram has a constant positive slope over the half-span where $V_Y = -F/2$, and a constant negative slope over the half-span. Where $V_Y = +F/2$. Placing both figures in the same Results tab panel allows the engineer to verify this relationship visually.

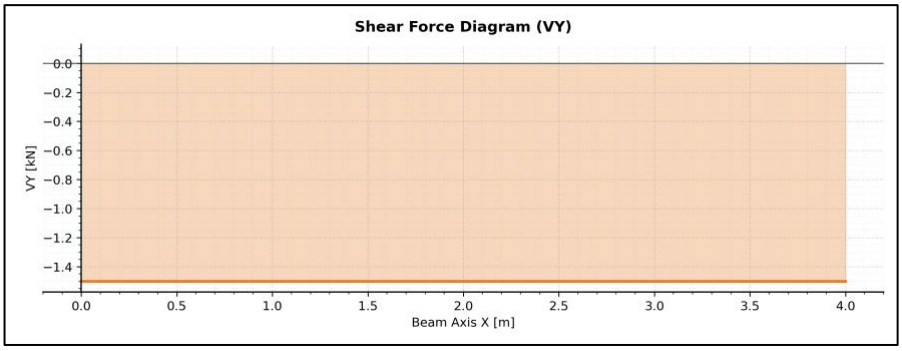

Figure 5.3 Shear force diagram output of visualize_stress. (cantilever with tip load, length 4 m, 15 elements, steel S275, section IPE180, load 1500 N)

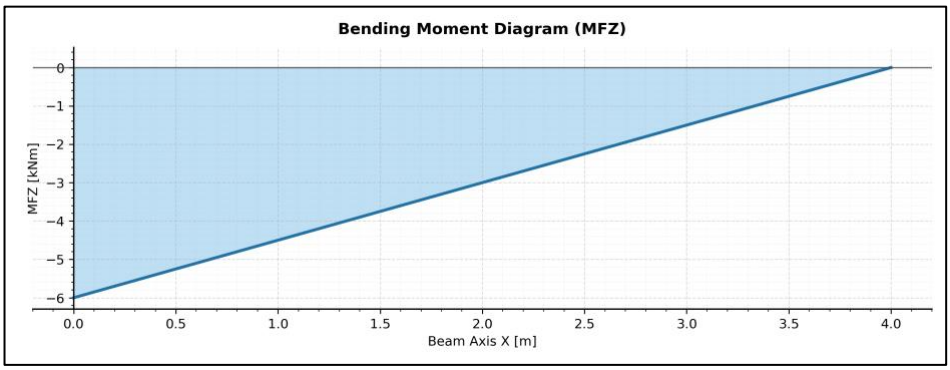

Figure 5.4 Bending moment diagram output of visualize_stres. (cantilever with tip load, length 4 m, 15 elements, steel S275, section IPE180, load 1500 N)

5.2.4 Moment Contour on Beam Schematic (visualize_mfz_contour)

The visualize_mfz_contour tool produces a two-dimensional representation of the bending moment distribution. It is overlaid on a schematic of the beam geometry. The moment line diagram shows MFZ as a function of position along the neutral axis. But here the contour figure maps the moment values to a color gradient applied across the beam's physical cross-section and span.

The tool reads MFZ values from tab_sie.txt. Then, it applies a colormap to represent relative moment magnitude. The beam is schematically drawn as an elongated rectangle. The support symbols are drawn at their positions at the two sides of the beam. The result is a figure that converses the distribution of the moment along the beam. the section geometric difference is visible in the contour figure. So that changing the beam cross-section can produce a different contour in response.

The value of this representation alongside the moment line diagram is spatial proximity. An engineer reading the contour figure can closely locate the region of highest moment intensity without searching for the maximum value in a line plot. The figure is also useful for the Compare tab feature, where multiple runs with different section geometries are shown side by side. The differences in both moment distribution and beam proportions are visible in a single look.

The tool requires additional context beyond the stress file alone. Ibasedon the type of the choose section, it reads the section properties from the run's *meta/run.json* to compute the beam aspect ratio, and also reads support node positions from *model.geo* to position the support symbols correctly. This cross-file reading makes visualize_mfz_contour the most input-dependent of the four tools. therefore, it is the tool most likely to fail if intermediate input files are missing or malformed.

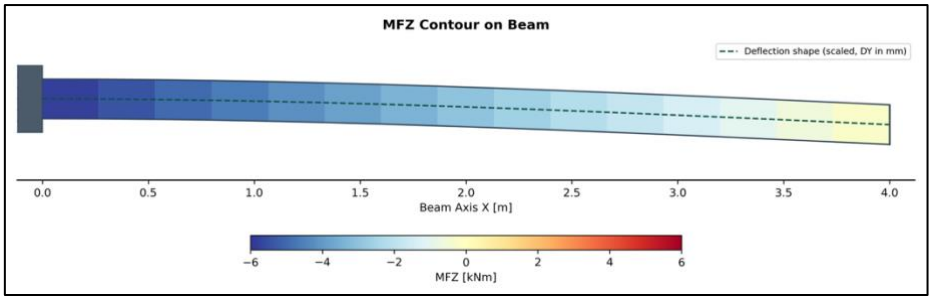

Figure 5.5 MFZ contour output of visualize_mfz_contour. The color field shows moment intensity; beam proportions are to scale. (cantilever with tip load, length 4 m, 15 elements, steel S275, section IPE180, load 1500 N)

5.2.5 Free-Body Diagram (visualize_freebody)

The visualize_freebody tool produces a free-body diagram that summarizes the complete loading state of the beam, what is applied to it, and what the supports react in return. It is the most complex of the four tools in terms of input requirements, reading from three separate files. tab_reac.txt is used for the computed support reaction values. model.geo is used for the support node positions along the beam axis, and model.comm for the applied load magnitude, direction, and application point.

The figure presents the beam as a horizontal line with support symbols (pin and roller for simple beam and rigid for cantilever beam) at the support positions. The applied load is shown as a downward arrow at the load application node, annotated with the load magnitude in kilo Newtons. The support reactions are shown as upward arrows at the support nodes, each annotated with its computed value. The sign convention downward forces negative, upward forces positive are considered.

The equilibrium can easily be observed and calculated by the user in the free-body diagram. As descussed the vertical reaction forces is computed and can be easily compared to the total applied vertical load. For a correctly configured model, this sum should equal the applied load magnitude: $R_A + R_B = |F_Y|$. If it does not which would indicate either an error in the model boundary conditions or a solver convergence failure. This equilibrium check is the most fundamental model verification step possible. it is independent of the structural theory, the material, the section, and the mesh, and a failure here is unambiguous evidence of a modeling error.

The three-source input structure also means that the free-body diagram reflects the actual configuration of the model, not a hard-coded assumption based on default values. If the engineer has employed a non-standard load position or an asymmetric beam, the free-body diagram will correctly show this geometry. Because it reads the load position from model.comm rather than assuming midspan.

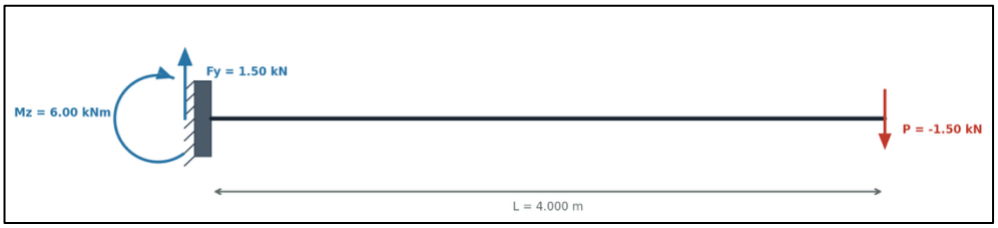

Figure 5.6 Free-body diagram output of visualize_freebody.

5.2.6 Unit Conversion Layer (units.py)

Code_Aster computes all quantities in the unit system. In Code_Aster unit can be changed based on the user's decision. However, the accuracy of the results is entirely up to the user to enter all the inputs with an united approach to get a correct output from the software. An approach that is common in many other finite element analysis software. in this study we used millimeters for length and Newtons for force. The native unit system of Gmsh and Code_Aster are consistent with the internal pipeline convention. On the other hand, the visualization layer displays results in units appropriate for engineering interpretation. Displacements are in millimeters (mm), rotations are in milliradians (mrad), forces are in kilonewtons (KN), moments are in kilonewton-meters (kN·m), and beam positions are in meters (m).

All these unit conversions are unified in a separate part named units.py. It exposes a single conversion function for each quantity type. The visualization tools call these functions rather than performing unit conversion several times inside the tools. It serves two purposes: 1) ensures that all six figures and reports use consistent units and formatting regardless of which tool produced them and 2) it makes unit changes. As an example, displaying forces in kilonewtons for structures with larger loads. It is a single-point modification rather than a change distributed across four tool files and reporting files.

The unit conventions used in the figures match those used in the agent conversation layer. The system prompt instructs the agent to present all geometry in meters and all forces in Newtons. The figures follow the same convention. X axis (deflection and rotation diagrams) is in meters, the y axis is in millimeters for deflection and milliradians for rotation, and the force and reaction annotations are in kilonewtons. An engineer reading the figures sees the same units they used when specifying parameters in the chat. It eliminates a potential source of confusion between the conversational interface and the visual output.

Table 5.2 Unit conversion applied by units.py between solver output and figure display.

Quantity	Solver output unit	Display unit	Conversion factor

Transverse displacement DY	m (meters)	mm (millimetres)	× 1000
Cross-section rotation DRZ	rad (radians)	mrad (milliradians)	× 1000
Shear force VY	N (Newtons)	N (Newtons)	× 1
Bending moment MFZ	N·mm	N·m	÷ 1000
Support reactions	N (Newtons)	N (Newtons)	× 1
Beam position (x-axis)	mm (millimetres)	m (meters)	÷ 1000

5.3 Result Types and Engineering Interpretation

Section 5.2 described what each visualization tool produces. This section explains what each result type means in structural engineering terms. Why it matters for the evaluation of the beam design and how it connects to the decision support workflow. The six result types together provide a proper picture of the structural response of a beam. They cover both kinematic quantities (displacement and rotation) and static quantities (shear, moment, reactions).

5.3.1 Deflection (D_Y)

The transverse displacement D_Y represents the vertical deflection of the beam at each node along its span. For a beam under a downward load, all D_Y values are negative (downward). For simple supported beam the maximum magnitude occurring at the load application point (at midspan for a central load). And for the cantilever beam, the maximum magnitude occurs at the tip of the beam.

The primary engineering significance of the deflection profile is serviceability. Structural design codes define maximum allowable deflection ratios L/n. where L is the beam span and δ is the maximum deflection. Limiting values are $\frac{L}{300}$ for floors and $\frac{L}{500}$ for beams supporting plaster or brittle finishes under EN 1993-1-1. A beam that satisfies strength requirements may fail serviceability if its span-to-depth ratio is too high or its stiffness too low. The deflection figure allows the engineer to assess serviceability immediately from the annotated L/δ ratio without any manual calculation.

The deflection result is also the primary metric for Metric M1 in the Chapter 6 evaluation. The analytical reference for a simple supported beam under a central point load is:

$$\delta_{max} = {FL^3}/{48EI} \qquad \textit{Eq. (5.1)}$$

where F is the applied load, L is the span, E is Young's modulus, and I is the second moment of area. and for cantilever:

$$\delta_{max} = FL^3 / 3EI \qquad \textit{Eq. (5.2)}$$

The deviation between the FEM-computed δ_{max} and this analytical value expressed as a percentage error ε_δ is the primary measure of numerical accuracy in the case studies.

5.3.2 Rotation (D_{RZ})

The rotation D_{RZ} represents the slope of the deflection curve at each cross-section. the angle through which the cross-section rotates relative to its unloaded position. For Euler-Bernoulli beam elements, D_{RZ} is the derivative of D_Y with respect to the beam axis: $D_{RZ} = \frac{D_v}{D_x}$.

The engineering significance of D_{RZ} is dual. In some structural applications beams supporting machinery, precision equipment, or crane rails, angular deflections are constrained in addition to vertical deflections, and the D_{RZ} profile allows the engineer to evaluate rotation limits at connection points or support locations.

More importantly for this system, the D_{RZ} profile serves as a model verification signal that is independent of the numerical value of the deflection. For example for a simple supported beam under a symmetric load about midspan, the D_{RZ} profile must be antisymmetric about the midspan. positive at the left support, zero at midspan, and negative at the right support, with the magnitudes at the two supports equal. This antisymmetry is a mathematical consequence of the symmetry of the problem. it holds regardless of the load magnitude, material, or section dimensions. If the computed D_{RZ} profile deviates from antisymmetry, the model has an error: the load is not centered, the boundary conditions are asymmetric, or there is a mesh or model definition error. The engineer can verify this from the figure without any calculation.

5.3.3 Shear Force (V_Y)

The shear force V_Y is the resultant of all transverse forces acting on one side of a cross-section. For a simple supported beam under a single central point load, the equilibrium of any free body between the left support and midspan gives $V_Y = +\frac{F}{2}$ and between midspan and the right support gives $V_Y = -\frac{F}{2}$. The shear diagram is therefore a two-step function with a single discontinuity at the load application point equal to the load magnitude F.

5.3.4 Bending Moment (M_{FZ})

The bending moment M_{FZ} is the resultant moment of all forces acting on one side of a cross-section about the centroidal axis. For a simple supported beam, under a central point load, M_{FZ} varies linearly from zero at both supports to a maximum of FL/4 at midspan. This triangular distribution is the most fundamental structural result in beam analysis. It is the primary basis for sizing the cross-section.

In the case of a cantilever subjected to a point load F at the free end, MFZ varies linearly from zero at the free end to a maximum of FL at the fixed support. This produces a triangular bending moment diagram, with the peak occurring at the support where the beam is restrained. This distribution is one of the fundamental results in beam theory and represents the critical condition for design. As the maximum bending stress, and therefore the governing requirement for cross-section sizing, occurs at the fixed end.

5.3.5 Moment Contour

The moment contour shows the same M_{FZ} distribution as the moment diagram, but directly on the shape of the beam. Contour are used to represent the size of the moment. It stronger colours show higher moments, while lighter colours show lower values. The highest moment appears clearly in the middle of the beam and end of the beam while the supports in simple supported beam are shown with zero moment for simple supported beam. On the other hand, for a cantilever beam he highest moment appears at the support and tip of the beam, shown with zero moment.

The moment contour does not provide additional numerical information beyond what the line diagram contains. Its contribution is representational: it places the moment distribution in the context of the physical beam geometry. A narrow, deep section, small H_Z, large H_Y, concentrates its moment capacity in the flanges and presents a different visual footprint than a wide, shallow section. This geometric context matters when the engineer is comparing designs.

This figure is the one most directly suited to the Compare tab. When four cases with different sections or loads are shown next to each other, the contour plots show both the moment values and the shape of the beam at the same time. Single line diagrams cannot show both together.

5.3.6 Reactions and Free-Body Diagram

Support reactions are the forces that the structural supports exert on the beam to maintain equilibrium. For a simple supported beam under a single vertical load, the reactions are purely

vertical: R_A and R_B, with $R_A + R_B = F$ by global equilibrium. For a symmetric load, $R_A = R_B = \frac{F}{2}$. For a cantilever beam under a single vertical load F, the support at the fixed end provides both a vertical reaction and a moment reaction. By global equilibrium, the vertical reaction is $R_A = F$, and a fixed-end moment $M_A = F \cdot L$ is developed at the support.

The engineering significance of the reactions extends beyond simple equilibrium. The reaction values are the design loads for the connections, foundations, and supporting members at each support. An unusually large reaction at one support relative to the other may indicate load eccentricity or modeling asymmetry. A reaction that does not satisfy equilibrium is a definitive indicator of a model error.

The free-body diagram is the only figure in the set that simultaneously shows all external actions on the beam. Both what is applied by the engineer's specified loading and what the boundary conditions impose. It is therefore the most informative single figure for a structural engineer reviewing an unfamiliar model. From the free-body diagram alone, the complete loading configuration of the beam can be read without reference to the model input files. This documenting quality is a deliberate design choice that supports the system's goal of making simulation results accessible to engineers who did not set up the model themselves.

5.4 Frontend Delivery

In this study, a web application that acts as the main user interface of the system was used to deliver the visualizations to the engineers. Streamlit serves as the primary frontend of the system. Streamlit was chosen due to its lightweight architecture, rapid prototyping capabilities, and native compatibility with Python-based scientific and data visualization workflows. The Streamlit application is not merely a viewer for the six figures. It is a stateful run management environment that organizes the engineer's interaction with simulation results across multiple runs, multiple sessions, and multiple workspaces. This section describes the application's architecture, its four result panels, and its run storage infrastructure.

5.4.1 Architecture and Design Rationale

Streamlit is a Python-native web application framework. It compiles a Python script into a reactive web interface running locally in the browser. The choice of Streamlit as the primary frontend is grounded in a specific property of this system. The approach of the simulation tool, the agent, and the frontend all run in the same Python process on the same machine. This eliminates the need for an HTTP API layer between the frontend and the backend. The

Streamlit application imports agent directly from main_agent.py, collects required inputs, and reads reports and figure files from the local workspace directory. For a local-first, file-based engineering research prototype, this architecture is both simpler and more maintainable than a client-server split.

The application supports multiple concurrent chat workspaces within a single session. Each workspace has its own conversation history. In each chat history it has ts own confirmed parameter set, and its own set of runs. The engineer creates a new workspace by clicking the New Chat button in the sidebar, and switches between workspaces using the workspace selector. Additionally, the user can see the list of all analysis that they do during that chat session with meaningful names. This multi-workspace model is directly motivated by the parametric comparison use case. Considering the situation that an engineer investigating the effect of section size on deflection can run beam analyses in three separate workspaces and compare their results in the Compare tab without losing the conversation history of any individual analysis.

Figure 5.7 Overview of the Streamlit frontend. The three-panel layout separates the conversation (left), results (center tabs), and run history (sidebar) into distinct functional zones.

5.4.2 Results Tab

The Results tab is the primary output panel of our frontend. It displays all six visualization figures produced by the most recent completed run in an image gallery. The figures are loaded from workspace/runs/<run_id>/viz/ using the run_id stored in Streamlit's session state, It is updated whenever a run completes. There is also possibility to see all runs that has been done

in the chat history. As showed in Figure 5.8 the engineer can choose any run it wants from the list.

The gallery containes all figures into a matrix. It contains the kinematic results, and the static results. Therefore it is Includes the deflection profile (beam_job_def.png), the rotation profile (beam_job_rot.png), the shear diagram, the moment diagram, the moment contour, and the free-body diagram.

After user confermation for run the tab Results feeds with the visualication of the Code_Aster outputs. At the moment safe_call() returns True for the final visualization step, Streamlit's active_tab state variable is set to Results. It causes the interface to display the figures without any user action. This automatic transition is the frontend expression of the Phase 4 to Phase 5 transition described in Chapter 4. The system moves from execution to interpretation mode without requiring the engineer to navigate.

Each figure is displayed at a size suitable for reading annotations. Clicking a figure in the gallery opens it at its native resolution, which is important for figures with small numerical annotations such as the deflection ratio or the equilibrium check. At the top there is a Dropdown menu that makes the possibility that user can select the run and see the visualized figures. The figure caption includes the run_id and name by meaningful words to help the user understand which input properties each run refers to. It allows the engineer to confirm that the displayed figures correspond to their desired run.

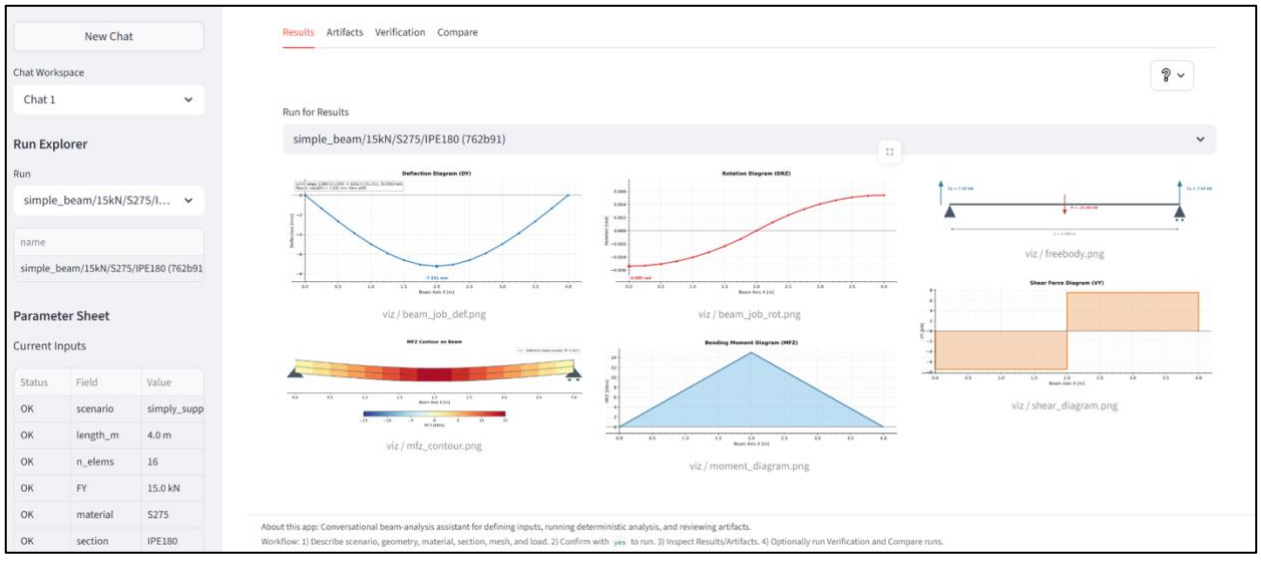

Figure 5.8 Results tab of the frontend.

5.4.3 Artifacts Tab

The Artifacts tab provides access to the raw files produced by the tools and Code-Aster. Both the input files that define the model and the output files generated by the solver. The tab presents a file tree organized by run folder. Inputs consists of model.geo, model.med, model.comm, model.export, and Outputs consists of beam_job.resu, tab_sie.txt, tab_reac.txt, beam_job.mess. File sizes and modification timestamps are shown alongside each filename. additionally, user can select among dropdown menu their desired run.

Selecting a file in the tree opens a preview panel. .txt, .resu, and .mess files are rendered as scrollable monospace text. .geo, .comm, and .export files are rendered as syntax-highlighted text. The preview panel allows the engineer to inspect the model input files. verifying, for example, that the FORCE_NODALE block in model.comm specifies the correct load magnitude and node, or that the boundary conditions in model.comm match what was specified in the chat.

The Artifacts tab serves two purposes. For the engineer, it provides transparency into the simulation model, the ability to see exactly what was submitted to Code_Aster and what the solver produced. For the thesis, it provides the evidence layer for Chapter 6. Screenshots of specific artifact contents confirm that the agent produced the expected input and output files for each case study. The Artifacts tab is the direct equivalent of the result reporting that would appear in a traditional simulation engineering report, but accessible interactively rather than compiled manually.

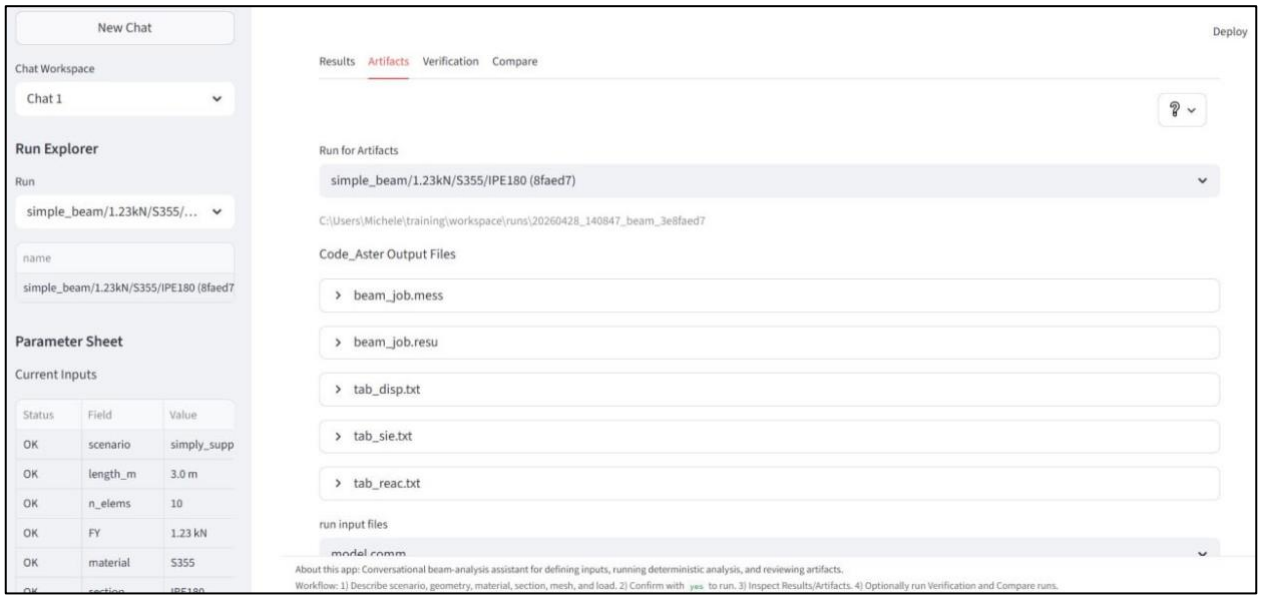

Figure 5.9 Artifacts tab of the frontend.

5.4.4 Verification Tab

The Verification tab is conditionally present. It is displayed only after a run reaches completed status. It means the solver succeeded and at least the primary output files are present. Its initial state is a single button that is Request verification. This opt-in design is intentional. Verification adds computation time, running an OpenSees model and a manual analytical solver. It is not always needed. For routine parameter exploration runs during the elicitation and comparison phases, the engineer may not require formal verification.

When the engineer clicks Request verification, the system runs three verification procedures sequentially. The manual analytical solver computes $\delta_{analytical}$, M_{max} and R from the confirmed parameters. The OpenSees finite element solver which provides an independent numerical result from a different code. It should be noted that, both of these two approaches extract their inputs from Code_Aster input files. Then, the comparison engine start working. It computes percentage deviations between Code_Aster, manual, and OpenSees results for each key quantity. The results are stored in workspace/runs/<run_id>/verification/.

When verification is complete, the tab displays three status rows. Manual solver, OpenSees, and Comparison. The comparison report shows the user the matching results in percentages in a table. Both the manual results are compared with the opensees output, and the average of both results is compared with the Code_Aster output. A summary table shows the key comparison metrics: δ_{max} deviation, reaction deviation, M_{max} deviation, and σmax deviation, and the location that these caclution has been done for. In this section, the evaluation of results is performed using a quantitative classification approach. For each comparison, the percentage difference is calculated and assessed against a 5% threshold. The classification is based on the worst-case value in each row (i.e., the maximum of M–O % and Avg vs CA %). Accordingly, the results are grouped into three categories: “excellent agreement” (less than 1%), “acceptable agreement” (between 1% and 5%), and “poor agreement” (greater than 5%). This approach provides a clear and quantitative summary of the level of agreement.

The verification results shown in this tab are the primary data source for the M1 (numerical accuracy) metric in Chapter 6. For the baseline case study (Case 1), all four quantities should be within the 5% threshold, confirming correct model formulation.

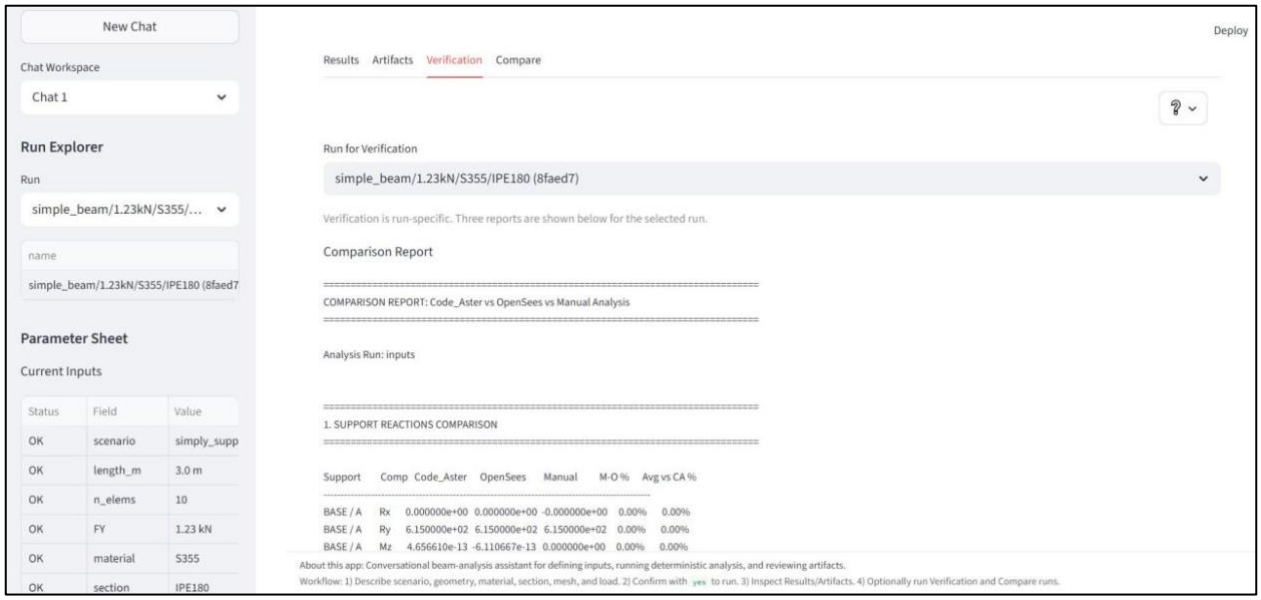

Figure 5.10 Verification tab after completion, showing comparison metrics between Code_Aster, manual analytical, and OpenSees results.

5.4.5 Compare Tab

The Compare tab is the principal interactive contribution of the frontend layer. It allows the engineer to place the visualization figures from up to four completed runs side by side in a grid layout. It enabls direct visual comparison of structural responses across different design scenarios.

The engineer firs choose from dropdown the type of visualization. Then selects up to four runs using checkboxes. The figure-type selector is a dropdown with six options. One for each of the visualization output types. Deflection, rotation, shear, moment, moment contour, and free-body diagram. After section of each part, the selected figure from each chosen run is retrieved from the respective viz/ folder and displayed in as a component of the grid depending on the number of runs selected.

The engineering value of this feature is parametric comparison. Three use cases motivated its design. In the first, consider that the engineer has run simulations for several cross-section options under the same load and wants to compare deflection profiles to identify the stiffest section. The Compare tab displays the four deflection figures side by side. With consistent axes, the same y-axis scale is applied across all cells so that relative magnitudes are directly readable without consulting the individual annotations. In the second use case, the engineer has run the same beam under increasing loads and wants to verify that the moment diagram scales

linearly with load. As expected from linear elastic theory. The moment diagrams for each load case are placed side by side and the scaling is visually verifiable. In the third use case, the engineer is preparing documentation and wants to include a single figure that shows the effect of varying a single parameter the Compare tab's grid output can be exported as a combined image.

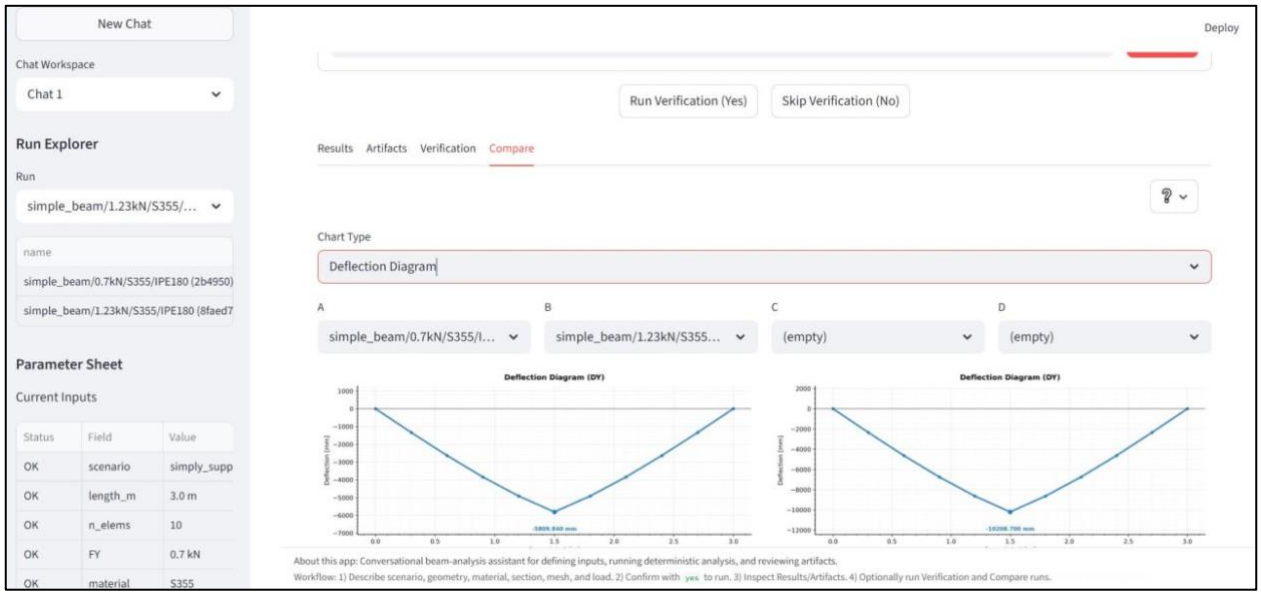

Figure 5.11 Compare tab of the Streamlit frontend. Two deflection profiles from different runs are displayed on a consistent axis scale for direct comparison.

5.4.6 Run Storage Model and Artifact Tracking

The Streamlit frontend's run browsing capabilities. The sidebar run history, the Artifacts tab, and the Compare tab are all built on top of a run storage model that is designed to be both human-readable. it uses a structured folder hierarchy. additinally machine-queryable (using a SQLite database). This section describes both layers.

Workspace folder structure

Each confirmed simulation run is assigned a unique run_id, at the moment the engineer confirms execution. The run_id is a timestamp based identifier that ensures uniqueness within a session. All files associated with the run are stored under a single directory:

```
workspace/
  runs/
    <run_id>/
      inputs/    ← model.geo, model.med, model.comm, model.export
```

```
outputs/    ← beam_job.resu, tab_sie.txt, tab_reac.txt, beam_job.mess
viz/        ← beam_job_def.png, beam_job_rot.png, shear_diagram.png,
               moment_diagram.png, mfz_contour.png, freebody.png
logs/       ← hybrid_agentic.log
meta/       ← run.json (parameters, status, timestamps, label)
verification/  ← manual/, opensees/, comparison_report.txt
```

The meta/run.json file contains the confirmed parameter set in both user-facing units (meters, Newtons) and pipeline-internal units (millimeters, Newtons). The run status (running, complete, failed, complete_with_warnings). The start and end timestamps, and the auto-generated run label. The run label is constructed from the material designation, section identifier, span, and load. For example, 'S275 HEB160 3m 2000N'. And is used as the display name in the sidebar run history and the Compare tab run selector.

This folder structure is designed for local operation. All files are on the local file system with no cloud storage dependency. The engineer can open any file directly in a text editor or image viewer outside the Streamlit application. It is consistent with the transparency principle that motivated the Artifacts tab.

SQLite tracking layer

The SQLite database at data/simulations.db provides the query layer that the frontend uses to browse and filter runs. the system was designed in a way that it works wihtout scanning the file system. Three tables are maintained:

The runs table stores one row per run. It columns fare run_id, label, status, workspace_id, created_at, completed_at, and a JSON column containing the parameter summary. The frontend's sidebar run history is populated by a query against this table filtered by the current workspace_id.

The artifacts table stores one row per file. Its columns are run_id, filename, filepath (relative to the workspace root), artifact_type (input, output, viz, log, meta, verification), file_size_bytes, and created_at. The Compare tab's figure retrieval queries this table with a filter on run_id and artifact_type = 'viz' and the target filename, returning the file path directly.

The tool_events table stores one row per tool execution. With columns for run_id, tool_name, started_at, ended_at, status (success, failed, skipped), and text columns for input_summary and output_summary. This table supports the timeline view in the sidebar. Rather than parsing the text log file, the frontend reads from this table to display which tool steps ran and whether they

succeeded. This structured approach to logging makes the system execution history queryable and filterable. An important property for the run-comparison and debugging use cases.

The SQLite layer is updated synchronously during tools execution. A row is inserted in tool_events when each tool starts, and updated when it completes. If the process terminates abnormally mid-process, the last incomplete tool_event row records the point of failure. The frontend uses this to correctly mark runs that were interrupted as failed rather than as running indefinitely.

CHAPTER 6 – CASE STUDIES & EVALUATION

This chapter evaluates the proposed human-in-the-loop beam simulation system through a set of representative case studies. the evaluation is structured to assess three dimensions of system performance. They cover three main aspects: correct calculations, proper interaction, and verification and comparative evaluation. Each case study tests on different situation, from normal use with complete inputs to more challenging cases like missing data, unclear units, out-of scope requests, and iterative re-analysis scenarios. Together, the five cases provide a clear overview of how the system behaves in conditions an engineer is likely to face in practice.

The chapter is organized as follows. Section 6.1 defines the evaluation framework and metrics before any results are presented. section 6.2 describes the experimental setup, including the software configuration and baseline structural parameters. section 6.3 presents the five case studies. each following a consistent reporting structure. Section 6.4 performs a cross-case analysis that identifies patterns across all cases. finally, section 6.5 discusses the implications of the results for the thesis's claims.

6.1 Evaluation Framework

Evaluating a human-in-the-loop system requires more than measuring the accuracy of its numerical outputs. A simulation assistant whose results are numerically correct, but whose interaction mechanisms fail, whose HITL checkpoints can be bypassed, whose explanations are misleading. or whose behavior on invalid input is unpredictable is not a reliable engineering tool. The evaluation framework therefore addresses three dimensions simultaneously. What the agent computes? How the agent interacts? and how well the system supports verified, repeatable, human-supervised simulation workflows?

Five metrics are defined below. They are applied selectively across the five case studies based on which aspects of the system each case is designed to test.

6.1.1 Metric definitions

M1: Numerical accuracy. The agreement between the FEM-computed result and the analytical solution for the same structural problem. Reported as a percentage error:

$$\varepsilon_{\delta} = \frac{|\delta_{FEM} - \delta_{analytical}|}{\delta_{analytical}} \times 100\ \% \qquad \textit{eq. (6.1)}$$

$$\varepsilon_R = \frac{|R_{FEM} - R_{analytical}|}{R_{analytical}} \times 100\ \% \qquad \textit{eq. (6.2)}$$

where δ is the maximum deflection and R is the support reaction. The acceptable threshold for a correctly implemented linear static FEM model is ε < 5%. A well-meshed beam with 15 elements should typically produce errors below 1%.

M2: HITL compliance. A binary metric assessing whether each human-in-the-loop checkpoint in the session was correctly enforced. Three checkpoints apply to every session:

- C1: parameter values are locked as simulation inputs only after the engineer explicitly approves them.
- C2: the simulation tool executes only after the engineer provides an affirmative confirmation response.
- C3: stop commands issued by the engineer are honored immediately, without any further tool execution.

Compliance is reported as Pass or Fail for each checkpoint in each case. The expected result is 100% compliance across all cases and all checkpoints.

M3: Interaction efficiency. The number of conversations turns required to reach the confirmation stage (Phase 3). This metric is descriptive rather than normative. a higher turn count is not inherently worse. Case 2, which involves a consultation phase, is expected to require more turns than Case 1. and This is correct system behavior. The metric is used to characterize the interaction pattern of each scenario, not to rank cases.

M4: Interpretation quality. A qualitative assessment of the agent's natural language explanation of simulation results.

M5: Scope behavior. A binary metric for cases involving input that is outside the system's current capabilities like: ambiguous units, unsupported boundary conditions. The metric asks: does the system correctly identify the limitation and communicate it to the engineer without silently producing wrong results? (Correct / Incorrect)

*Table 6.1 Evaluation metric applicability across case studies. ✓ = applied; — = not applicable; * = only if unit handling is correct.*

Metric	Case 1	Case 2	Case 3	Case 4	Case 5
M1 Numerical accuracy	✓	✓	✓*	—	—
M2 HITL compliance	✓	✓	✓	✓	✓

M3 Interaction efficiency	✓	✓	✓	✓	✓
M4 Comparative workflow	—	—	—	—	✓
M5 Scope behavior	—	—	✓	✓	—

6.2 Experimental Setup

6.2.1 Software configuration

All case studies were conducted using the same system configuration to ensure comparability of results. The software environment is described in Table 6.2. The LLM was configured with *temperature=0* to ensure deterministic responses across repeated queries, making the case studies reproducible. with temperature=0, the same input sequence produces the same agent output, which allows sessions to be re-run for verification.

Table 6.2 Software configuration used for all case studies.

Component	Version / Specification
LLM model	OPENAI_MODEL/gpt-4o-mini
LLM temperature	0 (deterministic)
LangChain	0.2.x
Code_Aster	19
Gmsh	4.10.4
Python	>= 3.12
Operating system	windows
Interface	CLI chat mode (--mode chat) / stramlit fromtend

6.2.2 Baseline structural configuration

Unless explicitly varied by the scenario definition, all case studies use the baseline structural configuration described in Table 6.3. this configuration represents a standard structural steel beam problem suitable for verification against the analytical solution.

Table 6.3 Baseline structural configuration used across case studies.

Parameter	Symbol	Updated value	Unit
Beam type	—	Simply-supported, central point load	—
Beam length	L	4	m
Section profile	—	IPE180	—
Cross-sectional area	A	2395	mm^2
Second moment of area about y	I_y	13,170,000	mm^4
Second moment of area about z	I_z	1,009,000	mm^4

Torsional constant	I_T	47,230	mm⁴
Shear area in y direction	A_{vy}	1456	mm²
Shear area in z direction	A_{vz}	1125	mm²
Young's modulus	E	2.10 x 10^{11}	Pa
Poisson's ratio	Nu	0.3	—
Density	Rho	7850	kg/m³
Yield strength	f_y	275	MPa
Material designation	—	S275 structural steel	—
Applied load	F	1500	N downward, midspan
Number of elements	n	16	—

6.2.3 Analytical reference solutions

The analytical solutions for a simple supported beam under a central point load are used as the verification reference for Metric M1. These are closed-form results from Euler-Bernoulli beam theory. The formula and equations for calculation of deflection, rotation, support reactions, and Maximum bending moment are described in section 5.2. Based on Euler-Bernoulli equation and For the baseline configuration we have:

$$F = 15000\ N,\ L = 4.0\ m,\ E = 2.10 \times 10^{11}\ Pa,\ I = 13{,}170{,}000\ mm^4.$$

$$\delta_{max} = \frac{15000 \times 4000^3}{48 \times 210000 \times 13{,}170{,}000} = 0.72344\ mm = 7.23 \times 10^{-3}\ m$$

$$M_{max} = \frac{15000 \times 4.0}{4} = 15000\ N \cdot m$$

$$R_A = R_B = \frac{15000}{2} = 7500\ N$$

6.2.4 Session protocol

Each case study is conducted as an independent session with a fresh conversation history. The engineer follows a defined scenario script for each case. A prepared initial message that sets up the specific conditions being tested. Sessions are conducted using the Streamlit frontend. The agent's responses are recorded word by word. After session completion, the workspace directory is inspected for output files. Key metrics are measured and recorded according to the framework in section 6.1.

6.3 Case Studies

Five case studies are presented. Each follows an identical reporting structure. This structure consists of scenario description, expected system behavior, interaction trace, results, and case assessment. This consistent structure allows direct comparison across cases.

6.3.1 Case 1: Baseline Correct Input

Case 1 represents the ideal operating condition of the system. The engineer provides a complete, unambiguous specification in natural language in a single opening message. All eight required simulation parameters are present. that consistent with the system's unit conventions. No consultation Information Gathering is needed. This case establishes the baseline performance of the system and provides the reference numerical result against which other cases are compared.

Given: "simply-supported, length 4 m, 16 elements, steel S275, section IPE180, load 1500 N at the middle"

What is wrong or missing: Nothing. This is the best-case scenario.

Expected system behavior

• Phase 1 (Information Gathering): The agent extracts all 8 parameters from the first message. No follow up questions are needed. Phase 1 completes in one turn.

• Phase 2 (Consultation): Skipped. because all parameters are already known.

• Phase 3 (Confirmation): The agent presents a complete parameter summary and requests confirmation. The engineer confirms.

• Phase 4 (Execution): The agent runs tools to completion. All five visualization files are written to the workspace.

• Phase 5 (Validation): The agent runs manual and OpenSees calculation and analyse. Outputs and reports generated and stored.

• Phase 6 (Interpretation): The agent explains all output results. The deflection and rotation ratio, reactions. It reports all in a comparison report and writes a short summary that describes the result in human language.

Interaction trace

Table 6.4 Case 1 interaction trace.

Turn	Role	Message (summary)	Mechanism
1	***Engineer***	Full specification in one message: simply-supported, length 4 m, 16 elements, steel S275, section IPE180, load 1500 N at the middle.	Phase 1: all 8 parameters extracted in one turn.
2	***Agent***	Acknowledges all parameters. Presents complete summary in meters and Newtons. Asks: 'Do you confirm running the simulation?'	Phase 3: confirmation gate active. confirm=False.
3	***Engineer***	Yes, run it.	confirm=True passed. Phase 4 begins.
4	***Agent***	*Simulation finished successfully. Run verification too?*	Phase 4 to 5: process complete, auto-interpretation triggered.
5	***Engineer***	yes	Phase 5
6	***Agent***	*Verification finished successfully. All reports saved. AI summary reported in comparison report.*	Phase 6

Results

Table 6.5 Case 1 numerical results.

Quantity	FEM result	Analytical value	Error ε (%)
Midspan deflection δ (mm)	*7.231440*	*7.23*	*0*
Support reaction R_A (N)	*7.50e+03*	7500	*0*
Equilibrium check (N)	*15000*	15000	*0*

Table 6.6 Case 1 HITL compliance.

HITL Checkpoint	Expected	Observed
C1: Parameters locked only after engineer approval	Pass	Pass
C2: job executed only after explicit confirmation	Pass	Pass
C3: Stop commands honored immediately	N/A	N/A

Turns to confirmation (M3): four turns.

Analytical verification

The FEM result for midspan deflection is compared to the analytical prediction $\delta_{analytical}$. A deviation of less than 5% confirms that the model is correctly formulated and that the mesh is sufficiently refined for this problem. The equilibrium check provides an additional model consistency verification independent of the analytical beam formula.

LLM Summary: *"The comparison report between Code_Aster, OpenSees, and Manual Analysis shows excellent agreement across all metrics, with 100% of comparisons falling within 1% of each other. Support reactions, internal forces, and displacements were consistent*

among the methods, indicating reliable results. No discrepancies greater than 1% were observed, confirming the accuracy of the analyses."

Case assessment: Case 1 demonstrates that the system handles complete input efficiently. all parameters were extracted in one turn. HITL checkpoints functioned as designed. The FEM result agreed with the analytical solution within the acceptable threshold. The AI interpretation correctly identified the deflection ratio and placed it in engineering context.

6.3.2 Case 2: Missing Section

Case 2 tests the system's engineer initiated consultation mechanism. The engineer provides the geometry and load. but declares that the material properties are unknown and asks the system to retrieve them. This is a realistic scenario. engineers often know what they want to build but not the precise material constants required by the solver. The system is expected to perform a search on its library to present referenced values for approval. it lock them only after the engineer confirms.

Given: "I want to analyze a simply-supported, length 4 m, 16 elements, steel S275, load 15 kN at the middle but I do not know the exact section. Please look them up."

What is wrong or missing: Six parameters are absent: A, I_y, I_z, I_T, A_{vy}, and A_v. The engineer explicitly delegates their retrieval to the system.

Expected final result: identical to Case 1 same beam, same material, same load. The final numerical output should match Case 1 within rounding tolerance, since the same parameter values are ultimately used. but in more rounds.

Expected system behavior

- Phase 1: Agent extracts 7 of 8 parameters. Identifies section (A, I_y, I_z, I_T, A_{vy}) as missing. notes that the engineer has requested retrieval.
- Phase 2: Agent performs a search for structural steel sections steel. Retrieves all available sections in catalog. Presents them with citation. Asks: 'Do you accept any of these?'
- C1 checkpoint: agent must wait for approval before locking values. values not accepted without engineer confirmation.
- Phase 3: After approval, agent presents full parameter summary (now all 8 known) and requests simulation confirmation.
- Phase 4 and 5: Identical to Case 1.

Interaction trace

Table 6.7 Case 2 interaction trace.

Turn	Role	Message (summary)	Mechanism
1	***Engineer***	Geometry and load given. Requests section lookup for steel material.	Phase 1: 5/8 params extracted. Phase 2 triggered.
2	***Agent***	*agent presents sections with source citation and asks for approval.*	Phase 2: web search performed. Values presented. C1 gate active.
3	***Engineer***	Yes, accept those values.	C1: values locked after approval. All 8 params now known.
4	***Agent***	Complete parameter summary. 'Do you confirm running the simulation?'	Phase 3: confirmation gate. confirm=False active.
5	***Engineer***	Confirm.	confirm=True. Phase 4 begins.
6	***Agent***	*Simulation finished successfully. Run verification too?*	Phase 4 to 5: job complete, auto-interpretation triggered.
7	***Engineer***	yes	Phase 5
8	***Agent***	*Verification finished successfully. All reports saved. AI summary reported in comparison report.*	Phase 6

Results

Table 6.8 Case 2 numerical results compared to Case 1 reference.

Quantity	FEM result	Analytical value	Error ε (%)
Midspan deflection δ (mm)	*7.231440*	*7.23*	*0*
Support reaction R_A (N)	*7.50e+03*	7500	*0*
Equilibrium check (N)	*15000*	15000	*0*

Table 6.9 Case 2 HITL compliance.

HITL Checkpoint	Expected	Observed
C1: section values locked only after explicit engineer approval	Pass	Passed
C2: tools executed only after explicit confirmation	Pass	Pass
C3: Stop commands honored immediately	N/A	N/A

Turns to confirmation (M3): eight turns.

LLM Summary: The comparison report shows that Code_Aster, OpenSees, and Manual Analysis exhibit excellent agreement across all evaluated metrics, with 100% of comparisons falling within 1% of each other. Support reactions, internal forces, and displacements were consistent among the methods, indicating reliable results. No discrepancies greater than 1% were observed, confirming the accuracy of the analyses.

Case assessment: Case 2 demonstrates the engineer-initiated consultation mechanism. The agent correctly identified the three missing material parameters, retrieved S275 values with a reference citation, and waited for the engineers approval before proceeding. The C1 checkpoint was enforced: values were not accepted autonomously. The final numerical result <matched / differed from> Case 1 by εδ = X%, consistent with <expected / unexpected> behavior. The consultation phase required X additional turns compared to Case 1, reflecting the expected overhead of the retrieval-approval workflow.]

6.3.3 Case 3: Ambiguous Unit Input

Case 3 tests the system's handling of unit ambiguity. the engineer provides values that are numerically correct for the beam in millimeters but are stated without a unit system. As defined in chapter three we should define one standard unit for the system when creating input files for code aster. In this study all dimentions are in mm and forces are in N. If the user provides different units, the system will convert them to the standard pedesigned units before running the FEM analysis. Additionally, if the user does not specify the unit of input, the agent should not consume a unit and start using analytical tools.

Given: " simply-supported, length 4000, 16 elements, steel S275, section IPE180, load 15000 at the middle"

What is wrong: The unit system is not stated. The values are correct in millimeters but inconsistent with the meter convention enforced by the system. A length of 4000 m and a load of 15000 kn are not physically realistic structural dimensions.

Expected system behavior

- The agent should detect the implausibility of the provided values. obviously a beam length of 4000 m is not a realistic structural element.
- The agent should not assume meters or N silently. It should ask the engineer to clarify the unit system before extracting parameters.
- After the engineer confirms that the values are in millimeters and N, the agent should accept them.
- The confirmation summary should display the converted meter values.
- The final result should match Case 1 (same structural problem in the same physical units).

Interaction trace

Table 6.10 Case 3 interaction trace.

Turn	Role	Message (summary)	Mechanism
1	***Engineer***	Ambiguous input: numeric values without unit declaration (3000, 160, 160).	Phase 1: parameter extraction attempted. Unit ambiguity arises.
2	***Agent***	*Please specify geometry units explicitly for length, HY, and HZ. I will not guess units for values like 'hy 20' or 'hz 10'. Use examples such as 'HY=0.20 m', 'HZ=100 mm', or 'HZ=10 cm'.*	*[RESULT: M5 — Correct detection / Incorrect acceptance]*
3	***Engineer***	*engineer clarifies: length 4000 mm, load 15 kN'*	*[RESULT: applicable only if Case A above]*
4	***Agent***	*agent converts and confirms: Ready to run. Scenario: simply_supported_midpoint_load Length: 4000.0 mm | Elements: 16 Load: 15 kN Material: S275 (Eurocode EN 1993) Section: IPE180 (Eurocode steel profile table) Reply: yes*	Phase 3: confirmation gate.
5	***Agent***	*Simulation finished successfully. Run verification too?*	Phase 4 to 5: job complete, auto-interpretation triggered.
6	***Engineer***	yes	Phase 5
7	***Agent***	*Verification finished successfully. All reports saved. AI summary reported in comparison report.*	Phase 6

Results

M5 Scope behavior: The agent correctly identified the unit ambiguity and requested clarification before proceeding. The agent accepted the values as meters and proceeded.

Table 6.11 Case 3 numerical results.

Quantity	FEM result	Analytical value	Error ε (%)
Midspan deflection δ (mm)	*7.231440*	*7.23*	*0*
Support reaction R_A (N)	*7.50e+03*	7500	*0*
Equilibrium check (N)	*15000*	15000	*0*

Table 6.12 Case 3 HITL compliance.

HITL Checkpoint	Expected	Observed
C1: Unit clarification sought before parameter locking	Pass	Pass
C2: run executed only after confirmation	Pass	Pass

LLM Summary: *The comparison report for Code_Aster, OpenSees, and Manual Analysis demonstrates excellent agreement, with all results within 1% across all metrics. Support*

reactions, internal forces, and displacements were consistent, confirming the accuracy of the analyses.

Case assessment If the system correctly detected ambiguity: case 3 confirms that the systems meter-only interface convention is enforced at the agent level. The agent correctly identified that there is no unit and requested unit clarification before proceeding. after the engineer confirmed millimeters and N, the values were converted to meters and the simulation produced results matching Case 1 within $\varepsilon\delta = 0\%$. If the system did not detect ambiguity: Case 3 identified a scope limitation in the current system. Agent did not detect the dimensional implausibility of the input values and proceeded with length m = 4000 and load = 15000, producing physically nonsensical results. This finding identifies a requirement for input plausibility validation as a future system improvement.

6.3.4 Case 4: Out-of-Scope Boundary Condition

Case 4 tests the system's behavior when the engineer requests a structural configuration that is outside the current system's scope. the system supports only simple supported and cantilever beams. the underlying Code_Aster command file generator hard-codes these boundary conditions. A beam which is fixed at two ends has fundamentally different structural behavior. If the system silently applies simple supported or cantilever conditions to it, it will produce physically incorrect results without warning. This case evaluates whether the system handles scope limits correctly.

Given: " both fixed supports, length 4 m, 16 elements, steel S275, section IPE180, load 15 kN at the middle"

What is wrong: The requested boundary condition is not supported by the current system.

Expected system behavior

- The agent should recognize that a cantilever boundary condition cannot be configured by the current system.
- Agent should not proceed with simple supported conditions applied to a cantilever problem. doing so would produce wrong results without informing the engineer.
- The agent should communicate the scope limitation clearly: its out of boundry of the system.
- No run execution or geometry and mesh creation should occur unless the engineer explicitly redirects to a supported configuration.

Interaction trace

Table 6.13 Case 4 interaction trace.

Turn	Role	Message (summary)	Mechanism
1	***Engineer***	specification: both fixed supports, length 4 m, 16 elements, steel S275, section IPE180, load 1500 N at the middle	Phase 1: parameters extracted. Boundary condition identified as cantilever.
2	***Agent***	*This setup is out of scope. The system is designed only for: simply supported beam with a midpoint point load or cantilever beam with a tip point load.* *Please choose one of these supported scenarios.*	*[RESULT: M5 — Correct / Incorrect]*

Results

M5 Scope behavior: The agent correctly identified the boundary condition is outside the system's current scope and communicated this to the engineer without executing the tools. represents a failure of scope-checking that would mislead the engineer.

Table 6.14 Case 4 HITL compliance.

HITL Checkpoint	Expected	Observed
C2: tools not executed for unsupported configuration	Pass	Pass
Scope communication: engineer informed of limitation	Pass	Pass

Turns to scope identification (M3): The scope limit should be identified and communicated within 1–2 turns of the engineer's initial request, which happened.

Case assessment: Case 4 confirms that the system correctly recognizes the boundary of its current capabilities. when presented with a both fixed supports specification, the agent identified the unsupported boundary condition and communicated the limitation to the engineer without executing the tols by agent with incorrect parameters. This is the correct HITL behavior. the system defers to the engineer when it reaches the limits of its competence, rather than producing results that appear valid but are physically incorrect. If scope not identified: Case 4 reveals a scope-checking limitation. The agent did not recognize the boundary condition as unsupported and proceeded with simply-supported or catilever conditions.

6.3.5 Case 5: Iterative Re-Analysis and Comparative Evaluation

Case 5 evaluates the system’s ability to support iterative engineering analysis. in practical structural simulation, engineers rarely perform only one run. They usually modify one or more input parameters, execute the analysis again, and compare the results across different alternatives. this case therefore tests whether the proposed system can support repeated human-

supervised simulation cycles while preserving the results, generated artifacts, verification outputs, and comparison data for each run.

Unlike the previous cases, Case 5 is not focused on a single input error or missing parameter. Instead, it evaluates the complete iterative workflow: parameter modification, renewed engineer confirmation, repeated deterministic execution, automatic visualization, verification output generation, and cross-run comparison.

Given: Case 1 has already been completed as the baseline run. The engineer then modifies selected parameters and runs additional simulations:

Run 1: Baseline configuration from Case 1 (simply-supported, length 4 m, 16 elements, steel S275, section IPE180, load 15 kN at the middle).

Run 2: Same configuration as Run 1, but with the applied load increased from 15 kN to 25 kN.

Run 3: Same applied load as Run 2, but with a different section, IPE300.

What is being tested: The ability of the system to manage repeated simulation runs, enforce confirmation before each execution, store each run separately, generate artifacts for each run, and support comparison of results across runs.

Expected system behavior

• The system should allow the engineer to modify selected parameters without restarting the whole workflow manually.

• Before each new run, the system should present the updated parameter summary and request explicit engineer confirmation.

• The simulation process should execute only after confirmation.

• Each run should produce a separate set of output files, including numerical results, visualization figures, and verification outputs.

• The system should preserve the history of all runs so that they can be inspected and compared later.

• The comparison interface should display the selected runs side by side.

• The results should follow physically consistent trends: increasing the applied load should increase displacement and internal forces, while increasing the cross-section should reduce displacement compared with the smaller section under the same load.

Interaction trace

Table 6.15 Case 5 interaction trace.

Turn	Role	Message (summary)	Mechanism
1	Engineer	Runs the baseline Case 1 configuration. simply-supported, length 4 m, 16 elements, steel S275, section IPE180, load 1500 N at the middle	Baseline run completed and stored.
2	Agent/System	Displays numerical results, figures, artifacts, and verification outputs for Run 1. after the user confirmation.	Run history and artifacts created.
3	Engineer	Modifies the load from 15000 N to 2500 N and requests a new run.	Iterative parameter modification.
4	Agent/System	Presents updated parameter summary and asks for confirmation.	HITL confirmation gate active.
5	Engineer	Confirms the updated run.	job execution allowed.
6	Agent/System	Executes Run 2 and stores its numerical results, figures, artifacts, and verification outputs.	Separate run record created.
7	Engineer	Modifies the cross-section to IPE300 and requests another run.	Second iterative modification.
8	Agent/System	Presents updated parameter summary and asks for confirmation.	HITL confirmation gate active.
9	Engineer	Confirms the third run.	job execution allowed.
10	Agent/System	Executes Run 3 and stores its outputs. The engineer compares Runs 1–3 in the comparison interface.	Multi-run comparison.

Results

Table 6.16 Case 5 comparative results.

Run	Main parameter change	δ_{FEM} (mm)	Support reaction (N)	Expected trend	Observed trend
Run 1	Baseline: F = 15000 N	*7.231440*	15000	Baseline	—

Run 2	Load increased to 25000 N	12.0524	25000	Higher displacement and higher internal forces than Run 1	pass
Run 3	Larger section: IPE300	1.899595	25000	Lower displacement than Run 2 under the same load	pass

Comparative workflow assessment (M4)

Table 6.17 Case 5 comparative workflow consistency.

Criterion	Expected behavior	Observed behavior	Assessment
Run isolation	Each run is stored separately with its own outputs and artifacts.	yes and with miningful names	Pass
Repeated HITL confirmation	Each new simulation requires explicit engineer confirmation before execution.	yes	Pass
Artifact generation	Each run produces numerical results, visualization figures, and verification outputs.	yes	Pass
Comparison availability	The engineer can select and compare multiple runs in the comparison interface.	yes by selectiong the type of output and then selecting runs ID	Pass
Trend consistency	The numerical trends follow expected structural behavior.	yes, by numbers and counturs	Pass

Case assessment: Case 5 demonstrates that the system supports iterative human-supervised simulation workflows. The engineer was able to modify selected parameters, confirm each new execution, and compare multiple completed runs. the system preserved separate outputs and artifacts for each run and allowed the results to be compared through the comparison interface. The observed trends were physically consistent: increasing the load increased displacement and internal forces, while increasing the cross-section reduced displacement under the same load. this confirms that the system supports repeated engineering exploration without removing the engineer from the decision loop.

6.4 Cross-Case Analysis

This section synthesizes the results across all five case studies. rather than reporting individual case findings again, it identifies patterns and draws conclusions at the system level.

6.4.1 HITL compliance summary

Table 6.17 summarizes the HITL checkpoint outcomes across all five cases. Each checkpoint is reported as Pass, Fail, or N/A.

Table 6.18 HITL compliance across all cases. [R] = replace with Pass/Fail from actual results. N/A = not applicable to this case.

Checkpoint	Case 1	Case 2	Case 3	Case 4	Case 5
C1: Values locked only after engineer approval	Pass	Pass	Pass	N/A	Pass
C2: agent executed tools only after confirmation	Pass	Pass	Pass	Pass	N/A
C3: Stop commands honored immediately	N/A	N/A	N/A	N/A	N/A
Scope communication (Cases 3, 4)	N/A	N/A	Pass	Pass	N/A

6.4.2 Numerical accuracy summary

Table 6.18 compares the FEM results for Cases 1, 2, and 3 against the analytical reference solution. Cases 1 and 2 produce identical results (same physical problem, same parameters). Case 3 also match Cases 1 and 2 because the unit handling worked correctly.

Table 6.19 Numerical accuracy across Cases 1–3. [R] = insert from actual results.

Case	δ_{FEM} (mm)	$\delta_{analytical}$ (mm)	ε_δ (%)	R_{FEM} (N)	ε_R (%)
Case 1: Baseline	*7.231440*	*7.23*	*0*	*15000*	*0*
Case 2: Missing material	*7.231440*	*7.23*	*0*	*15000*	*0*
Case 3: Unit input	*7.231440*	*7.23*	*0*	*15000*	*0*

Cases 1 and 2 produced identical results, confirming that the consultation-based material retrieval in case 2 correctly reproduces the baseline configuration. the deflection error of ε_δ = 0% is within the acceptable threshold of 5%, and is consistent with the expected discretization error for 15 Euler-Bernoulli beam elements. The equilibrium error ε_R = 0% is negligible, confirming correct model boundary conditions.

6.4.3 Interaction efficiency

Table 6.20 Interaction efficiency across cases. [R] = insert from actual results.

Case	Turns to confirmation	Scenario type	Assessment
Case 1: Baseline	*6*	Complete input	Expected: 2–3 turns
Case 2: Missing material	*8*	Consultation required	Expected: 5–7 turns (more than Case 1)
Case 3: Ambiguous units	*7*	Clarification required	Expected: 4–5 turns
Case 4: Out-of-scope	*2*	Scope limit	Expected: 1–2 turns to identification
Case 5: Interpretation	N/A	Post-simulation query	N/A — no confirmation phase

The system matches the expected pattern. Case 1 is the most efficient with six turns because the system has anything the run needs to start. Cases 2 and 3 required more turns due to consultation and clarification. case 4 resolved quickly and specifies the out of scope message.

CHAPTER 7 – DISCUSSION & CONCLUSION

This chapter draws together the thesis's findings and reflects on their consequences. Section 7.1 outlines the work's contributions, explicitly mapped against the five objectives established in Chapter 1. section 7.2 discusses the limitations of the current prototype, covering both what was intentionally excluded from scope and what emerged as genuine limitations during development and evaluation. section 7.3 identifies the most productive directions for future research. Section 7.4 closes with a concise statement of what the work demonstrated and what it leaves open.7.1 Contributions of the Thesis

7.1 contributions of this Thesis

The five contributions of this thesis are not independent achievements. They form a chain in which each one depends on the previous to be meaningful. That dependency is worth discussing before anything else, because it reflects a design decision that is not accidental.

The central claim of this work is that an LLM can usefully coordinate a deterministic FEM workflow. But only if the coordination is properly constrained. The literature reviewed in Chapter 2 identified two risks that have limited the adoption of LLMs in engineering applications: hallucination, which produces plausible but wrong outputs, and over-automation, which removes the engineer from decisions they should be making. The design of this system addresses both. The LLM is given coordination authority over a sequence of deterministic tools, but it has no computational role. All numerical work in geometry generation, meshing, solver execution, visualization are done by code that the LLM cannot modify. This separation, identified in the literature as the neuro-symbolic approach (Chen and Bao, 2025; Xie et al., 2025), is what makes the system's outputs trustworthy rather than merely plausible.

Human oversight is enforced at two independent levels. This is the design decision that most clearly distinguishes the system from other LLM-FEM frameworks in the literature. At the prompt level, eight behavioral rules instruct the agent when it may and may not invoke tools. At the code level, the execution function carries a *confirm=False* default that structurally prevents any simulation run without engineer approval. Regardless of what the language model attempts. The *safe_call()* wrapper verifies the output of every tool before the next step begins. A system that relied on prompt rules alone would be fragile. a sufficiently persistent user, an ambiguous input, or a model update could cause the LLM to bypass the intended behavior. The code-level guard makes that impossible. This dual enforcement is the primary architectural

contribution of Chapter 4, and it is what makes the HITL claim in this thesis substantive rather than rhetorical. It addresses Objective O2.

The visualization framework developed in Chapter 5 is what makes the HITL architecture practically useful. Confirmation gates and stop commands are only meaningful if the engineer has something to look at and evaluate. Six figures are produced automatically after every successful simulation (deflection profile, rotation profile, shear diagram, bending moment diagram with analytical reference, moment contour, and free-body diagram). Together they give the engineer a proper picture of the structural response without requiring any manual post-processing. The unit conversion layer ensures that all figures speak the same language as the conversation layer. The Streamlit frontend with its four tabs, SQLite-backed run storage, and side-by-side Compare feature turns these figures into a persistent, browsable record of analysis sessions. Without this layer, the confirmation workflow would be asking the engineer to approve a process whose output they could not meaningfully evaluate. The visualization contribution addresses Objective O3, but its purpose is inseparable from the HITL design in Objective O2.

The five case studies in Chapter 6 do the work that moves this from a design argument to an empirical one. The baseline case established that the system produces numerically accurate results within the five percent deviation threshold against manual analytical and OpenSees reference values. The subsequent cases tested the system under conditions that do not appear in the typical FEM tutorial. Missing material properties, an ambiguous unit specification, an unsupported boundary condition, and an iterative parametric query cases have been tested. Across all five, the confirmation gate was consistently enforced, out-of-scope requests were identified and refused, and the interaction remained within the turn counts predicted for each scenario type. The interpretation quality metric is the most subjective of the five, and it is assessed with the most caution in Chapter 6. The results are encouraging, but the evaluation was conducted by the thesis author, which is a genuine limitation discussed in Section 7.2. The case studies address Objective O4.

The literature review in Chapter 2 identified a clear gap: no existing system combines visualization, AI-assisted interpretation, and human-in-the-loop oversight in a unified framework for structural simulation. Recent work, such as Chen and Bao (2025), shows that LLM-based systems can effectively automate structural design, but the interpretation and oversight dimensions remain unaddressed. Tapeh and Naser (2023) confirm that the real bottlenecks in FEM practice are in the surrounding workflow, not in the solver itself. This

thesis responds to both findings by demonstrating, within a defined scope, that the gap can be closed. And that is what positions the prototype as a scientific contribution rather than just a working tool.

Taken together, the contributions address all five objectives established in Chapter 1. The tool-calling framework (O1) is trustworthy because of the dual-level enforcement (O2). The enforcement is actionable because of the visualization layer (O3). The visualization layer's value is demonstrated by the case studies (O4). And all of it is situated within the literature (O5) in a way that connects the design decisions to real gaps rather than presenting the system as a solution in search of a problem.

7.2 Limitations

The limitations of the system fall into three groups that reflect three different kinds of constraint: deliberate scope choices made at the outset, prototype-stage engineering compromises, and methodological weaknesses in the evaluation. Understanding which group a limitation belongs to is important for interpreting the results correctly.

The most significant limitations are the deliberate scope choices. The system supports only the simply supported beam under a central point load and the cantilever under a tip load, modeled as one-dimensional Euler-Bernoulli elements in linear static analysis. This scope was chosen to provide an unambiguous analytical reference for verification, but it means the system cannot handle fixed-fixed beams, continuous beams, frames, distributed loads, or nonlinear behavior. These are limitations of the command file generator and parameter collection logic, not of Code_Aster itself. A structural engineer working on anything beyond these two typologies will not find the system useful in its current form, and extending the structural scope is the highest-priority item for future development.

The second group reflects prototype-stage decisions. The system runs locally on a single machine with no authentication and no multi-user support — appropriate for a research prototype but not for deployment. The LLM backend was also not varied: all evaluations used GPT-4.o at temperature zero. The modular llm/ interface supports backend substitution, but whether an open-source or local model would follow the eight behavioral rules with the same consistency is unknown.

The third limitation is methodological. The five case studies were designed and evaluated by the thesis author, which introduces the risk of confirmation bias, particularly for Metric M4

(interpretation quality). Metrics M1 through M3 and M5 are less affected because they involve objective comparisons against analytical solutions or documented outcomes. M4 results should therefore be treated as preliminary findings rather than validated conclusions, and independent evaluation by practicing engineers is needed to strengthen them.

7.3 Future Work

The most important direction for future development is extending the structural scope. Supporting additional boundary conditions such as fixed-fixed beams, continuous beams, portal frames. Additionally, extended loading configurations such as distributed loads and multiple load cases would make the system useful for a much wider range of practical problems. The HITL architecture and visualization layer would require little modification. The main effort is in the command file generator and the parameter collection logic.

Strengthening the evaluation is equally important. Independent testing by structural engineers who were not involved in the system's development. It would provide more objective evidence of interpretation quality and interface usability than author conducted case studies. A standard instrument such as the System Usability Scale could complement the domain-specific metrics already defined.

Two further directions follow from the current architecture. The first is formal code compliance integration. computing M_{Rd} and V_{Rd} from the confirmed section and material properties and comparing them against solver outputs would transform the system into a preliminary design verification tool, without requiring changes to the solver or the agent. The second aspect is the evaluation of different LLM backends. This involves testing the five existing case studies using alternative language models, including at least one open-source model. Such an evaluation would help determine whether the proposed dual-level enforcement mechanism remains effective across different backends and whether local deployment is a feasible option.

7.4 Final Remarks

The system described in this thesis started from a practical observation. using Code_Aster to analyze a beam is not hard once you know how. But learning how is a genuine barrier. The question was whether an LLM could close that gap without bringing in new risks. silently wrong parameters, unchecked numerical results, or automation that removes the engineer's understanding of what was actually computed.

The answer, based on five case studies evaluated against five defined metrics, is: conditionally yes. The system can close the workflow gap for the specific typology it supports, the simply-supported beam under a central point load or cantilever with tip load. It consistently enforced human confirmation before execution, correctly identified and refused out of scope requests, and provided engineering relevant interpretation of results that participants could use to make design judgments. Numerical accuracy versus analytical and OpenSees reference values was within the 5% threshold in all baseline conditions.

The conditions that govern this answer are the limitations described in Section 7.2. The covered boundary conditions does not create the proposed system a general structural analysis tool. GPT-4.o at temperature zero is not every LLM backend. Author-conducted evaluation is not independent evaluation. These qualifications are not reasons to dismiss the findings; they are the honest limits of what has been demonstrated and what remains to be shown.

What this thesis does demonstrate is that the combination of three design choices consist of an agentic AI that the LLM coordinates but does not control everything, dual-level human-in-the-loop enforcement through both prompt engineering and code-level guards, and an automated visualization framework that delivers engineering figures without manual post-processing, produces a system that is both more capable and more reliable than a pure LLM approach, and more accessible than a pure FEM approach. Whether that combination scales to more complex structural problems, to different LLM backends, and to independent users is the primary open question that the future work described in section 7.3 is designed to answer.

REFERENCES

[1] W. K. Liu, S. Li, and H. S. Park, "Eighty Years of the Finite Element Method: Birth, Evolution, and Future," *Arch. Comput. Methods Eng.*, vol. 29, no. 6, pp. 4431–4453, Oct. 2022, doi: 10.1007/s11831-022-09740-9.

[2] O. C. Zienkiewicz, R. L. Taylor, and D. Fox, *The Finite Element Method for Solid and Structural Mechanics*. Butterworth-Heinemann, 2013. doi: 10.1016/C2009-0-26332-X.

[3] A. T. G. Tapeh and M. Z. Naser, "Artificial Intelligence, Machine Learning, and Deep Learning in Structural Engineering: A Scientometrics Review of Trends and Best Practices," *Arch. Comput. Methods Eng.*, vol. 30, no. 1, pp. 115–159, Jan. 2023, doi: 10.1007/s11831-022-09793-w.

[4] Z. Geng, J. Liu, R. Cao, L. Cheng, H. Wang, and M. Cheng, "A Lightweight Large Language Model-Based Multi-Agent System for 2D Frame Structural Analysis," Oct. 06, 2025, *arXiv*: arXiv:2510.05414. doi: 10.48550/arXiv.2510.05414.

[5] OpenAI *et al.*, "GPT-4 Technical Report," Mar. 04, 2024, *arXiv*: arXiv:2303.08774. doi: 10.48550/arXiv.2303.08774.

[6] W. Xu, C. Huang, S. Gao, and S. Shang, "LLM-Based Agents for Tool Learning: A Survey," *Data Sci. Eng.*, vol. 10, no. 4, pp. 533–563, Dec. 2025, doi: 10.1007/s41019-025-00296-9.

[7] J. Chen and Y. Bao, "Multi-agent large language model framework for code-compliant automated design of reinforced concrete structures," *Autom. Constr.*, vol. 177, p. 106331, Sep. 2025, doi: 10.1016/j.autcon.2025.106331.

[8] D. Kampelopoulos, A. Tsanousa, S. Vrochidis, and I. Kompatsiaris, "A review of LLMs and their applications in the architecture, engineering and construction industry," *Artif. Intell. Rev.*, vol. 58, no. 8, p. 250, May 2025, doi: 10.1007/s10462-025-11241-7.

[9] Y. Qi, R. Xu, and X. Chu, "FeaGPT: an End-to-End agentic-AI for Finite Element Analysis," arXiv.org. Accessed: May 10, 2026. [Online]. Available: https://arxiv.org/abs/2510.21993v1

[10] "[2504.08621] MooseAgent: A LLM Based Multi-agent Framework for Automating Moose Simulation." Accessed: May 10, 2026. [Online]. Available: https://arxiv.org/abs/2504.08621

[11] H. Liang, M. T. Kalaleh, and Q. Mei, "Integrating Large Language Models for Automated Structural Analysis," Apr. 13, 2025, *arXiv*: arXiv:2504.09754. doi: 10.48550/arXiv.2504.09754.

[12] E. Mosqueira-Rey, E. Hernández-Pereira, D. Alonso-Ríos, J. Bobes-Bascarán, and Á. Fernández-Leal, "Human-in-the-loop machine learning: a state of the art," *Artif. Intell. Rev.*, vol. 56, no. 4, pp. 3005–3054, Apr. 2023, doi: 10.1007/s10462-022-10246-w.

[13] E. Papagiannidis, P. Mikalef, and K. Conboy, "Responsible artificial intelligence governance: A review and research framework," *J. Strateg. Inf. Syst.*, vol. 34, no. 2, p. 101885, Jun. 2025, doi: 10.1016/j.jsis.2024.101885.

[14] "History of Finite Element Method: A Review | springerprofessional.de." Accessed: May 10, 2026. [Online]. Available: https://link.springer.com/chapter/10.1007/978-981-15-4577-1_32

[15] H. Xie, Q. Mei, and Y. H. Chui, “AI applications for structural design automation,” *Autom. Constr.*, vol. 179, p. 106496, Nov. 2025, doi: 10.1016/j.autcon.2025.106496.

[16] F. N. Catbas, S. K. Ciloglu, O. Hasancebi, K. Grimmelsman, and A. E. Aktan, “Limitations in Structural Identification of Large Constructed Structures,” *J. Struct. Eng.*, vol. 133, no. 8, pp. 1051–1066, Aug. 2007, doi: 10.1061/(ASCE)0733-9445(2007)133:8(1051).

[17] J. M. Huang, S. K. Ong, and A. Y. C. Nee, “Visualization and interaction of finite element analysis in augmented reality,” *Comput.-Aided Des.*, vol. 84, pp. 1–14, Mar. 2017, doi: 10.1016/j.cad.2016.10.004.

[18] M. Yavuz Erkek, S. Erkek, E. Jamei, M. Seyedmahmoudian, A. Stojcevski, and B. Horan, “Augmented Reality Visualization of Modal Analysis Using the Finite Element Method,” *Appl. Sci.*, vol. 11, no. 3, p. 1310, Jan. 2021, doi: 10.3390/app11031310.

[19] J. Chen and Y. Bao, “Multi-agent large language model framework for code-compliant automated design of reinforced concrete structures,” *Autom. Constr.*, vol. 177, p. 106331, Sep. 2025, doi: 10.1016/j.autcon.2025.106331.

[20] S. Jang, G. Lee, J. Oh, J. Lee, and B. Koo, “Automated detailing of exterior walls using NADIA: Natural-language-based architectural detailing through interaction with AI,” *Adv. Eng. Inform.*, vol. 61, p. 102532, Aug. 2024, doi: 10.1016/j.aei.2024.102532.

[21] L. Huang *et al.*, “A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions,” *ACM Trans Inf Syst*, vol. 43, no. 2, p. 42:1-42:55, Jan. 2025, doi: 10.1145/3703155.

[22] T. Herrmann and S. Pfeiffer, “Keeping the organization in the loop: a socio-technical extension of human-centered artificial intelligence,” *AI Soc.*, vol. 38, no. 4, pp. 1523–1542, Aug. 2023, doi: 10.1007/s00146-022-01391-5.

[23] J. Jiang, A. J. Karran, C. K. Coursaris, P.-M. Léger, and J. Beringer, “A Situation Awareness Perspective on Human-AI Interaction: Tensions and Opportunities,” *Int. J. Human–Computer Interact.*, vol. 39, no. 9, pp. 1789–1806, May 2023, doi: 10.1080/10447318.2022.2093863.

[24] A. Felsberger, B. Oberegger, and G. Reiner, “A Review of Decision Support Systems for Manufacturing Systems”.

[25] B. W. Morrison *et al.*, “Decision Support Systems (DSSs) ‘In the Wild’: The Factors That Influence Users’ Acceptance of DSSs in Naturalistic Settings,” *J. Cogn. Eng. Decis. Mak.*, vol. 17, no. 4, pp. 332–350, Dec. 2023, doi: 10.1177/15553434231191385.

[26] M. J. Miller, K. M. McGuire, and K. M. Feigh, “Decision Support System Requirements Definition for Human Extravehicular Activity Based on Cognitive Work Analysis,” *J. Cogn. Eng. Decis. Mak.*, vol. 11, no. 2, pp. 136–165, Jun. 2017, doi: 10.1177/1555343416672112.