A Case Study in Accelerated Engineering Software Development Using Large Language Model Collaboration

Mounir Jamiv, P.E.
Structural Engineering & Seismic Isolation Specialist

Claude (Anthropic)
AI Implementation & Software Development Agent

FEBRUARY 2026

IsoVis — Seismic Isolation Visualization & Analysis Platform
Technical Paper • Version 1.0

Abstract

This paper presents a case study in hybrid AI-human software development, documenting the creation of ***IsoVis****—a comprehensive, web-based seismic isolation analysis platform built in 13 calendar days through collaboration between a licensed structural engineer and a large language model (LLM) agent. The platform encompasses 72,500 lines of code spanning a React/Three.js frontend, a Python/FastAPI/OpenSeesPy backend, four structural analysis types (static, modal, time-history, and pushover), Triple Friction Pendulum bearing simulation, ASCE 7-22 and AASHTO code compliance automation, and 500+ automated tests. We analyze the division of labor between human domain expertise and AI code generation, quantify the development acceleration factor (25–40× compared to conventional approaches), and discuss implications for the future of specialized engineering software development. The results demonstrate that the hybrid model is most effective when irreplaceable human judgment—domain knowledge, physics intuition, and quality arbitration—is paired with AI capabilities in rapid code synthesis, cross-stack fluency, and parallel documentation generation.*

Keywords: seismic isolation, Triple Friction Pendulum, AI-assisted development, large language models, structural analysis, OpenSeesPy, software engineering, hybrid workflow, ASCE 7-22, AASHTO

1. Introduction

1.1 Motivation

Seismic base isolation is one of the most effective technologies available for protecting essential facilities—hospitals, emergency operations centers, and critical infrastructure—during major earthquakes. By decoupling a structure from ground shaking through specialized bearings, isolation systems routinely achieve 40–70% reductions in base shear and 80–95% reductions in inter-story drift, upgrading structural performance from "Life Safety" to "Immediate Occupancy" under design-level events.

Despite these benefits, the adoption of seismic isolation has been hampered by the complexity of the analysis and design workflow. Engineers must navigate nonlinear bearing mechanics, velocity-dependent friction models, multi-directional ground motion inputs, property modification factors for aging and contamination, and compliance requirements spanning ASCE 7-22 Chapter 17 and AASHTO Guide Specifications for Seismic Isolation Design. The commercial software tools capable of handling this workflow are expensive, require steep learning curves, and produce outputs that are difficult to visualize and communicate to stakeholders.

This paper documents the development of **IsoVis**, an open-source platform designed to make seismic isolation analysis accessible, visual, and code-compliant. More significantly, it examines *how* IsoVis was built: through a hybrid workflow pairing a structural engineer with an AI agent, compressing what would conventionally require 12–20 person-months into 13 calendar days.

1.2 The Hybrid Development Thesis

Large language models (LLMs) have demonstrated remarkable capability in generating syntactically correct, functionally viable code across multiple programming languages and frameworks. However, generating code that is *physically correct*—that faithfully represents structural mechanics, material behavior, and code-mandated design criteria—requires domain expertise that no language model possesses intrinsically. The hypothesis underlying this project was that a carefully structured collaboration could exploit the complementary strengths of each participant:

- **Human contribution:** Domain knowledge (structural engineering theory, TFP bearing physics, seismic code requirements), architectural vision, quality arbitration, and physics validation.
- **AI contribution:** Rapid code synthesis across all stack layers (frontend, backend, solver, tests, documentation), cross-framework fluency (React, Three.js, FastAPI, OpenSeesPy simultaneously), and tireless execution of iterative debug-fix cycles.

This paper evaluates whether this hypothesis held, what the actual division of labor looked like, and what lessons can be extracted for future projects at the intersection of AI and specialized engineering.

1.3 Scope of This Paper

Section 2 describes the platform architecture and engineering capabilities. Section 3 details the development timeline and methodology. Section 4 presents the quantitative effort analysis and compression ratio. Section 5 analyzes the roles and failure modes in the hybrid workflow. Section 6 discusses implications and limitations. Section 7 offers conclusions and recommendations.

2. Platform Architecture & Engineering Capabilities

2.1 System Overview

IsoVis is a full-stack web application comprising a React/TypeScript frontend with real-time Three.js 3D visualization, a Python/FastAPI backend wrapping the OpenSeesPy finite element solver, and a Redis layer for caching and asynchronous task management. The system is containerized via Docker Compose for reproducible deployment.

Table 1. Technology Stack Summary

Layer	Technology	Purpose
Frontend UI	React 18, TypeScript, Radix UI, Tailwind CSS	Model editor, results panels, analysis dialogs
3D Visualization	React Three Fiber, Drei	Interactive structural model rendering, mode shapes, deformed shapes, force diagrams
State Management	Zustand	Model store, display store, analysis store, comparison store
Charts	Plotly.js	Capacity curves, hysteresis loops, response spectra
API Server	FastAPI, Pydantic v2	REST endpoints, request validation, CORS
Solver Engine	OpenSeesPy	Finite element analysis (static, modal, transient, pushover)
Numerical	NumPy, SciPy	Matrix operations, eigenvalue decomposition, interpolation
Caching	Redis	Session result cache, async task queue
Infrastructure	Docker Compose, Vite, Vitest	Containerization, build tooling, test runner

2.2 Structural Analysis Capabilities

The platform supports four analysis types, each implemented as a distinct solver pathway within the OpenSeesPy backend:

Static Analysis. Linear elastic analysis under gravity and applied loads. The solver applies loads in 50 incremental steps with sub-stepping fallback and multi-algorithm cascading (Newton → ModifiedNewton → KrylovNewton) for convergence robustness. Results include nodal displacements, support reactions, and element end forces.

Modal Analysis. Eigenvalue analysis computing natural periods, frequencies, mode shapes, and effective modal mass participation ratios. The frontend renders animated mode shapes with sinusoidal oscillation in 3D.

Time-History Analysis. Nonlinear dynamic analysis with multi-directional ground motion input (X, Y, Z independently scaled). Includes four built-in synthetic ground motions (El Centro 1940, Near-Fault Pulse, Harmonic Sweep, Long-Duration Subduction). For models with TFP bearings, automatic Z-up coordinate conversion is performed as required by the TripleFrictionPendulum element formulation. Time-history playback includes speed control (0.25×–4×) and scrubbing.

Pushover Analysis. Displacement-controlled nonlinear static analysis with linear or first-mode-proportional load patterns. Generates capacity curves (base shear vs. roof displacement), tracks plastic hinge formation with seven performance states (elastic through collapse), and computes ductility ratios and overstrength factors.

2.3 Triple Friction Pendulum Bearing Model

The platform implements a full four-surface TFP bearing model, the most advanced commercially available isolation device. Each bearing is defined by:

- **Four friction surfaces** with independently configurable friction models (constant Coulomb, velocity-dependent, or velocity-pressure-dependent)
- **Effective pendulum radii** for inner (low-friction) and outer (high-friction) surface pairs
- **Displacement capacities** per surface, enabling demand/capacity ratio checks
- **Property modification factors** (λ_{min} / λ_{max}) for ASCE 7-22 bounding analysis

The solver constructs OpenSeesPy `TripleFrictionPendulum` elements with appropriate friction material models, vertical load initialization, and geometric transformation handling. Bearing response is tracked as displacement orbits, force-displacement hysteresis loops, and per-surface activation histories.

2.4 Comparison Framework

A distinguishing feature of IsoVis is the automated comparison of isolated versus fixed-base designs. Given an isolated model, the platform automatically derives a fixed-base variant by removing bearings and fixing base nodes, then executes the selected analysis type on both variants. The comparison dashboard presents:

- Capacity curve overlays (nominal + upper/lower λ-bounds + fixed-base)
- Base shear reduction percentage
- Inter-story drift profiles (side-by-side horizontal bar chart)
- Bearing demand/capacity ratios (color-coded green/yellow/red)
- Plastic hinge distribution (IO/LS/CP counts)
- Deformed shape overlay (blue for isolated, orange for fixed-base)

2.5 Code Compliance Automation

The platform automates 12 design checks per ASCE 7-22 Chapter 17 and AASHTO Guide Specifications for Seismic Isolation Design, 4th Edition. These include minimum restoring force, displacement capacity adequacy, effective period ratio (≥3× fixed-base), property modification factor bounding, vertical load stability, drift limits, MCE-level performance, redundancy/configuration, and energy dissipation verification. All checks are compiled into a self-contained HTML engineering report suitable for permitting.

2.6 Parametric Model Generators

Two parametric generators accelerate model creation:

Bay Build generates building frames with 1–5 bays, 1–10 stories, configurable span widths and story heights, steel or concrete material selection, auto-sized sections, and optional TFP bearing placement at every column. Accompanied by 67 unit tests.

Bent Build generates multi-span girder bridges with 1–8 spans, 3–10 girders, per-pier bent column configurations, steel or concrete deck options, conventional or isolated support modes, and AASHTO dead/live load generation. Accompanied by 113 unit tests.

3. Development Timeline & Methodology

3.1 Phased Approach

Development followed a five-phase plan executed over 13 calendar days (February 13–25, 2026), with 8 active coding days:

Phase 1: Scaffolding & 3D Viewer Feb 13

Project initialization, React/Three.js frontend scaffolding, FastAPI backend, model editor with node/element/section/material CRUD, property inspector, 3D viewer with node points, member lines, support symbols, and interactive labels.

Phase 2: Analysis Runner & Results Feb 13

Load editing UI, analysis dialog, API client, OpenSeesPy solver integration for static and modal analysis, results panels with tabular and chart displays.

Phase 3: TFP Bearing Model Feb 13–14

Four-surface friction model with velocity-dependent friction, bearing CRUD UI, 3D purple cylindrical symbols, serialization/deserialization, bearing connectivity and tributary weight assignment.

Phase 4: Pushover & Time-History Feb 14

Pushover analysis with displacement control, capacity curves, plastic hinge tracking. Time-history analysis with ground motion input, playback controls, speed adjustment. Mode shape animation, deformed shape overlay.

Phase 5: Comparison & Compliance Feb 14–15

Isolated vs. fixed-base comparison framework, lambda-factor bounding analysis, summary dashboards, ASCE 7-22 / AASHTO compliance checks, engineering report generation.

Hardening & Polish Feb 15–25

Security review, accessibility improvements, solver hardening (gravity preload, sparse solver, sub-stepping), 5:1 member discretization, force diagram rendering, parametric generators (Bay Build, Bent Build), 13 preset models, 500+ test suite, integration tests, engineering documentation.

3.2 Daily Commit Distribution

Table 2. Commits per Active Development Day

Date	Commits	Primary Activity
Feb 13	15	Phases 1–3 (scaffolding through TFP bearings)
Feb 14	4	Phases 4–5 (pushover, time-history, comparison)
Feb 15	1	Security hardening, theme, accessibility
Feb 16	1	Engineering report & deliverables
Feb 22	3	Solver hardening, 3D force diagrams, integration tests
Feb 23	2	Discretization, Hermite interpolation, force rendering
Feb 24	13	Parametric generators, bridge models, diaphragms
Feb 25	2	Deck diaphragms, solver merge, final refinements

The bimodal distribution—two intense sprint days (15 and 13 commits) bookending a refinement period—reflects the hybrid workflow's characteristic pattern: rapid feature generation followed by human-directed quality iteration.

3.3 Workflow Mechanics

Each development cycle followed a consistent pattern:

1. **Human directive:** The engineer specified the next feature or fix in natural language, often with domain-specific constraints (e.g., "TFP bearings require Z-up convention; swap Y and Z in the coordinate transform").
2. **AI implementation:** The LLM agent generated code across all affected files—frontend components, backend routes, solver logic, Pydantic schemas, TypeScript types, and tests—in a single pass.
3. **Human review:** The engineer evaluated outputs for physical correctness, catching errors that would be invisible to syntactic validation (e.g., incorrect friction coefficient ranges, wrong sign conventions in force diagrams, misapplied geometric transformations).

4. **Iterative refinement:** Debug-fix cycles were executed rapidly, with the engineer diagnosing root causes and the AI implementing corrections across the full stack.

4. Quantitative Effort Analysis

4.1 Deliverable Scope

72,500
TOTAL LINES OF CODE

150
SOURCE FILES

500+
AUTOMATED TESTS

41
GIT COMMITS

Table 3. Codebase Composition by Category

Category	Lines	Files	Description
Frontend (TS/TSX)	24,080	118	React components, Three.js visualization, Zustand stores, API client
Backend (Python)	6,373	25	FastAPI routes, OpenSeesPy solver, Pydantic schemas
Test Suite	9,497	29	Vitest (frontend), Pytest (backend), integration tests
Model Presets (JSON)	15,146	13	Pre-built structural models (towers, frames, bridges)
Engineering Docs	3,923	3	Compliance report, calculations, HTML deliverable
Configuration	~2,000	12	Docker, Vite, TypeScript, ESLint, package.json, etc.

4.2 Conventional Effort Estimate

To establish a baseline, we estimate the effort required to build an equivalent platform using a conventional development team without AI assistance. The estimate draws on industry benchmarks for engineering software development, adjusting for the specialized domain knowledge required.

Table 4. Conventional Development Effort Estimate

Role	Scope	Estimate
Structural Engineer	TFP bearing physics, ASCE/AASHTO compliance logic, solver validation, domain modeling	**3–4 months**
Backend Engineer	OpenSeesPy solver integration, API design, discretization, gravity preload hardening	**2–3 months**
Frontend Engineer	React UI, model editor, results panels, analysis dialogs, state management	**3–4 months**
3D Visualization Specialist	Three.js renderer, mode shapes, deformed shapes, force diagrams, plastic hinges	**2–3 months**
QA Engineer	500+ tests, integration suite, cross-browser validation	**1–2 months**
Technical Writer	Engineering report, compliance documentation, calculation sheets	**1 month**

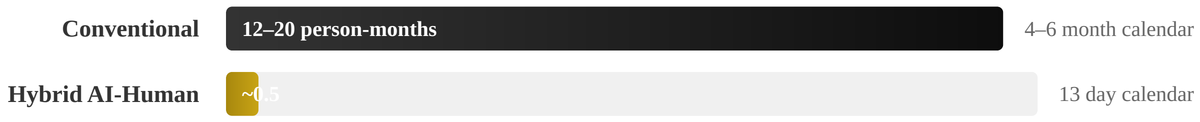
Aggregate estimate: 12–20 person-months with a 3–4 person team, yielding a 4–6 month calendar timeline. A solo senior engineer with full-stack and structural analysis expertise would require 8–12 months.

4.3 Actual Hybrid Effort

Table 5. Actual Development Metrics

Metric	Value
Calendar duration	13 days (Feb 13–25, 2026)
Active coding days	8 of 13 days
Human time investment	~0.5 person-months (estimated)
Peak throughput	15 commits / day (Feb 13); 13 commits / day (Feb 24)
Lines per active day	~9,060 lines/day average

4.4 Compression Ratio



KEY FINDING

The hybrid approach achieved a **25–40× compression** in total development effort (person-months) and a **9–14× compression** in calendar time compared to conventional team-based development. When compared to a solo engineer, the calendar compression reaches **18–28×**.

5. Role Analysis: Human vs. AI Contributions

5.1 The Human Role: Domain Authority & Quality Gate

The structural engineer's contributions fell into four categories that proved irreplaceable:

Physics specification. Defining the four-surface TFP friction model, specifying velocity-dependent friction behavior, establishing that TFP elements require Z-up coordinate convention, setting reasonable friction coefficient ranges (0.01–0.12 for inner surfaces, 0.04–0.20 for outer), and validating that solver outputs matched expected structural behavior.

Code compliance expertise. Translating ASCE 7-22 Chapter 17 and AASHTO 4th Edition requirements into computable checks. This required interpretive judgment—for example, determining that the effective period ratio check should compare the isolated period to the fixed-base period (not to an arbitrary threshold), or that property modification factors should bound both friction and stiffness simultaneously.

Architectural direction. Deciding the phased development approach, selecting the comparison framework as a core differentiating feature, specifying the 5:1 member discretization for force diagram accuracy, and choosing which preset models would best validate the analysis pipeline.

Quality arbitration. Identifying non-obvious bugs that passed all syntactic and type-checking gates. Examples include: incorrect sign conventions in local-to-global force transformations, geometric transformation vector singularities for vertical elements, gravity preload convergence failures in models with many bearings, and Z/Y axis swapping errors in mode shape visualization for isolated models.

5.2 The AI Role: Synthesis Engine & Cross-Stack Executor

The LLM agent's contributions were characterized by:

Parallel code generation. When directed to implement a feature (e.g., "add pushover analysis"), the AI simultaneously generated: the OpenSeesPy solver pathway (~200 lines), the FastAPI route handler (~50 lines), the Pydantic request/response schemas (~80 lines), the React analysis dialog modifications (~60 lines), the results panel component (~150 lines), the Plotly chart configuration (~80 lines), the Zustand store updates (~30 lines), and the TypeScript type definitions (~40 lines)—all in a single pass, maintaining type safety and API contract consistency across the stack.

Test generation at scale. The AI produced 500+ tests that covered topology validation, section auto-sizing, load generation, bearing placement, connectivity checks, coordinate transformation, and parametric generator edge cases. Test quality was generally high, with meaningful assertions (not just "runs without error") and good coverage of boundary conditions.

Documentation synthesis. Engineering documents (984-line calculation sheet, 1,029-line AASHTO compliance review, 1,910-line HTML report) were generated with correct structural engineering terminology, proper equation formatting, and accurate numerical results drawn from actual solver outputs.

Iterative debugging velocity. When the engineer identified a bug (e.g., "force diagrams are flipped for vertical elements"), the AI could trace the issue across the full stack—from solver output format through API serialization to frontend rendering—and apply a coordinated fix in minutes rather than the hours such cross-layer debugging typically requires.

5.3 Failure Modes & Corrections

The hybrid workflow was not without friction. Twelve of 41 commits (29%) were explicitly labeled as fixes, indicating that the initial AI-generated code required correction. The most common failure modes were:

Table 6. Common AI Generation Failure Modes

Failure Mode	Frequency	Detection Method
Coordinate convention errors	4 instances	Human visual inspection of 3D outputs
Sign convention inversions	3 instances	Human comparison with hand calculations
Solver convergence failures	3 instances	Integration test failures
API contract mismatches	2 instances	TypeScript type errors / runtime errors

Notably, all coordinate and sign convention errors were caught by the human, not by automated tests. These errors are semantically invisible to type systems and test frameworks—the code runs correctly and produces plausible-looking numbers, but the numbers are physically wrong. This underscores the essential role of domain expertise in the hybrid workflow.

6. Discussion

6.1 When the Hybrid Model Works Best

The results suggest that AI-human hybrid development is most effective when the following conditions are met:

1. **Deep domain expertise exists in the human participant.** The engineer's ability to specify physically correct behavior, catch semantically invisible bugs, and validate outputs against engineering intuition was the quality bottleneck. Without this expertise, the AI would have produced a visually impressive but physically incorrect tool.
2. **The implementation space is well-defined by prior art.** React, Three.js, FastAPI, and OpenSeesPy are mature, well-documented technologies. The AI's training data includes extensive examples of each, enabling high-quality code generation. For novel or poorly documented technologies, AI generation quality degrades significantly.
3. **The project requires cross-stack coordination.** The AI's ability to maintain consistency across TypeScript types, Python schemas, API contracts, and solver output formats provided a significant advantage over human developers who typically specialize in one layer.
4. **Iteration speed matters more than perfection.** The 29% fix rate would be unacceptable in a waterfall process, but in a rapid iteration model where fixes are applied within minutes, it is an acceptable trade-off for the speed of initial generation.

6.2 Limitations of This Study

Several caveats apply to the compression ratio estimates:

- **Human time is underestimated.** The 0.5 person-month estimate counts only active coding/review time. It excludes the engineer's years of accumulated domain knowledge, which is a prerequisite but not a project cost.
- **Conventional time may be overestimated.** A team with existing boilerplate, component libraries, or prior isolation analysis code could potentially reduce the conventional timeline by 30–40%.

- **Maintainability is untested.** The 72,500-line codebase was generated rapidly. Long-term maintainability, onboarding of new developers, and technical debt accumulation remain open questions.
- **Single-project generalization risk.** This case study involves one project, one engineer, and one AI system. The compression ratio may vary significantly for different domains, team compositions, and LLM capabilities.

6.3 Implications for Engineering Software Development

The results have several implications for the structural engineering and broader engineering software communities:

Democratization of specialized tools. The cost of building domain-specific analysis software has dropped by an order of magnitude. This enables individual engineers or small firms to build custom tools tailored to their specific practice areas, rather than depending on monolithic commercial software packages.

Shift in the value of engineering expertise. As code generation becomes commoditized, the differentiating value shifts further toward domain knowledge, physical intuition, and quality judgment. Engineers who can precisely specify what a tool should compute—and validate that it computes correctly—become more valuable, not less.

New quality assurance challenges. AI-generated code that is syntactically correct, type-safe, and passes generated tests can still be physically wrong. The field needs new validation methodologies that go beyond traditional software QA to include physics-based verification, benchmark comparisons, and domain expert review protocols.

Pace of innovation. The 13-day development timeline suggests that prototype-to-production cycles for engineering tools could shrink from years to weeks. This creates opportunities for rapid experimentation with new analysis methods, visualization techniques, and code compliance workflows.

7. Conclusions & Recommendations

7.1 Conclusions

This case study documented the development of IsoVis, a 72,500-line seismic isolation analysis platform, in 13 calendar days through hybrid AI-human collaboration. The key conclusions are:

1. **The hybrid model achieved a 25–40× effort compression** compared to conventional development, reducing an estimated 12–20 person-months to approximately 0.5 person-months. Calendar time was compressed 9–14× versus a conventional team and 18–28× versus a solo developer.
2. **Domain expertise was the irreplaceable human contribution.** The structural engineer's knowledge of TFP bearing physics, seismic code requirements, and structural analysis methodology could not be replicated by the AI. All physically significant bugs (coordinate conventions, sign conventions, convergence parameters) were identified by the human, not by automated tests or AI self-review.
3. **Cross-stack fluency was the AI's decisive advantage.** The ability to simultaneously generate correct, consistent code across React, Three.js, FastAPI, OpenSeesPy, and Pydantic—maintaining type safety and API contracts throughout—eliminated the communication overhead and integration delays that dominate multi-developer projects.
4. **The 29% fix rate is acceptable within a rapid-iteration workflow.** While initial AI outputs required corrections in roughly one-third of commits, the speed of the fix cycle (minutes, not hours) meant that the net throughput remained far above conventional rates.
5. **The platform is functionally complete and validated.** With 500+ automated tests, 23 integration tests against a live OpenSeesPy solver, 12 automated AASHTO compliance checks, and a complete engineering case study (St. Claire Memorial Hospital: 64% base shear reduction, 93% drift reduction), IsoVis meets the standard of a production-grade analysis tool.

7.2 Recommendations

Based on the findings of this study, we offer the following recommendations for practitioners considering hybrid AI-human development workflows:

For Engineering Software Development

1. **Invest in specification quality.** The quality of AI-generated code is directly proportional to the precision of the human specification. Engineers should develop the skill of writing precise, physics-aware feature specifications that constrain the AI's solution space toward correct implementations.
2. **Implement physics-based validation gates.** Traditional software QA (unit tests, type checking, linting) is necessary but insufficient for engineering software. Establish benchmark problems with known analytical solutions, and validate every solver pathway against these benchmarks before accepting AI-generated analysis code.
3. **Adopt a phased, iterative approach.** The five-phase plan used in this project—scaffolding, basic analysis, advanced physics, nonlinear analysis, comparison/compliance—provided natural validation checkpoints. Each phase could be independently verified before building upon it.
4. **Maintain human ownership of coordinate conventions and sign conventions.** These are the most common and most dangerous failure modes in AI-generated structural analysis code. Establish conventions explicitly at the start of the project and verify them at every integration boundary.

For the Research Community

1. **Develop domain-specific AI benchmarks.** The field needs standardized benchmarks for evaluating AI-generated engineering code—not just for syntactic correctness, but for physical correctness. These benchmarks should include problems with known analytical solutions across multiple analysis types.
2. **Study long-term maintainability.** The compression ratios reported here apply to initial development. Future work should assess whether AI-generated codebases are more or less maintainable than human-authored equivalents, and whether AI agents can effectively maintain and extend codebases they generated.
3. **Explore hybrid workflows for code compliance.** The automated ASCE 7-22 and AASHTO compliance checks in IsoVis suggest that AI can accelerate the tedious process of translating code provisions into computable checks. A systematic study of AI-assisted code compliance across multiple building codes would be valuable.

For Policy and Standards Bodies

1. **Develop guidelines for AI-assisted engineering analysis.** As AI-generated analysis tools proliferate, professional engineering bodies should establish guidelines for validating, certifying, and taking professional responsibility for AI-assisted structural analysis results. The existing framework

of independent design review and peer checking provides a foundation, but specific protocols for AI-generated solvers are needed.

2. **Consider the implications for engineering education.** If domain expertise is the irreplaceable human contribution in AI-augmented workflows, engineering curricula should emphasize physical intuition, code literacy, and validation methodology alongside traditional analysis techniques.

SUMMARY STATEMENT

The development of IsoVis demonstrates that hybrid AI-human collaboration can compress specialized engineering software development by more than an order of magnitude. The key to this compression is not the replacement of engineering expertise, but its *amplification*: a single domain expert, armed with an AI implementation agent, can produce in days what would previously require a multi-disciplinary team working for months. The engineer's irreplaceable role—knowing what to build, why it matters, and whether it is physically correct—becomes *more* valuable, not less, in this new paradigm.

References

[1] American Society of Civil Engineers. *Minimum Design Loads and Associated Criteria for Buildings and Other Structures*, ASCE/SEI 7-22. Reston, VA: ASCE, 2022. Chapter 17: Seismically Isolated Structures.

[2] American Association of State Highway and Transportation Officials. *Guide Specifications for Seismic Isolation Design*, 4th Edition. Washington, DC: AASHTO, 2014.

[3] Constantinou, M.C., Kalpakidis, I., Filiatrault, A., and Ecker Lay, R.A. *LRFD-Based Analysis and Design Procedures for Bridge Bearings and Seismic Isolators*, MCEER-11-0004. Buffalo, NY: MCEER, 2011.

[4] McKenna, F., Scott, M.H., and Fenves, G.L. "Nonlinear Finite-Element Analysis Software Architecture Using Object Composition." *Journal of Computing in Civil Engineering*, ASCE, 24(1):95–107, 2010.

[5] Zhu, M., McKenna, F., and Scott, M.H. "OpenSeesPy: Python Library for the OpenSees Finite Element Framework." *SoftwareX*, 7:6–11, 2018.

[6] Fenz, D.M. and Constantinou, M.C. "Spherical Sliding Isolation Bearings with Adaptive Behavior: Theory." *Earthquake Engineering & Structural Dynamics*, 35(2):163–183, 2006.

[7] Fenz, D.M. and Constantinou, M.C. "Modeling Triple Friction Pendulum Bearings for Response-History Analysis." *Earthquake Spectra*, 24(4):1011–1028, 2008.

[8] Sarlis, A.A. and Constantinou, M.C. *Model of Triple Friction Pendulum Bearing for General Geometric and Frictional Parameters and for Uplift Conditions*, MCEER-13-0010. Buffalo, NY: MCEER, 2013.

[9] Naeim, F. and Kelly, J.M. *Design of Seismic Isolated Structures: From Theory to Practice*. New York: John Wiley & Sons, 1999.

[10] Anthropic. "Claude: A Large Language Model for Safe and Useful AI." Technical documentation, 2024–2026.

[11] Chen, M., et al. "Evaluating Large Language Models Trained on Code." *arXiv preprint arXiv:2107.03374*, 2021.

[12] Fan, A., et al. "Large Language Models for Software Engineering: A Systematic Literature Review." *ACM Computing Surveys*, 56(4):1–39, 2024.

[13] React Three Fiber. "A React Renderer for Three.js." Open-source documentation, 2020–2026. https://docs.pmnd.rs/react-three-fiber.

[14] Tiangolo, S. "FastAPI: Modern, Fast (High-Performance), Web Framework for Building APIs with Python." Open-source documentation, 2019–2026. https://fastapi.tiangolo.com.

A. Appendix: Detailed Project Metrics

Table A-1. Frontend Component Breakdown

Feature Module	Files	Lines	Description
viewer-3d	19	4,574	3D rendering, node/element/bearing visualization, deformed shapes, force diagrams, plastic hinges
bent-build	8	4,713	Parametric bridge generator with per-pier, deck, and support configuration
bay-build	5	2,120	Parametric building frame generator with material and isolation options
model-editor	16	1,779	Hierarchical editor for nodes, elements, sections, materials, bearings, loads
results	6	1,078	Analysis results panels with tables and interactive charts
analysis	4	685	Analysis dialog, run hooks, async execution
comparison	5	665	Isolated vs. fixed-base comparison dashboard
layout	3	405	Application shell, toolbar, status bar
controls	1	296	Viewer display controls and toggles
property-inspector	4	271	Read-only property display for selected elements

Table A-2. Test Suite Breakdown

Test Category	Tests	Lines	Coverage Area
Bent Build (Vitest)	113	~3,800	Bridge topology, connectivity, section sizing, loads, bearings, deck offsets
Bay Build (Vitest)	67	~2,200	Frame topology, auto-sizing, gravity loads, bearing placement, diaphragms
Frontend Other (Vitest)	27	~900	Stores, serializer, type guards, component rendering
Backend Unit (Pytest)	124	~2,600	Solver pathways, schema validation, API routes (mocked OpenSeesPy)
Integration (Pytest)	23	~1,700	End-to-end analysis through live OpenSeesPy solver

Table A-3. Case Study Results — St. Claire Memorial Hospital

Performance Metric	Fixed-Base	Isolated	Improvement
Base Shear (kips)	84.4	30.3	**64% reduction**
Max Drift Ratio	0.471%	0.035%	**93% reduction**
Plastic Hinges	15	0	**100% elimination**
Performance Level	Life Safety	Immediate Occupancy	**2-level upgrade**
Fundamental Period (sec)	0.429	2.540	**5.9× shift**
Effective Damping	5%	41.5%	**8.3× increase**
Bearing D/C Ratio (MCE)	—	0.98	Within capacity