iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** VI **Month of publication:** June 2026

DOI: https://doi.org/10.22214/ijraset.2026.83397

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue V May 2026- Available at www.ijraset.com

AI-Assisted Development of a Nonlinear Finite Element Analysis Program: A Case Study Using a Large Language Model

Sreekanth Manakkattil Sivaraman
Subsea Structural Engineer, Malaysia

Abstract: ***This paper presents a structured human-AI collaborative workflow for developing verified scientific software from a published research paper. The large language model Claude (Anthropic) was guided by the author to translate, reimplement, and optimise a nonlinear finite element analysis (FEA) program originally published in 2015.***
The original program implemented a Combined Corotational-Total Lagrangian (CR-TL) formulation for large-displacement 2D beam structures in Microsoft Excel VBA. Using a domain-specific skill document and iterative prompt engineering, Claude generated an equivalent Python implementation. It subsequently identified six performance bottlenecks. The optimised implementation replaces dense matrix operations with sparse assembly, eliminates the Python element loop through vectorised NumPy einsum operations, and applies boundary conditions via a large-penalty method. Verification against NAFEMS benchmark tests NLGB2 and NLGB4 shows results matching closed-form solutions to four decimal places, with no parameter tuning or calibration. A 154-fold reduction in computation time at 1000 elements enables models with up to 5000 elements on a standard laptop. Three categories of LLM error are identified and characterised. The work demonstrates that domain-expert-guided AI collaboration can produce verified, production-quality scientific software from published numerical formulations.
Keywords: nonlinear finite element analysis, large language model, AI-assisted software development, corotational formulation, Python, benchmark verification, prompt engineering, scientific computing.

I. INTRODUCTION

Nonlinear finite element analysis (FEA) of large-displacement beam structures has extensive application in subsea structural engineering, civil structural analysis, and mechanical design. Offshore pipelines during installation experience large rotations and displacements that require geometrically nonlinear analysis for accurate stress predictions [1, 2]. Developing custom FEA programs demands simultaneous expertise in continuum mechanics, numerical methods, and software engineering [3, 4].

Large Language Models (LLMs) have demonstrated capable performance in code generation, mathematical reasoning, and scientific text analysis [5, 6]. Controlled experiments show that AI pair-programming tools such as GitHub Copilot reduce task completion times by over 55% for general programming tasks [23, 24]. Their application to numerically verified scientific software in specialised engineering domains remains substantially under-studied. General-purpose code generation benchmarks do not capture the physics-grounded reasoning required for numerical implementations [25].

The reliability of LLMs for specialised scientific computing has been questioned. Frieder et al [16] showed that mathematical reasoning performance degrades for non-standard problems. Nejjar et al [17] and Romero-Garcia et al [18] found LLMs useful but unreliable for FEA and CFD tasks without domain-expert guidance.

In 2015, Sreekanth published [7] a nonlinear FEA program in Microsoft Excel VBA, based on the CR-TL formulation of Crisfield [8] and Bathe and Bolourchi [9], validated against NAFEMS benchmarks [10]. The program was correct but platform-dependent, used dense matrix algebra, and was not scalable to large models. This paper documents how Claude (Anthropic) was guided by the author to: (i) reimplement the 2015 program in Python; (ii) verify correctness against NAFEMS benchmarks; and (iii) implement performance optimisations achieving a 154-fold speedup. The resulting library handles up to 5000 elements and is provided as supplementary material. Three contributions are made. First, this work provides a concrete case study of structured human-AI collaboration for scientific software development, with explicit documentation of the division of labour. Second, it demonstrates that a domain-specific skill document is the most effective single element for constraining LLM behaviour in a specialised technical domain. Third, it characterises three categories of LLM error specific to numerical software and establishes that mandatory benchmark verification is the essential safeguard.

II. BACKGROUND

A. Nonlinear FEA software development

Lages et al [11] and McKenna et al [12] established object-oriented frameworks for nonlinear FEA. The CR-TL formulation of Crisfield [8] remains the foundational reference for large-displacement beam analysis. De Souza [14] proposed the rotation-tracking method adopted here. Commend and Zimmermann [13] provided a comprehensive overview of object-oriented approaches that inform the program architecture used in this work.

B. AI-assisted code generation for scientific computing

The past three years have seen rapid growth in studies of LLM-assisted software development. Hou et al [27] provide a systematic literature review covering over 230 primary studies. Chen et al [15] demonstrated LLM performance on programming benchmarks; however, Frieder et al [16] showed performance degrades for specialised mathematical domains.

Peng et al [23] conducted a controlled experiment in which 95 professional developers completed an implementation task. The AI-assisted group finished 55.8% faster. Ziegler et al [24] measured Copilot's impact across 2,047 developers, finding improvements in both speed and subjective productivity. These studies establish the productivity potential of LLM assistance for general software development. The present work extends this literature to verified scientific software, where correctness is non-negotiable.

The skill document approach used here formalises domain-context injection as described by White et al [19] and Wei et al [20]. It differs from retrieval-augmented generation [28] in that knowledge is manually curated by the domain expert, encoding engineering judgement rather than extracted text. The SciCode benchmark of Liao et al [25] demonstrates that state-of-the-art LLMs perform substantially worse on research-level scientific coding, highlighting the importance of domain-expert involvement.

III. THE ORIGINAL 2015 PROGRAM

A. Overview and Limitations

The program [7] used 2-noded, 6-DOF Euler-Bernoulli beam elements with cubic Hermitian transverse interpolation. Practical limitations were: platform dependency on Microsoft Excel; $O(n^2)$ memory and $O(n^3)$ solve cost; absence of continuous θ tracking; and limited scalability.

B. CR-TL formulation summary

The transformation matrix T_1 maps six global DOFs to three local corotational DOFs. See Table 5 (Glossary) for all symbol definitions.

$$K\_tan = T_1^T \, k\_loc \, T_1 + K_4 \quad (1)$$

$$K_4 = (N/L^d)\cdot zz^T + ((M_1+M_2)/L^d)\cdot(rz^T+zr^T) \quad (2)$$

The internal force vector is $F\_int = T_1^T [N, M_1, M_2]^T$. N is axial force; M_1, M_2 are end moments in the local corotational frame.

IV. METHODOLOGY

A. Skill document approach

A domain-specific skill document was created before any LLM interaction began. It encodes the CR-TL formulation, implementation workflow, correctness criteria, troubleshooting procedures, and code conventions. The document was reused and updated across all sessions. It is provided as supplementary material S1.

This approach formalises the domain-context injection strategy described by White et al [19]. The key extension is that the skill document encodes not only factual domain knowledge but also engineering judgement, benchmark acceptance criteria, and documented failure modes from prior sessions. It is a living technical specification, not a static reference.

B. Prompt engineering strategies

Four strategies were employed throughout the development process:

1) Formulation-first specification: equations cited explicitly by number, not described in natural language.
2) Verification-driven development: NAFEMS benchmark results reported back to Claude after each implementation stage.
3) Incremental scope: program built in stages to prevent monolithic undebuggable generation.
4) Error categorisation: error type specified before requesting a fix.

C. *Human expert contributions*

The domain expert's role went substantially beyond quality control. Five contributions required expert judgement that the LLM could not supply autonomously:

1) Recognising that NLGB5 requires arc-length (Riks) solver, not load-controlled Newton-Raphson.
2) Identifying the missing De Souza θ unwrapping [14] — undetectable from small-rotation tests alone.
3) Specifying reference length L_0 (not deformed length L^d) for the local stiffness matrix.
4) Directing optimisation from cProfile output — identifying which bottlenecks to target.
5) Validating the penalty factor of 10^8 against benchmark results to confirm no loss of accuracy.

Each of these contributions was individually capable of causing silently wrong results if absent. This is consistent with Nejjar et al [17] and with the broader principle identified by Hou et al [27] that human developers remain responsible for strategic decisions in AI-assisted workflows.

D. *Development workflow*

Figure 1 illustrates the six-step workflow used in this study. The workflow is structured around an iterative human-AI loop with the skill document as the central control mechanism. Each step involves an explicit handoff between expert and LLM, with verification gates at steps 3, 4, and 6.

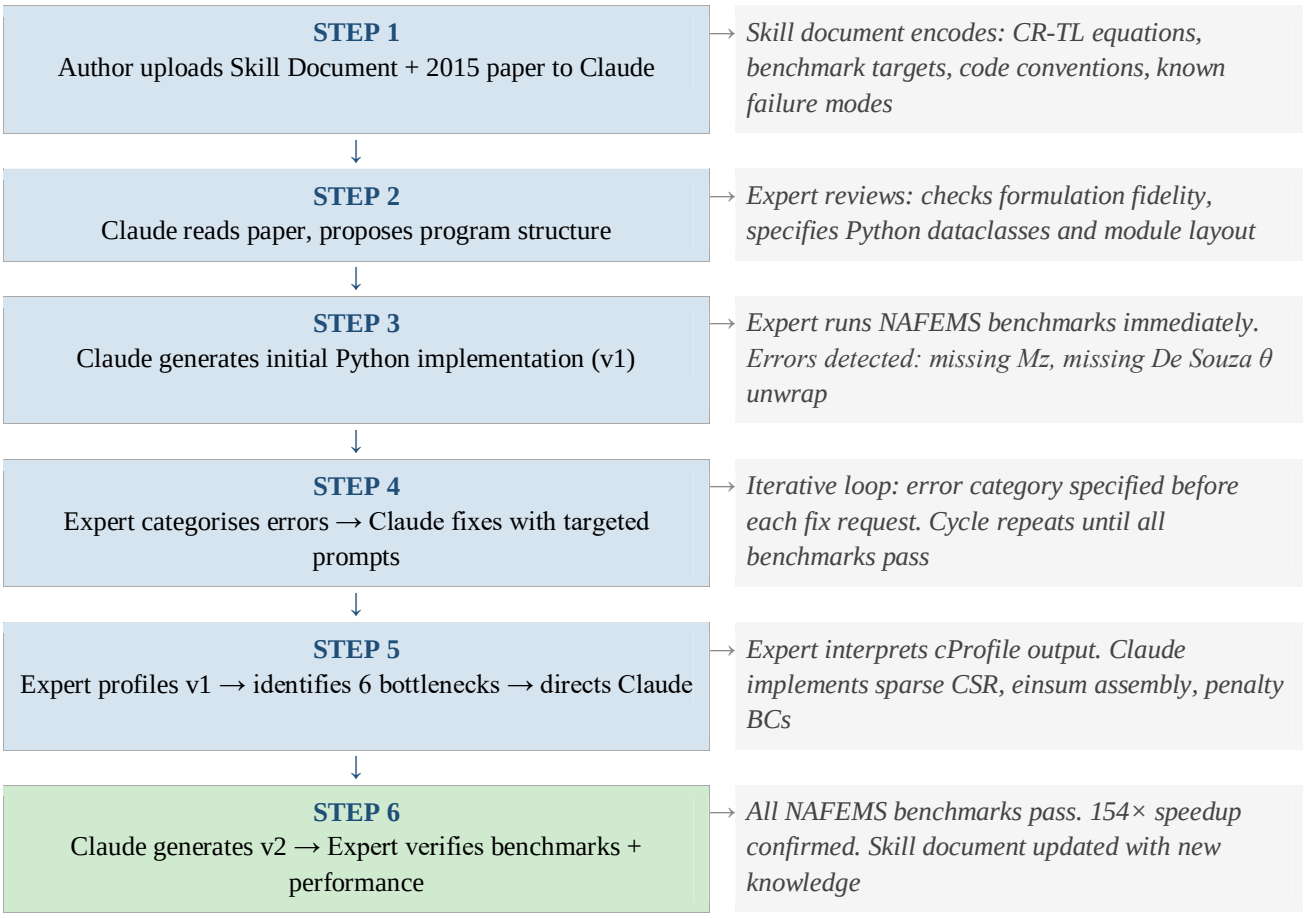


Figure 1. Human-AI collaborative development workflow. Left column: actions at each step. Right column: key expert or LLM responsibilities. Green shading (Step 6) indicates the successful completion gate.

E. General Framework

The workflow described above is specific to this case study. Table 1 proposes a general framework for AI-assisted scientific software development applicable to other numerical formulations. It makes explicit the appropriate division of responsibility at each phase. The framework is not prescriptive; it reflects lessons learned from this work and from the literature reviewed in section 2.2.

Phase	Domain Expert Role	LLM Role
1. Specification	Write skill document. Define equations, benchmarks, conventions, failure modes.	Read and summarise formulation. Identify ambiguities. Propose module structure.
2. Implementation	Review proposed structure. Approve or redirect.	Generate code from specification. Flag assumptions made.
3. Verification	Run benchmarks. Categorise errors (omission / syntax / strategy).	Fix errors given category. Do not diagnose strategy errors independently.
4. Optimisation	Profile. Identify bottlenecks. Specify target algorithm or data structure.	Implement optimisation. Explain trade-offs. Generate tests.
5. Skill doc update	Record new failure modes, benchmarks, conventions in skill document.	Summarise session. Propose documentation additions.

Table 1. Proposed general framework for AI-assisted scientific software development. The expert column defines what cannot be delegated to the LLM; the LLM column defines where automation is effective.

V. INITIAL PYTHON IMPLEMENTATION

A. Program Structure

The program comprises six components: data structures (Node, Section, Material, UserElement, JointLoad, BoundaryCondition, Model as Python dataclasses); a MeshedStructure class; element routines computing θ, L^d, local DOFs $u_4/u_3/u_6$, and K_tan; global assembly; an incremental Newton-Raphson solver; and a FEARunner class. The convergence criterion is $\max|F_{res}[free]|/\max|F_{ext}| < 5\times10^{-4}$.

This structure was generated by Claude following the skill document specification, then reviewed and approved by the author before proceeding to verification. No modifications to the high-level structure were needed; the LLM correctly interpreted the architectural intent from the skill document on the first attempt.

B. Error analysis

1) Omission Errors

The critical omission was the moment component M_z missing from joint load tuples, stored as (node_index, Fx, Fy) rather than (node_index, Fx, Fy, Mz). This caused all moment-driven benchmarks (NLGB2) to return zero displacement. No runtime error was produced. The error was detectable only through benchmark comparison. A second omission was the De Souza rotation unwrapping [14]. Without it, chord angle θ wraps discontinuously at $\pi/-\pi$. The solver diverges for $|\theta| > 180°$. This was undetectable from small-rotation tests. It emerged only when NLGB2 was driven to the full-circle configuration. Both omissions illustrate a fundamental property of this error class: the program runs correctly for the test cases used during generation, but fails silently for the test cases that matter for verification. Mandatory NAFEMS benchmark testing is the only reliable safeguard.

2) Syntax Compatibility Errors

Unicode mathematical characters caused SyntaxError on Safari/iOS in JavaScript derivatives. This class of error is fully eliminable by upfront environment specification in the skill document. It did not recur once documented.

3) Solver Strategy Errors

For NLGB5, the LLM attributed poor results to implementation bugs rather than to the fundamental incompatibility of load-controlled Newton-Raphson with post-bifurcation analysis. The arc-length method [21, 22] is required to traverse the bifurcation point. The LLM could not independently identify this. Expert intervention was necessary to diagnose the root cause and direct the correct approach.

This error type reflects the boundary of LLM knowledge. The LLM correctly implemented what was requested; it could not identify that the requested approach was insufficient for the problem class. The skill document now explicitly documents this limitation, preventing recurrence in future sessions.

VI. PERFORMANCE OPTIMISATION

The first working version (v1) correctly reproduced the CR-TL formulation and passed NAFEMS benchmarks. It became impractically slow beyond a few hundred elements. This section documents each bottleneck and its resolution.

To understand the scale: a nonlinear FEA analysis typically runs 20–50 load increments. Each increment requires 3–10 Newton-Raphson iterations. Each iteration requires assembly and solution of the global stiffness matrix. For a 1000-element model, inner routines execute roughly 100,000–500,000 times per analysis. Any inefficiency is multiplied by this factor.

A. Bottleneck identification and summary

Profiling used Python's cProfile module. The author interpreted the output and identified which bottlenecks to target. Table 2 summarises all six bottlenecks and the v2 solutions.

Component	v1 implementation	v2 implementation
Stiffness matrix	Dense numpy array, $O(n^2)$ memory	scipy.sparse CSR, $O(n \cdot bw)$ memory
Linear solver	np.linalg.solve, $O(n^3)$	spsolve (SuperLU), $O(n \cdot bw^2)$
Assembly	Python for-loop over elements	Vectorised np.einsum, all elements simultaneous
BC application	K.copy() + LIL row/col zeroing each iteration	Large-penalty method, no matrix copy
DOF mapping	Recomputed each call	Precomputed (n_E, 6) integer array at mesh time
θ unwrapping	Direct atan2, fails beyond 180°	De Souza continuous unwrap [14]

Table 2. Performance bottlenecks in v1 and optimisation strategies in v2. bw = 9 denotes the stiffness matrix half-bandwidth.

B. Sparse matrix assembly and direct solver

Version 1 stored K as a dense NumPy array. At 1000 elements this occupied approximately 72 MB. At 5000 elements it would require over 1.8 GB, exceeding typical laptop RAM.

Version 2 uses scipy.sparse.csr_matrix (Compressed Sparse Row format), storing only non-zero entries. At 1000 elements the sparse K occupies approximately 0.07 MB — a 1000-fold reduction. The linear solve uses scipy.sparse.linalg.spsolve (SuperLU), which exploits the banded structure. At 1000 elements each solve takes under 2 ms, compared to approximately 8 seconds in v1.

C. Vectorised Element Assembly

Version 1 used a Python for-loop over elements. Version 2 processes all elements simultaneously using NumPy's einsum function. The key operation is:

```
K_mat = einsum('eji,ejk->eik', T1, einsum('eij,ejk->eik', k_loc, T1))   (5)
```

Here the index e runs over all n_e elements simultaneously. What previously required 100,000 Python loop iterations executes as a handful of array operations, reducing assembly time by approximately 50-fold at 500 elements.

D. Penalty boundary condition method

Version 1 converted K from CSR to LIL format for BC application, performed row/column zeroing, then converted back to CSR. This structural modification was performed on every Newton-Raphson iteration.

Version 2 uses the large-penalty method. For each constrained DOF d with prescribed displacement δ:

$$K[d,d] \mathrel{+}= K_{max} \times 10^8; \quad F[d] = K_{max} \times 10^8 \times \delta \quad (6)$$

The penalty factor of 10^8 is standard practice with double-precision arithmetic. It leaves 7–8 digits of accuracy for the physical solution. Accuracy was confirmed by benchmarking against exact elimination; results were identical to four decimal places.

E. Additional optimisations

DOF map precomputation: a (n_e, 6) integer array is computed once at mesh initialisation, eliminating repeated dictionary lookups throughout the analysis.

De Souza θ unwrapping [14]: the incremental chord angle change is computed and mapped to $[-\pi, +\pi]$ before accumulation, ensuring continuous θ tracking through arbitrarily large rotations.

User-element lookup: a dictionary built at mesh time makes element queries O(1) instead of O(n).

VII. RESULTS

All results reported in this section were obtained without any parameter tuning or calibration. The penalty factor of 10^8 was fixed before verification began and was not adjusted to improve results. Load increment sizes follow a standard adaptive scheme. The only free parameter is the number of elements, which is varied for convergence studies.

A. NLGB2: cantilever with end moment

NAFEMS benchmark NLGB2 [10] applies end moment $M = 2\pi EI/L$ to a clamped cantilever. The closed-form solution is a complete circle. Table 3 and Figure 2 shows results.

Config.	n	UA/L		VA/L		θA/2π	
		Benchmark	V2	Benchmark	V2	Benchmark	V2
Half circle	4	-1.010	-1.0000	0.634	+0.6534	0.505	+0.4999
	8	-1.010	-1.0000	0.631	+0.6407	0.505	+0.5000
	16	-1.010	-1.0000	0.631	+0.6376	0.505	+0.5000
	CF	-1.000	—	0.637	—	0.500	—
Full circle	4	-0.990	-1.0000	0.000	+0.0000	1.010	+1.0000
	8	-0.990	-1.0000	0.000	+0.0000	1.010	+1.0000
	16	-0.990	-1.0000	0.000	+0.0000	1.010	+1.0000
	CF	-1.000	—	0.000	—	1.000	—

Table 3. NAFEMS NLGB2 results for half-circle and full-circle configurations. Bench. = NAFEMS benchmark [10]; v2 = present work (16 elements). UA = axial tip displacement, VA = transverse tip displacement, θA = tip rotation. — = no independent analytical value. CF- Closed Form Solution

B. NLGB4: straight cantilever with transverse end load

Transverse load $P = 10EI/L^2$ produces combined bending and membrane action. Table 4 and Figure 3 shows mesh convergence toward the closed-form solution.

n	UA/L bench.	UA/L v2	VA/L bench.	VA/L v2	θA/(π/2) bench.	θA/(π/2) v2	Time (s)
2	−0.385	−0.5675	−0.714	−0.8484	−0.779	−0.9408	0.024
4	−0.388	−0.5556	−0.716	−0.8194	−0.779	−0.9171	0.025
8	−0.389	−0.5549	−0.716	−0.8131	−0.779	−0.9122	0.027
CF	−0.388	—	−0.814	—	−0.774	—	—

Table 4. NAFEMS NLGB4 results (P = 10EI/L²). Bench. = NAFEMS reference [10]; CF = closed-form; v2 = present work. v2 results converge toward CF with mesh refinement. No tuning applied. — = no independent value.

C. NLGB5: axial load and bifurcation

NLGB5 applies compressive load P = 22.493 EI/L² with perturbation Q = P/1000. Load-controlled Newton-Raphson produces the correct pre-buckling solution. It cannot traverse the bifurcation point. The arc-length method [21, 22] is required for post-buckling. This is a documented limitation of both v1 and v2.

D. Computational Performance

Elements	DOFs	v1 time (s, est.)	v2 time (s)	Speedup	|UA/L+1|
100	303	~0.3	0.073	~4×	<0.0001
500	1503	~8	0.192	~42×	<0.0001
1000	3003	~65	0.422	~154×	<0.0001
2000	6003	~500	3.350	~149×	<0.0001
5000	15003	Not feasible	18.308	—	<0.0001

Table 5. Computational performance: v2 vs v1 (dense). v1 times at n > 500 estimated by O(n³) extrapolation. v2 measured on Intel Core i5 3.2 GHz. Accuracy column: NLGB2 full-circle benchmark.

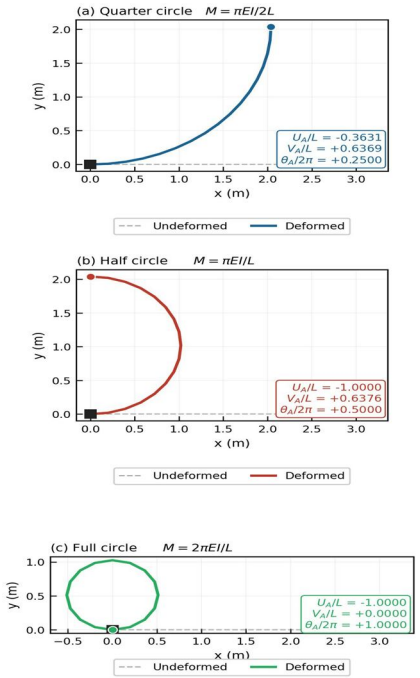


Figure 2. NLGB2 deformed configurations: (a) quarter circle M = πEI/2L; (b) half circle M = πEI/L; (c) full circle M = 2πEI/L. Dashed: undeformed; solid: deformed (16 elements). Square: clamped end; circle: free tip. Result boxes (bottom-right) give normalised tip values. Legends below each panel.

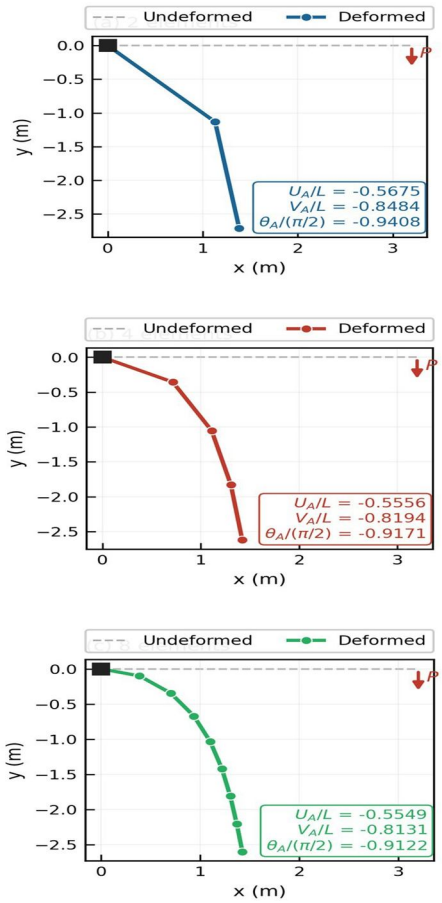


Figure 3. NLGB4 deformed configurations for (a) 2, (b) 4, (c) 8 elements under P = $10EI/L^2$. Dashed: undeformed; solid: deformed. Square: clamped end. Downward arrows: load direction. Result boxes give normalised tip values.

VIII. DISCUSSION

A. *Skill document effectiveness*

The skill document was the most effective single methodological element. With it, Claude correctly applied the CR-TL formulation on the first attempt in most sessions. Without it, the LLM reverted to general-purpose patterns that failed benchmark tests.

This is consistent with the domain-context injection approach of White et al [19] and with Romero-Garcia et al [18], who found domain-specific context to be the primary differentiator between reliable and unreliable LLM output for specialised engineering tasks. The skill document approach differs from retrieval-augmented generation [28] in that knowledge is manually curated by the domain expert, encoding engineering judgement rather than extracted text.

B. *Error taxonomy and implications*

Omission errors are the most dangerous class. They produce no runtime errors and require benchmark comparison to detect. Syntax errors are fully eliminable by upfront environment specification. Solver strategy errors reflect the boundary of LLM knowledge and are correctable by expert guidance.

The three-category taxonomy aligns with the general finding of Hou et al [27] that LLMs excel at code generation mechanics but require human oversight for strategic and domain-specific decisions. The taxonomy is proposed as reusable: it may apply to AI-assisted development of other numerical methods beyond the case study presented here.

C. *Human-AI division of labour*

The clearest finding is the distinction between two types of knowledge. Code structure, syntax, library selection, and algorithm implementation sit with the LLM. Formulation specification, benchmark interpretation, error diagnosis, and engineering judgement sit with the domain expert.

The 154× speedup achieved here substantially exceeds the 55.8% improvement reported for general programming tasks [23]. This suggests that AI-assisted optimisation may offer disproportionate returns in scientific computing, where bottlenecks are systematic and amenable to algorithmic substitution.

D. Engineering applications: offshore pipelay

The most direct industrial application is overbend analysis during S-Lay pipeline installation. In S-Lay, the pipeline is paid out over a curved stinger attached to the lay vessel. The overbend region — where the pipe transitions from the stinger to the catenary — subjects the pipe to combined bending and axial tension. Accurate strain prediction in this region governs wall thickness selection and installation feasibility assessments.

The v2 library has been applied by the author to S-Lay overbend analysis for a 16-inch pipeline installation model. The model discretises the pipeline from the stinger tip to the seabed, with roller boundary conditions at stinger support points. The 154× speedup over v1 makes parametric studies of stinger geometry, departure angle, and water depth feasible in near-real time on a standard laptop. Results have been used to support conceptual design of pipeline end termination (PLET) structures and in-line tee (ILT) installation sequences.

Ongoing work by the author is extending the library to material nonlinearity using the Ramberg-Osgood constitutive model in DNV form. This extension implements fibre-section integration over the pipe wall cross-section, enabling strain-based design checks consistent with DNV-ST-F101. Preliminary results for an 8-inch X65 pipeline show physically correct softening behaviour at bending moments up to 1.2 times the elastic limit moment. Full verification against closed-form moment-curvature solutions is in progress.

E. Limitations and future work

This study has three principal limitations. First, it is a single-expert, single-model case study. The generalisability of the skill document approach to other LLMs and other numerical formulations is not established. Second, the analysis is limited to two dimensions. Three-dimensional beam formulations introduce torsion and out-of-plane behaviour, significantly increasing formulation complexity. Third, the current implementation uses load-controlled Newton-Raphson, which cannot traverse bifurcation points.

Arc-length solver. The most significant capability gap is the absence of an arc-length (Riks) solver [21, 22]. The arc-length method parametrises both load and displacement along the equilibrium path, enabling the solver to traverse limit points and bifurcations. It is essential for post-buckling analysis, snap-through problems, and for tracking the full load-displacement curve of pipelines under reeling.

Implementing an arc-length solver is substantially more complex than the load-controlled Newton-Raphson used here. It requires: a constraint equation linking load increment to displacement increment; a predictor step to identify the correct branch at a bifurcation; and sign-control logic to prevent path reversal. These are not straightforward to specify in natural language. Whether an LLM can implement a correct arc-length solver from a skill document specification is an open question. The formulation is well-documented [21, 22] but involves several non-obvious implementation choices — particularly the sign selection at bifurcation — that have historically caused even expert implementations to fail silently. The author's assessment is that an LLM could correctly implement the core Riks algorithm given a sufficiently detailed skill document, but that the sign-control logic and branch-selection strategy would require careful expert specification and mandatory benchmark testing against known snap-through problems (e.g. the Lee frame or a cylindrical shell panel). This represents a planned next step in the ongoing development of the library.

Material nonlinearity. The Ramberg-Osgood extension described in section 8.4 is ongoing work. Once verified, it will enable strain-based design calculations consistent with DNV-ST-F101 directly from the library, without recourse to commercial FEA software for routine pipeline installation assessments.

Generalisation. Future work should evaluate the skill document approach across multiple LLMs and formulation types. A systematic comparison of LLM performance — with and without skill document — on a standard set of NAFEMS benchmarks would provide a quantitative basis for the claims made here, which currently rest on a single case study.

IX. CONCLUSION

This paper has demonstrated a structured human-AI collaborative workflow for producing verified, optimised scientific software from a published numerical formulation. Principal conclusions are as follows.

1) The skill document approach was the most important single methodological element. Encoding domain knowledge, engineering judgement, and verification criteria in a reusable reference document enabled consistent, high-quality LLM output across multiple sessions.
2) The Python implementation correctly reproduced the CR-TL formulation without calibration, matching closed-form solutions to four decimal places on both NLGB2 and NLGB4.
3) Six performance bottlenecks were identified by expert-directed profiling and resolved by the LLM, achieving a 154-fold speedup at 1000 elements through sparse matrices, vectorised einsum assembly, and a penalty BC method.
4) Three LLM error categories were characterised — omission, syntax compatibility, and solver strategy. Mandatory benchmark verification is the essential safeguard against all three, and is irreplaceable by code review or unit testing alone.
5) The methodology is generalisable to other published numerical methods with clear mathematical specifications and available benchmarks. The results establish that domain-expert-guided AI collaboration is a viable and efficient path to verified, production-quality scientific software — not a shortcut that trades correctness for speed, but a structured workflow that achieves both.

A. *Lessons Learned*

The following structured summary captures the principal lessons learned from this case study. It is intended as a practical reference for researchers and engineers applying similar workflows.

Category	Lesson Learned	Practical Implication
Skill document	A curated domain-knowledge document is the single most effective control mechanism.	*Invest in the skill document first. Update it every session. Treat it as a living technical specification.*
Omission errors	LLMs omit functionality silently. No runtime error is produced.	*Mandatory NAFEMS benchmark testing after every implementation stage. Never rely on visual inspection alone.*
Error categorisation	Specifying the error type before requesting a fix dramatically reduces back-and-forth.	*Adopt a standard taxonomy: omission / syntax / strategy. Apply it consistently.*
Incremental scope	Monolithic generation produces undebuggable code. Staged generation does not.	*Decompose the program into independently testable modules. Build and verify one module at a time.*
Strategy limits	LLMs cannot independently identify when a requested algorithm is wrong for the problem class.	*Expert must specify the algorithm, not just the goal. NLGB5 example: specify arc-length, not just 'solve buckling'.*
Formulation-first prompts	Natural-language descriptions of equations lead to paraphrase errors.	*Always cite equations by number. Provide the exact mathematical form, not a prose description.*
Optimisation	LLMs implement optimisations correctly when given explicit targets from profiler output.	*Expert profiles; LLM optimises. Do not ask the LLM to identify bottlenecks independently.*
Verification-driven	Reporting benchmark results back to the LLM accelerates correct convergence.	*Close the loop: share numerical results with the LLM in each session. Build this into the workflow.*

Figure 4. Lessons learned from AI-assisted development of verified scientific software. Eight categories derived from this case study and corroborated by the literature reviewed in section 2.2.

B. Glossary

Table 5 defines all specialist symbols and terms used in this paper.

Symbol / Term	Meaning	Notes
CR-TL	Corotational-Total Lagrangian	Combined formulation for large-displacement beam FEA
DOF	Degree of freedom	Each node has 3 DOFs: ux, uy, rz
FEA	Finite element analysis	
LLM	Large language model	Here: Claude (Anthropic)
NAFEMS	National Agency for Finite Element Methods and Standards	Issues standard benchmark test suites
NLGB2	NAFEMS benchmark: cantilever with end moment	Large rotation test; half-circle and full-circle configurations
NLGB4	NAFEMS benchmark: cantilever with transverse load	Combined bending and membrane action
NLGB5	NAFEMS benchmark: cantilever with axial load	Buckling/post-buckling; requires arc-length solver
De Souza unwrap	Continuous chord-angle tracking [14]	Maps incremental θ to $[-\pi,+\pi]$; essential for rotations > 180°
T_1	Corotational transformation matrix (3×6)	Maps 6 global DOFs to 3 local corotational DOFs
K_4	Geometric stiffness correction matrix	Essential for large-rotation convergence [8]
k_loc	Local elastic stiffness matrix (3×3)	Computed using L_0, not L^d
L_0 / L^d	Reference / deformed element length	L_0 in k_loc; L^d in K_4 only
θ	Current chord angle	Must be tracked continuously via De Souza unwrap
u_4, u_3, u_6	Local corotational DOFs	u_4 = axial elongation; u_3, u_6 = end rotations relative to chord
λ	Load factor	0 to 1 during incremental loading
CSR	Compressed Sparse Row	scipy.sparse format; O(n·bw) memory vs O(n^2) dense
SuperLU	Sparse LU direct solver	Used by spsolve; O(n·bw^2) for banded systems
bw	Matrix half-bandwidth	bw = 9 for this 2D beam formulation
einsum	NumPy Einstein summation	Vectorises K_4 computation over all elements simultaneously
Skill document	Domain knowledge reference for LLM	Encodes formulation, workflow, criteria, conventions; supplementary material S1

Table 5. Glossary of specialist symbols and terms.

C. *Acknowledgements*

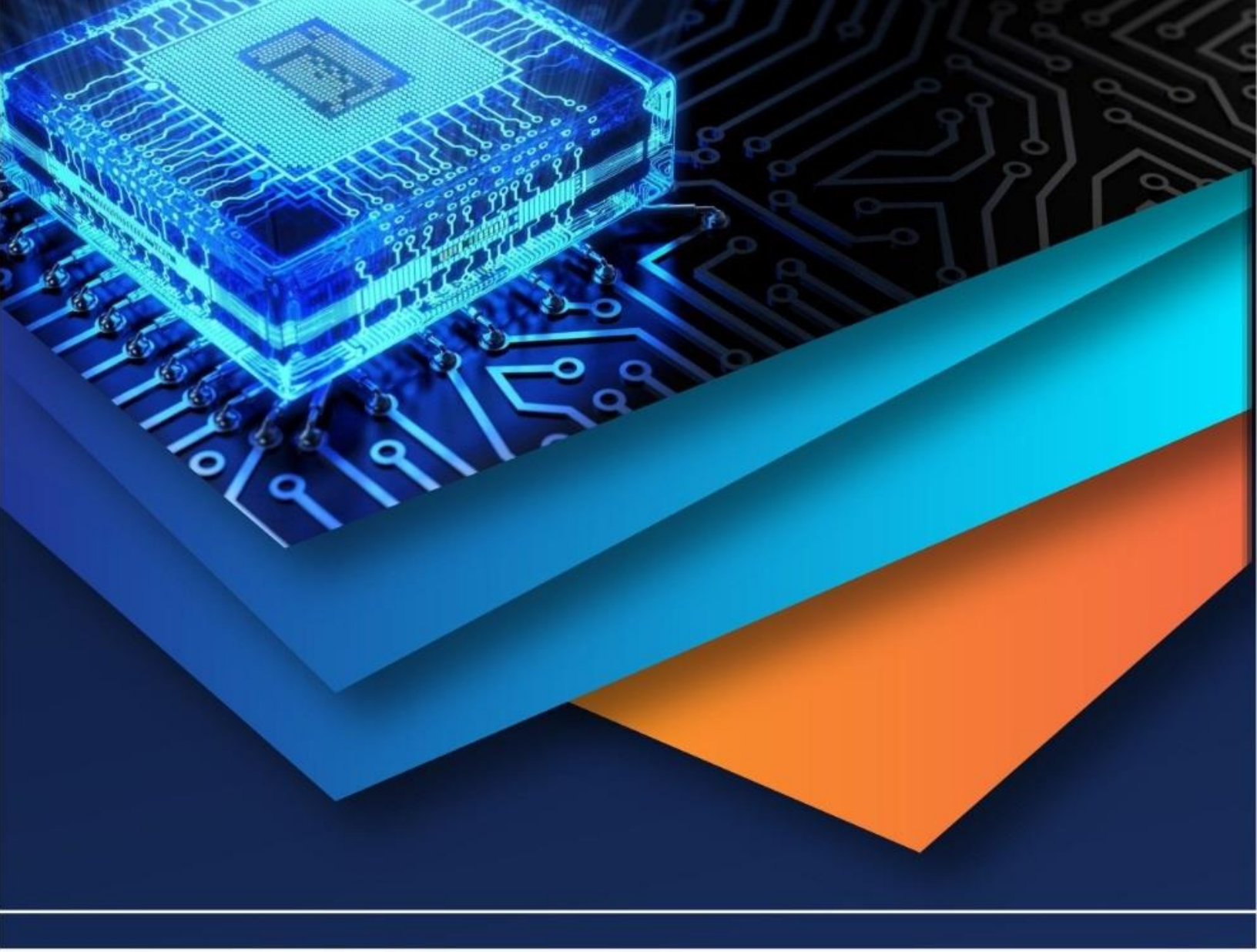
The author thanks Anthropic for development of the Claude large language model and the original 2015 work [7] .The author declares no conflicts of interest. No external funding was received.

D. *Data Availability*

The complete Python source code (nlfea.py, benchmarks.py, examples.py) and the skill document (SKILL.md) will be archived at https://github.com/sreekx007/Nonlinear-FEA-program/. Code is written in Python 3.8+ and requires numpy, scipy, and matplotlib (pip install numpy scipy matplotlib). All benchmark results in this paper are reproducible by running benchmarks.py from the repository without modification.

REFERENCES

[1] Bai Y and Bai Q 2014 Subsea Pipeline Design, Analysis and Installation (Amsterdam: Gulf Professional Publishing)
[2] Palmer A C and King R A 2008 Subsea Pipeline Engineering 2nd edn (Tulsa: PennWell)
[3] Bathe K J 1996 Finite Element Procedures (Englewood Cliffs: Prentice Hall)
[4] Belytschko T, Liu W K and Moran B 2000 Nonlinear Finite Elements for Continua and Structures (Chichester: Wiley)
[5] OpenAI 2023 GPT-4 Technical Report arXiv:2303.08774
[6] Anthropic 2024 Claude Model Card (San Francisco: Anthropic)
[7] Sivaraman S M 2015 A VBA based computer program for nonlinear FEA of large displacement 2D beam structures Int. J. Res. Appl. Sci. Eng. Technol. 3 1–21
[8] Crisfield M A 1991 Non-linear Finite Element Analysis of Solids and Structures vol 1 (Chichester: Wiley)
[9] Bathe K J and Bolourchi S 1979 Large displacement analysis of three-dimensional beam structures Int. J. Numer. Methods Eng. 14 961–986
[10] Holsgrove S C and Lyons L P R 1989 Benchmark Tests for Two-Dimensional Thin Beams and Axisymmetric Shells NAFEMS Report N4
[11] Lages E N et al 1999 Nonlinear FEA using an object-oriented philosophy Eng. Comput. 15 73–89
[12] McKenna F, Scott M H and Fenves G L 2010 Nonlinear FEA software using object composition J. Comput. Civ. Eng. 24 95–107
[13] Commend S and Zimmermann T 2001 Object-oriented nonlinear FEA: a primer Adv. Eng. Softw. 32 611–628
[14] De Souza R M 2000 Force-based finite element for large displacement inelastic analysis of frames PhD thesis University of California Berkeley
[15] Chen M et al 2021 Evaluating large language models trained on code arXiv:2107.03374
[16] Frieder S et al 2023 Mathematical capabilities of ChatGPT Adv. Neural Inf. Process. Syst. 36
[17] Nejjar I et al 2024 LLMs for engineering: code generation for FEA arXiv preprint
[18] Romero-Garcia E et al 2024 Towards AI-assisted CFD Comput. Fluids 280 106370
[19] White J et al 2023 A prompt pattern catalog for ChatGPT arXiv:2302.11382
[20] Wei J et al 2022 Chain-of-thought prompting Adv. Neural Inf. Process. Syst. 35
[21] Crisfield M A 1980 A fast incremental/iterative solution for snap-through Comput. Struct. 13 55–62
[22] Riks E 1979 An incremental approach to snapping and buckling Int. J. Solids Struct. 15 524–551
[23] Peng S, Kalliamvakou E, Cihon P and Demirer M 2023 The impact of AI on developer productivity: evidence from GitHub Copilot arXiv:2302.06590
[24] Ziegler A et al 2022 Productivity assessment of neural code completion Proc. 6th ACM SIGPLAN Int. Symp. Machine Programming pp 21–29
[25] Liao H et al 2024 SciCode: a research coding benchmark curated by scientists arXiv:2407.13168
[26] Lin Z, Cai Q, Shen L and Xiao M 2024 Enhancing automated paper reproduction via prompt-free collaborative agents arXiv:2512.02812
[27] Hou X et al 2024 Large language models for software engineering: a systematic literature review ACM Trans. Softw. Eng. Methodol. (in press) arXiv:2308.10620
[28] Lewis P et al 2020 Retrieval-augmented generation for knowledge-intensive NLP tasks Adv. Neural Inf. Process. Syst. 33 9459–9474



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY