AgentFEM: An AI-Native Open-Source Platform for Finite-Element Computing

Haoming Luo*
Xi'an Thermal Power Research Institute, Xi'an 710032, China

August 2026

Abstract

AI agents can increasingly construct solver programs, but dependable finite-element simulation still requires explicit models, numerical procedures, and evidence. This paper presents AgentFEM, an open-source platform for finite-element workflows shared by people and agents. Its readable Python language expresses the engineering model, a reusable capability layer implements finite-element methods, and FEniCSx, UFL, PETSc, and MPI provide the numerical foundation. Results record solver status, checks, and origin, and the same workflow extends to parameter studies and surrogate modelling. Three cases demonstrate the present scope: a linear-elastic beam, wave propagation through an inclusion, and finite-strain homogenization of a periodic porous cell using C3D10H elements. The current release also provides a direct path from parameterized simulation to learning-ready data and guarded surrogate models. AgentFEM is the AI-native CAE platform; agents and development environments provide ways to work with it. Its broader purpose is to make dependable finite-element software freely available and to prepare CAE for a future in which people and AI advance science together. The software is available at `https://github.com/haoming-luo/agentfem`.

Keywords: finite-element method; scientific software; AI agents; computational mechanics; simulation workflows; surrogate modelling; FEniCSx

1 Introduction

Finite-element analysis is both a scientific model and an executable software process. A model states not only equations but also its physical assumptions, boundary conditions, discretization, solution procedure, and requested results. Mature commercial and open-source systems have made much of this work routine. Yet a model is often divided among a graphical database, an input deck, user subroutines, and post-processing code, making it harder to review and reuse.

AI agents are beginning to participate in a growing share of this process. In 2023, MechAgents showed that collaborating language models could formulate mechanics problems and write, execute, and correct finite-element code (Ni and Buehler, 2023). In 2025, LLM-empowered CAE described agents as collaborators able to plan and adapt computational workflows, while FeaGPT carried a conversational request through geometry, meshing, simulation, and analysis (Guo et al., 2025; Qi et al., 2025).

Recent systems have also made reliability a more explicit concern. In 2026, ALL-FEM combined models trained for finite-element work with execution feedback (Deotale et al., 2026). A constrained interface for FEniCS kept solver formulations in validated deterministic templates (Upadhyay and Reinhart, 2026), while OASiS enabled general agents to work with several established solvers and verify their results (Hermann et al., 2026). Together, these systems record a movement from generating code toward broader agent participation in CAE.

These developments point to two distinct directions for CAE. *AI-enhanced CAE* adds intelligent interaction or orchestration around existing simulation software. *AI-native CAE* redesigns the software so that people and agents can work with the same explicit model, perform the same operations, and inspect the same feedback and evidence. The directions are complementary, but their difference is architectural rather than a matter of how many AI features have been added. This distinction matters because executable code alone is not a dependable simulation: a program may run while expressing the wrong physics (Song et al., 2026), and growing volumes of generated code can become difficult to inspect and maintain.

*Corresponding author: luohaoming@tpri.com.cn; horming.luo@gmail.com.

AgentFEM was initiated to address a practical question: *what should finite-element software look like when both people and agents are first-class users?* AI-native does not mean replacing finite-element computation with a learned model. It means that agents can build and run an explicit workflow while people can understand and control it. The immediate goal is a dependable open-source tool for selected engineering analyses.

AgentFEM addresses this question through its software design. People and agents work with the same readable Python model. Results state whether a calculation ran, converged, passed its checks, and where the data came from. Capability records state the assumptions, maturity, and benchmark evidence of the available methods. The engineering language reflects lessons from the author's long use of Abaqus, Cast3M/Gibiane, and PyTorch; these are intellectual influences, not integrated components (Dassault Systèmes, 2024; Commissariat à l'énergie atomique et aux énergies alternatives, 2026; Paszke et al., 2019). FEniCSx, UFL, PETSc, and MPI perform the numerical work, while AgentFEM organizes the engineering model and workflow (Baratta et al., 2023; Alnæs et al., 2014; Balay et al., 1997; Scroggs et al., 2022).

This paper presents AgentFEM through its design principles, software architecture, and representative applications. It first explains how physical models, numerical procedures, results, and evidence are organized within a readable and extensible finite-element workflow. Three examples then illustrate its current scope in static mechanics, wave propagation, and finite-strain homogenization. The paper further shows how the same simulation model can support parameter studies, scientific datasets, and surrogate modelling, before discussing AgentFEM as an AI-native CAE platform and the continuing responsibility of people for scientific judgement.

2 Design position: finite-element computing for humans and agents

2.1 A shared scientific workspace

Two unsatisfactory extremes are possible. At one extreme, an agent directly emits low-level backend code. This maximizes immediate freedom, but equivalent models fragment into different implementations and reviewers must reconstruct scientific intent from assembly details. At the other extreme, a rigid declarative schema attempts to encode every future finite-element problem. Such a schema may become either too weak for research or too complex to remain usable.

AgentFEM instead uses an executable Python object model built from familiar finite-element concepts such as materials, fields, loads, and steps. Objects define the model, while structured metadata records it when needed. JSON is one export format rather than the modelling language itself.

This design gives humans and agents the same working surface. A person can read the engineering program without learning every backend detail. An agent uses the documented interfaces and their validation messages instead of inventing a complete solver script. Specialists can still work directly with UFL forms or backend objects.

2.2 Probabilistic reasoning, deterministic computation

The division of labour is deliberate. Humans or AI may choose modelling assumptions and prepare the workflow. AgentFEM checks the resulting model and lowers it to deterministic finite-element operations. FEniCSx and PETSc then perform assembly and solution. The calculation therefore does not depend on a particular AI model.

Determinism does not establish correctness. AgentFEM separates four states that are often conflated:

1. *computed*: the program produced a result;
2. *converged*: declared numerical convergence conditions were met;
3. *verified*: specified numerical or regression claims passed;
4. *validated*: comparison with suitable physical evidence supports an intended use.

These states are recorded with the results. Golden results detect software regressions; they do not replace mesh convergence studies or experimental validation.

2.3 Defining AI-native CAE

> ***AI-native is not defined by the presence of AI, but by whether the software itself is designed for intelligence to become one of its native users.***

In this paper, *AI-native CAE* denotes a computational-engineering system in which people and AI agents work through the same explicit scientific model. Agents can inspect and operate that model, receive clear feedback, and trace results to supporting evidence. Open Python interfaces connect the system to the wider scientific and AI ecosystem. Numerical execution remains deterministic, the limits of each capability remain explicit, and scientific decisions remain open to human review. Simulation, data, learning, and verification belong to one workflow rather than being joined after the fact.

AgentFEM realizes this definition with a readable Python engineering language and reusable finite-element capabilities. Its checks report problems directly, and machine-readable records state what each method can do and how mature it is. Simulation results can become scientific datasets within the same workflow. AgentFEM is therefore a shared scientific workspace for people and agents, accessible through present development environments or future forms of human–AI interaction.

2.4 Project propositions and commitments

The definition above describes a software paradigm. The following propositions and commitments state why AgentFEM pursues it and what the project seeks to make possible.

Three propositions guide the project: (1) AI will not replace CAE, but CAE needs to be redesigned for AI; (2) AI-native CAE is a historical opportunity for new and open ecosystems; and (3) AI-native scientific tools, including CAE, will lead to exponential growth in scientific discoveries. Separately, AgentFEM has three commitments of its own: (1) everyone can use free finite-element software; (2) everyone can complete simple calculations with the help of AI; and (3) we work together to advance physical AI grounded in models, observations, and verification.

An AgentFEM case can run as a script on a workstation or cluster, or it can be developed with an AI agent. It is not tied to a particular interface. As human–AI interaction evolves, the same model can be reached through code, graphical tools, or more fluid forms of interaction.

People remain responsible for the scientific question, the model, validation, and use of the results. Agents may help build, run, compare, and document cases, but their work must remain reviewable. AI expands human scientific agency without transferring human scientific responsibility.

3 Software architecture and implementation

AgentFEM's distinguishing feature is not its use of FEniCSx. It describes engineering problems in terms familiar from traditional CAE, but the model is ordinary Python that can be read, composed, and inspected as it runs. UFL and PETSc remain accessible below this layer. One model can grow from a single analysis into parameter studies and learning data, and people, scripts, graphical interfaces, or AI agents can all use it. Results retain their quality and origin, while capability records show the limits of the available methods. This combination defines the platform more than any single solver or constitutive law.

3.1 How an analysis is expressed

AgentFEM is used through ordinary Python. A case begins by declaring the type of study, then creates a model with its mesh, fields, materials, loads, and constraints. A solution step advances the model and returns a `SimulationResult`. This order follows the way a finite-element analyst describes a problem; assembly code is not required at the top level.

The same case can be run directly, submitted with MPI on a cluster, or managed through the AgentFEM command-line interface. The command-line tools add project checks, repeatable run records, and result inspection, but the case remains readable Python. A graphical interface or an AI agent can use this public model without translating it into a separate representation.

3.2 Layered organization

Behind this public workflow, figure 1 shows three layers:

1. **Engineering workflow layer.** The Python objects seen by users define the analysis and its requested output.

2. **Finite-element capability layer.** Reusable materials, operators, solution procedures, and checks implement the mechanics. A formulation can become a tested capability instead of remaining local code in one case.

3. **Numerical kernel and provider layer.** Adapters connect these capabilities to FEniCSx, UFL, PETSc, MPI, and optional packages.

Results connect the three layers and record what was computed. Advanced users can still reach UFL or backend objects when needed.

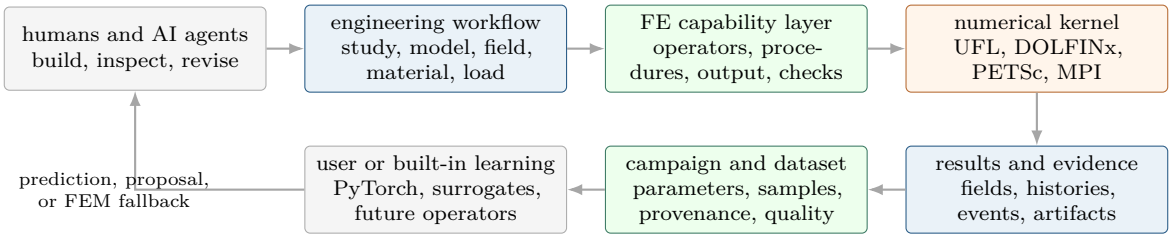

Figure 1. AgentFEM connects the engineering model to finite-element capabilities, numerical solvers, results, and learning workflows.

3.3 Current scope

The public release described here is version 0.2.1 (Luo, 2026) and targets Python 3.11 or newer. Supported analyses share the `model.step(...).solve_result()` lifecycle. Public interfaces are grouped into core, advanced, and expert levels so that users and tools can discover them gradually. Capability records state whether a path is implemented, partial, or exploratory. Because AgentFEM is developing rapidly, table 1 is a non-exhaustive snapshot rather than a fixed inventory.

Table 1. A non-exhaustive snapshot of implemented foundations and important development boundaries in the current release.

Area	Implemented foundation	Important current boundary
Solid mechanics	Linear elasticity, thermoelasticity, finite-strain hyperelasticity, global 3D J2 plasticity, and global power-law creep	Selected material paths; general contact, creep damage, and industrial validation remain future work
Dynamics and heat	Central difference, Newmark, generalized-α, transient heat integration, absorbing and periodic boundary models	Implicit structural dynamics remains linear; restart portability of constitutive integration-point state is capability-specific
Nonlinear execution	Automatic increments, Newton iteration, cutback, quadrature state, energy histories, and serial restart	Portable MPI restart of internal-variable state is not yet general
Interoperability	Abaqus C3D10/C3D10H mesh import, linear equation constraints, Gmsh and meshio pathways, XDMF/HDF5	User-material bridges and additional Abaqus set and mesh semantics are planned
Parallelism	MPI execution through DOLFINx/PETSc; distributed periodic equations with optional dolfinx_mpc	A campaign is serial within one runner; case-level parallelism currently uses deterministic sharding or an external scheduler
Simulation to learning	Deterministic campaigns, scientific datasets, NumPy/PyTorch adapters, ridge/POD/PyTorch baselines, domain guard and FEM fallback	Arbitrary-mesh neural-operator adapters and broader physics-informed workflows are next-stage extensions

The current capability records and documentation provide a more detailed and continuously updated view. Readers are invited to follow the project and contribute questions, cases, and technical feedback through the public repository or the correspondence addresses above.

3.4 From study to solution

A *study* identifies the physical problem and its admissible fields. The model holds the mesh, materials, and boundary data, while a *solution procedure* states how the problem is advanced. Separating these roles prevents a broad label such as "dynamic" or "nonlinear" from silently choosing an algorithm. Current procedures cover statics, transient heat transfer, implicit dynamics using Newmark or generalized-α integration (Newmark, 1959; Chung and Hulbert, 1993), and explicit central-difference dynamics.

At a lower level, operators represent contributions to the governing equations. A general semi-discrete form for second-order mechanics is

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{C}\dot{\mathbf{u}} + \mathbf{f}_{\text{int}}(\mathbf{u}, \boldsymbol{q}) = \mathbf{f}_{\text{ext}}(t) \tag{1}$$

where $\boldsymbol{q}$ denotes material state variables. For linear mechanics, $\mathbf{f}_{\text{int}} = \mathbf{K}\mathbf{u}$; in nonlinear mechanics, linearization introduces the tangent matrix $\mathbf{K}_{\text{t}} = \partial \mathbf{f}_{\text{int}} / \partial \mathbf{u}$. The general equation does not imply one solution strategy. An implicit procedure solves equilibrium at each increment, whereas an explicit procedure advances the state from the forces and mass. Standard providers handle common cases, and specialists may supply their own UFL operators.

3.5 Results that remain inspectable

All procedures return the same result type. A `SimulationResult` contains the computed quantities and reports whether the run converged and passed its requested checks. It also records the source of the data. An `OutputPlan` selects variables and write times, while a common event stream reports progress. Static, incremental, and transient analyses use these same interfaces.

Field names follow familiar finite-element conventions where possible: displacement `U`, stress `S`, small strain `E`, logarithmic strain `LE` at finite strain, deformation gradient `F`, first Piola stress `P`, volume ratio `J`, Mises stress, energy density, and element volume. Compact XDMF/HDF5 output preserves scientific fields and time coordinates, while optional presentation output constructs directly viewable deformed configurations and animations. Checkpoints are distinct from visualization files: they must preserve the state needed to continue an analysis. A field plot can therefore remain connected to the convergence and verification evidence behind it.

4 Executable case studies

The examples cover a minimal static model, an explicit wave problem, and an imported periodic cell at finite strain. All figures were regenerated with the repository code.

4.1 Minimal clamped beam

A 4×1 rectangular plane-stress domain is clamped at its left boundary and subjected to a downward traction at its right boundary. It is discretized by 80×20 bilinear quadrilaterals, with $E = 210$ GPa and $\nu = 0.3$. The case is deliberately elementary: its purpose is to expose the full top-level model without obscuring it behind an application-specific helper. The executable core is shown in listing 1; imports, plotting, and paper-data export are omitted.

Listing 1. Minimal engineering-level static workflow.

```
study = studies.static_solid(
    dimension=2, assumption="plane_stress")
domain = mesh.rectangle((0.0, 0.0), (4.0, 1.0), (80, 20),
                        cell_type="quadrilateral")
model = models.create(study=study, mesh=domain)
u = model.field(fields.displacement(domain, degree=1))
model.material(elasticity.isotropic_elastic(
    young=210e9, poisson=0.30, density=7800.0))

left = mesh.boundary(domain, lambda x: np.isclose(x[0], 0.0))
right = mesh.boundary(domain, lambda x: np.isclose(x[0], 4.0))
model.fix(u, on=left)
model.traction(value=(0.0, -20e6), on=right)
result = model.step(target=u).solve_result()
```

The code reads in the order in which an analyst describes the problem: analysis, domain, unknown, material, boundaries, loading, and step. The generic `model.step` dispatches from the declared study; a specialist may instead request a named provider or supply explicit operators. Figure 2 shows the generated model schematic and displacement field. The displayed deformation is scaled, and the recorded maximum displacement is 25.85 mm.

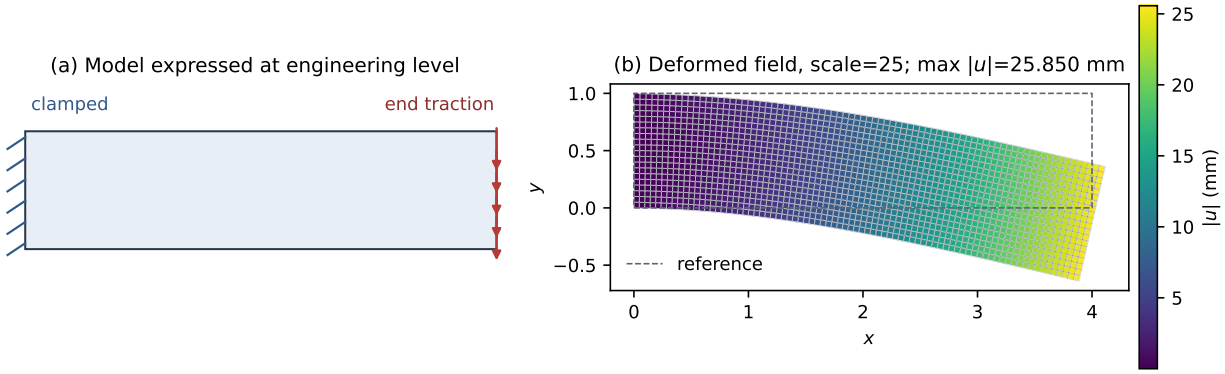

Figure 2. Minimal left-clamped, right-loaded beam. The reference outline is dashed in the field view; deformation is magnified for presentation.

4.2 Wave packet through a stiff inclusion

The dynamic example revisits a class of transient heterogeneous-wave problems studied in the author's doctoral work (Luo, 2020) and in a finite-element investigation of acoustic attenuation and thermal transport in two-dimensional nanophononic solids (Luo et al., 2019). The present example distils its characteristic ingredients into a compact workflow: a Gaussian-modulated displacement source, two material regions, periodic transverse (top and bottom) boundaries, an absorbing outlet, and explicit time integration.

The plane-strain plate has dimensions $1.2\,\mu\text{m} \times 0.24\,\mu\text{m}$ and a 240×48 quadrilateral mesh. A circular inclusion of radius $0.036\,\mu\text{m}$ is centred at one quarter of the plate length. Matrix properties are $E = 227.5$ GPa, $\rho = 2900\ \text{kg}\,\text{m}^{-3}$, and $\nu = 0.27$; the inclusion has the same density and Poisson ratio but twice the Young modulus. The source frequency is 40.78 GHz. Top and bottom are paired periodically, the right boundary uses a viscous absorbing model, and a central-difference procedure advances 2000 stable time steps.

This case shows how AgentFEM combines named material regions with a prescribed pulse, periodic boundaries, and an absorbing outlet. One explicit step controls the solution and its output. No problem-specific solver loop is required.

Figure 3 presents four frames of the longitudinal displacement. The run writes them to one XDMF/HDF5 time series and monitors the periodic mismatch, which remained zero to the implemented tolerance. The broader study that motivated this example is reported by Luo et al. (2019).

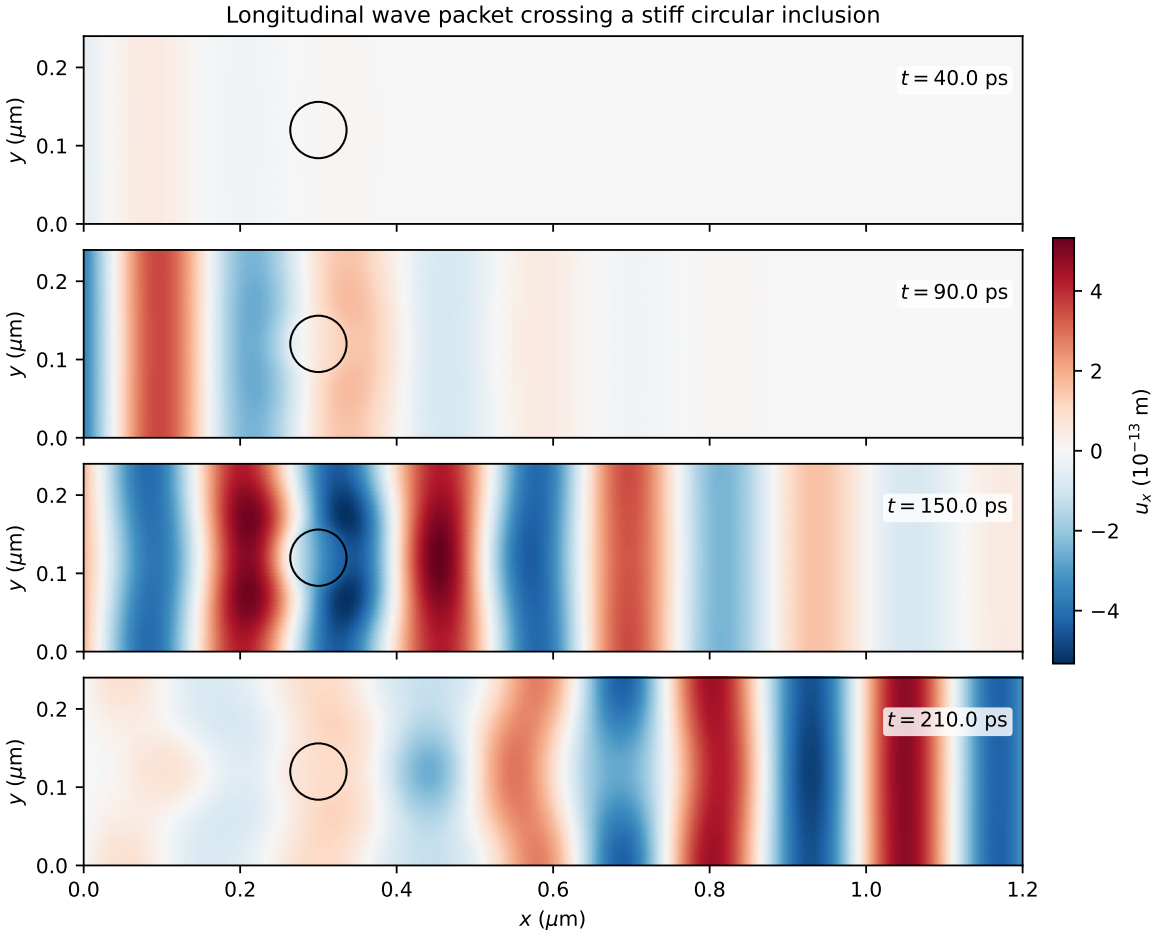

Figure 3. Longitudinal displacement snapshots for an explicit wave packet crossing a circular inclusion with twice the matrix stiffness. The black circle marks the inclusion boundary.

4.3 Finite-strain homogenization of an imported C3D10H porous cell

The third case demonstrates nonlinear solid mechanics rather than presenting a new micromechanics study. It imports an Abaqus-style ten-node tetrahedral mesh and its linear `*EQUATION` relations. Node labels and quadratic geometry are preserved, and the periodic equations become algebraic constraints in AgentFEM.

The matrix is described by a quasi-incompressible Neo-Hookean model with matrix shear modulus $\mu_m = 1$ and $\kappa/\mu_m = 10^4$. C3D10H is represented by a mixed quadratic-displacement/elementwise-constant-pressure space, so pressure is an independent unknown rather than a displacement-only penalty surrogate. Following the periodic uniaxial loading of Luo et al. (2023), the axial stretch is prescribed, the macroscopic shear components vanish, and the two lateral stretches are obtained from equilibrium:

$$\overline{\mathbf{F}} \simeq \lambda\, \mathbf{e}_1 \otimes \mathbf{e}_1 + \lambda_{\mathrm{lat}}(\mathbf{e}_2 \otimes \mathbf{e}_2 + \mathbf{e}_3 \otimes \mathbf{e}_3), \qquad \overline{\mathbf{S}} = S_{\mathrm{un}}\, \mathbf{e}_1 \otimes \mathbf{e}_1. \tag{2}$$

The corresponding three-reference-node controls remain in the executable case rather than being expanded into a boundary-condition manual here.

The reproduced run reached $\lambda = 1.20$ in three accepted increments, with three Newton iterations per increment and no cutback. The lateral stretches were 0.91786 and 0.91806. The two transverse first Piola components were below 2×10^{-11} in magnitude, and the maximum periodic-equation error was 1.4×10^{-17}. The output plan records the main finite-strain fields and homogenized histories. Figure 4 shows the reference and final configurations.

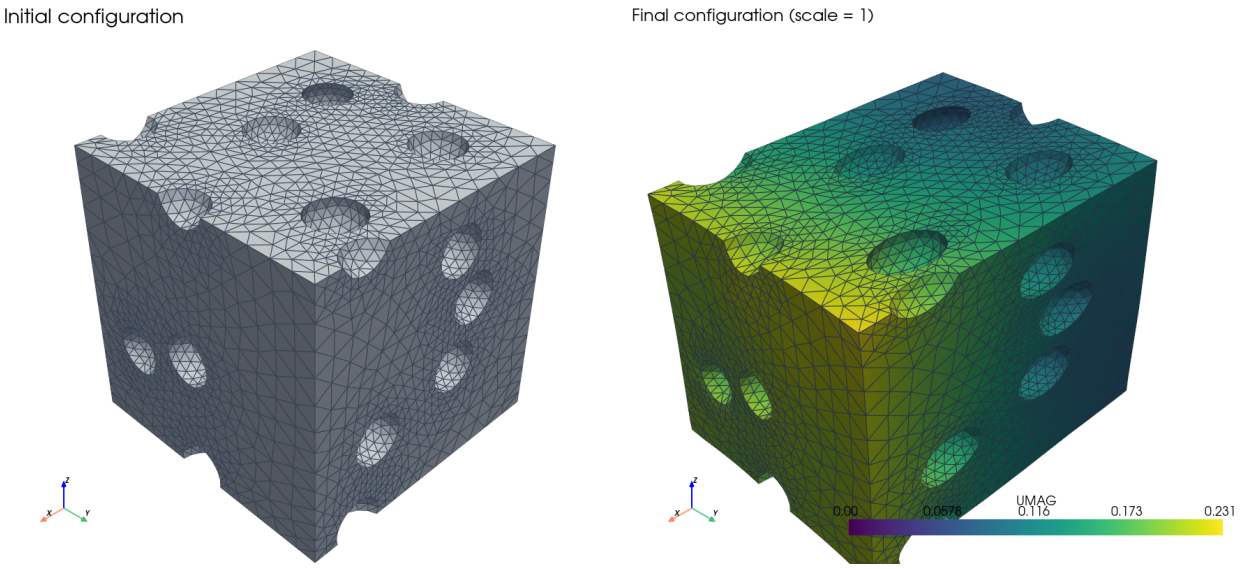

Figure 4. Imported C3D10H periodic cell before and after mixed quasi-incompressible finite-strain loading. The final configuration is coloured by displacement magnitude.

AgentFEM computes the macroscopic response directly from volume-integrated quadrature forms and the complete reference-cell volume. Following the notation of Luo et al. (2023), figure 5 reports the normalized energy W/μ_m against $I_1 = \mathrm{tr}(\overline{\mathbf{F}}^{\mathsf{T}}\overline{\mathbf{F}})$ and the normalized uniaxial first Piola stress S_{un}/μ_m against the applied stretch λ. These histories demonstrate automated recovery of finite-strain homogenized quantities.

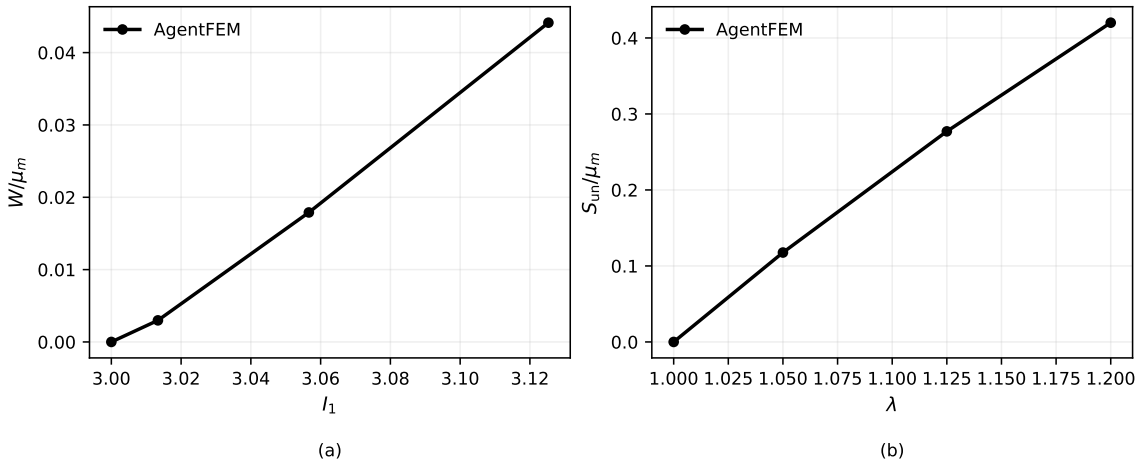

Figure 5. Normalized macroscopic quantities extracted by AgentFEM: (a) strain-energy density W/μ_m versus I_1 and (b) uniaxial first Piola stress S_{un}/μ_m versus applied stretch λ.

5 From simulation to learning

5.1 The data contract

Running many finite-element cases is straightforward; producing a trustworthy training set is not. Each sample must retain its parameters, result, solver status, and source. AgentFEM therefore treats the transition from simulation to learning as a data contract:

$$\mathcal{P} \longrightarrow \{\boldsymbol{p}_i\}_{i=1}^{N} \longrightarrow \{\mathcal{S}(\boldsymbol{p}_i), \mathcal{E}_i\}_{i=1}^{N} \longrightarrow \mathcal{D} \longrightarrow \widehat{\mathcal{S}}, \tag{3}$$

where $\mathcal{P}$ is the parameter space, $\mathcal{S}$ the simulator, $\mathcal{E}_i$ the evidence for one run, $\mathcal{D}$ the dataset, and $\widehat{\mathcal{S}}$ a learned approximation. Failed or unverified cases remain separate from accepted samples. The user chooses the required level of evidence.

A campaign defines its parameters and sampling plan, then records each case. MPI can accelerate a case, while an external scheduler distributes independent cases.

5.2 An implemented surrogate workflow

The campaign-to-surrogate path is implemented in the current release. Reviewed samples are stored in a `ScientificDataset`. The training function fits and validates a chosen estimator. The fitted model can call the original finite-element case outside its sampled domain:

Listing 2. From an AgentFEM campaign to a validated and guarded surrogate.

```
dataset = report.require_dataset(quality="engineering")
training = surrogates.train(
    dataset,
    estimator=surrogates.RidgeSurrogate(),
    validation_fraction=0.2,
    seed=2026,
)
predictor = training.guard(fallback=run_fem_case)
prediction = predictor.predict(candidate_parameters)
```

The current release includes `RidgeSurrogate` for scalar or small-vector responses, `PODRidgeSurrogate` for sampled fields, and an optional `TorchMLPSurrogate`. They use the same validation and FEM fallback. Users can also pass the dataset to their own model.

5.3 Using existing neural-network code

PyTorch remains the training framework, while AgentFEM supplies the simulation and data contract. A dataset can be read as NumPy arrays or converted to tensors and data loaders:

Listing 3. Handoff from finite-element campaign data to a user's PyTorch workflow.

```
dataset = datasets.ScientificDataset.read("trusted_dataset")
bundle = dataset.to_torch()
train_loader = bundle.loader(batch_size=64, shuffle=True)

for parameters, response in train_loader:
    prediction = my_existing_network(parameters)
    loss = loss_function(prediction, response)
    ...
```

Existing PyTorch code can be used without an AgentFEM-specific base class. AgentFEM supplies the simulation data and its record. Requests outside the sampled domain can be returned to finite-element analysis instead of being extrapolated silently.

5.4 Surrogates, neural operators, and physics-informed learning

For low-dimensional studies, a surrogate may map model parameters to selected responses. AgentFEM supports the path from the parameterized model to training, validation, and FEM fallback.

Neural operators such as the Fourier neural operator (Li et al., 2021) learn mappings between function spaces. AgentFEM defines field encodings and neural-operator specifications, but arbitrary meshes still require a consistent treatment of degrees of freedom, geometry, and boundary data.

Physics-informed neural networks (Raissi et al., 2019) can use governing residuals and boundary conditions during training. The current `PINNSpec` and optional `TorchPINNAdapter` connect these definitions to user PyTorch code. Automatic translation of arbitrary UFL forms into PINN losses is not yet implemented.

6 Discussion

The examples are modest, but they expose the main questions behind AI-native CAE. The issue is not simply whether an AI can write solver code. It is where the meaning of a model is kept, how finite-element knowledge is reused, and how responsibility is preserved as automation grows.

6.1 What AI-native changes

AI-native CAE is more than a conventional solver placed behind a conversational interface. The engineering model must be directly usable by both people and agents, while the numerical calculation remains deterministic. An agent may prepare or revise a case, but the accepted case can still be inspected and run without that agent.

This separates the CAE system from any particular interface. The same AgentFEM case may run as a script, on a cluster, in an AI-enabled environment, or through forms of interaction that have not yet settled. The model and its results remain the scientific state, while interfaces can change around them.

AgentFEM makes this distinction practical. In recent work, the author has used a coding agent to carry complete tasks from building cases and running batches to preparing datasets, inspecting results, and connecting a user-defined neural network. A versioned AgentFEM skill and detailed documentation guide the work. During a run, stable interfaces report failed model checks, solver progress, and results in a form the agent can use. The agent therefore works with the same finite-element concepts as a human user and leaves an inspectable record. The *agent harness*—the layer that connects reasoning to tools, files, computation, feedback, and evidence—is as important as the conversational interface itself (OpenAI, 2026).

6.2 "Intelligence flow" and productive tools

Successive industrial revolutions were organized around new flows: steam power, electricity, and information. Their influence did not come only from producing these resources more efficiently. It grew through engines, electrical appliances, computers, and networks that turned a general resource into useful work. In a similar sense, we use the term *"intelligence flow"* for model inference delivered and consumed through tokens. Better models, algorithms, and computing systems improve how that flow is generated. A broader opportunity lies on the other side: designing tools that use it to solve real problems and expand productive capacity.

Agents then become more than advisers. They can act as producers and users of scientific tools, provided that those tools make scientific objects directly accessible, report their state clearly, and preserve evidence. AI-native CAE is one class of such productive instrument. AgentFEM is designed so that an agent can work through finite-element concepts rather than operate a human interface indirectly or replace mechanics with generated code. The same principle extends beyond CAE: the value of increasingly abundant intelligence will depend on the instruments through which it enters science, engineering, and production.

6.3 Accumulating finite-element knowledge

AI will make code easier to produce, but generated scripts do not by themselves create a dependable engineering system. A useful formulation must survive beyond the case in which it first appeared. In AgentFEM, it can become a tested material, operator, solution procedure, or result definition that later cases can reuse.

An individual function or constitutive equation is relatively easy to reproduce. It is harder to keep its engineering interface, state update, solver behaviour, output, documentation, and benchmarks consistent as the software evolves. This is how progress can accumulate one verified capability at a time. The resulting ecosystem is defined by these connections rather than by any single feature.

6.4 From CAE with AI to AI-native CAE

Abaqus, Ansys, and other established CAE systems provide broad capabilities and long records of industrial use. Many are adding AI-assisted functions that simplify modelling or automate particular operations. These are useful developments, but adding intelligence around an architecture built for human use is better understood as CAE with AI than as AI-native CAE.

Their core implementation and much of the model state remain proprietary or are exposed through product-specific databases and interfaces. Input files and scripting provide important access (Dassault Systèmes, 2024; Ansys, Inc., 2026), but an agent may still have to coordinate several interfaces or operate a graphical application indirectly. This can improve existing functions, but it is less direct than working with a scientific model designed to be read, changed, executed, and inspected by both people and agents.

AgentFEM makes the readable Python case the primary model. Its implementation and documentation remain open, and its runs can be inspected. It still reuses established assets: the C3D10H example imports an Abaqus mesh and its equation constraints. The same distinction extends beyond CAE. As agents become users and producers of software, many tools may need to expose their state, accept programmable operations, report problems clearly, and preserve human review rather than receive AI only as an added feature.

6.5 Human responsibility and present limits

Agents can increase the number and speed of calculations, and they can also repeat a wrong assumption at the same scale. The distinctions among computed, converged, verified, and validated results are therefore central to the design. People remain responsible for the scientific question, the model, the evidence, and the use of the result. AgentFEM's role is to keep these decisions visible while automating routine work.

AgentFEM remains an evolving research and engineering platform rather than a claim of universal CAE coverage. Its capabilities have different levels of maturity, which the software records explicitly. Development favours deeper implementation and verification of useful paths while allowing experimental ideas to remain visible as directions for further work.

6.6 From AI-native CAE to AI-native research

AgentFEM is one example of a broader class of AI-native scientific tools. Such tools can allow AI to participate in a continuous research process—from literature and study design to simulation, analysis, revision, and the search for new results. As AI becomes more capable, it can undertake more of this work, explore more possibilities, and communicate findings more rapidly. AI-native scientific tools can therefore contribute to an explosive growth of scientific discovery. The purpose is not to remove people from research, but to give human curiosity and judgement more capable instruments.

7 Conclusion

This paper presented AgentFEM as an open-source AI-native CAE platform for humans and agents. Its Python language keeps the engineering model visible, while FEniCSx, UFL, PETSc, and MPI provide the numerical foundation. Common results connect the analyses to reproducible output and learning-ready data.

Three cases showed the current design: a minimal static model, an explicit wave problem, and an imported C3D10H periodic cell under quasi-incompressible finite-strain uniaxial loading. The next stage will deepen verification, inelastic analysis, restart, and mesh interoperability while preserving a simple top-level language.

AgentFEM's practical goal is to become a useful finite-element tool in selected domains. Its broader direction is that CAE should allow people and AI to share the same scientific model. The future value of AI in CAE will depend not only on how much code it can write, but on whether that work becomes reliable and reusable scientific knowledge.

Feynman left a sentence on his final blackboard at Caltech: "What I cannot create, I do not understand" (Alda, 2002). AI may make many scientific discoveries, but discovery is not the same as human understanding. At least in the near term, people must interpret what AI finds and give it meaning. AI can also help us learn through that process, deepening rather than diminishing human understanding.

What of the longer term? This raises a deeper philosophical question about humanity's place in the age of intelligence. When intelligence itself is decomposed by computation and everything enters a torrent of 0s and 1s, will humanity remain the measure of all things? Yet I believe that, somewhere in the world, there will always be scientists working to show that the ultimate destination of computation is the exploration of the deepest connection between consciousness and matter.

Software and data availability

AgentFEM is available under the Apache License 2.0 at `https://github.com/haoming-luo/agentfem`. The release described in this paper is version 0.2.1. The paper project includes scripts used to regenerate the figures, together with simulation-derived data. For manual inspection, the C3D10H driver, source inputs, and complete result directory are also archived under `simulations/c3d10h_periodic_cell`; the corresponding public example lives in the AgentFEM repository under `examples/abaqus_c3d10h_periodic_cell`.

Acknowledgements

The author gratefully acknowledges the education and research experience in computational mechanics gained at NWPU, INSA Lyon, and École Polytechnique. The author also thanks Xi'an Thermal Power Research Institute (TPRI) for providing the research platform and acknowledges support from project HNKJ25-H111. The author further acknowledges the open-source communities that develop FEniCSx, PETSc, MPI, and the broader Python scientific ecosystem.

A Reproducibility notes

The paper figures use simulation outputs rather than hand-drawn fields. The beam script also constructs its figure with Matplotlib. The wave plotter selects four times from the generated XDMF/HDF5 series. The hyperelastic figures are generated from accepted nonlinear increments. Plotting scale factors do not change the stored values.

For the periodic cell, the key high-level step is:

Listing 4. Finite-strain step with automatic increments and a declared output plan.

```
incrementation = steps.automatic(
    initial=0.25, minimum=1e-4, maximum=0.50,
    max_increments=10, max_cutbacks=5)
step = model.step(
    target=unknown,
    material=material,
    constraints=periodicity,
    incrementation=incrementation,
    output=output_request,
    solver_options=solvers.newton(
        relative_tolerance=1e-7,
        absolute_tolerance=1e-8,
        maximum_iterations=25,
        linear_solver=solvers.direct_solver(package="mumps")))
result = step.solve_result()
```

The complete code defines the mixed field, C3D10H formulation, reference-point controls, output, and result records.

References

Alan Alda. Finding Feynman. *Engineering & Science*, (2):17–21, 2002. URL `https://calteches.library.caltech.edu/684/2/Feynman.pdf`.

Martin S. Alnæs, Anders Logg, Kristian B. Ølgaard, Marie E. Rognes, and Garth N. Wells. Unified form language: A domain-specific language for weak formulations of partial differential equations. *ACM Transactions on Mathematical Software*, 40(2):1–37, 2014. doi: 10.1145/2566630.

Ansys, Inc. PyAnsys for developers. Ansys Developer Portal, 2026. URL `https://developer.ansys.com/docs/pyansys`.

Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object-oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools for Scientific Computing*, pages 163–202. Birkhäuser, Boston, 1997. doi: 10.1007/978-1-4612-1986-6_8.

Igor A. Baratta, Joseph P. Dean, Jørgen S. Dokken, Michal Habera, Jack S. Hale, Chris N. Richardson, Marie E. Rognes, Matthew W. Scroggs, Nathan Sime, and Garth N. Wells. DOLFINx: The next generation FEniCS problem solving environment. Preprint, 2023.

J. Chung and G. M. Hulbert. A time integration algorithm for structural dynamics with improved numerical dissipation: The generalized-α method. *Journal of Applied Mechanics*, 60(2):371–375, 1993. doi: 10.1115/1.2900803.

Commissariat à l'énergie atomique et aux énergies alternatives. The concepts of named objects, operators, and the Gibiane language. Cast3M documentation, 2026. URL `https://www-cast3m.cea.fr/index.php?notice=intk&page=notices_classees`.

Dassault Systèmes. Abaqus input syntax rules. Abaqus documentation, 2024. URL `https://docs.software.vt.edu/abaqusv2024/English/SIMACAEMODRefMap/simamod-c-inputsyntax.htm`.

Rushikesh Deotale, Adithya Srinivasan, Yuan Tian, Tianyi Zhang, Pavlos Vlachos, and Hector Gomez. ALL-FEM: Agentic large language models fine-tuned for finite element methods. *arXiv preprint arXiv:2603.21011*, 2026.

Jiachen Guo, Chanwook Park, Dong Qian, Thomas J. R. Hughes, and Wing Kam Liu. Large language model-empowered next-generation computer-aided engineering. *arXiv preprint arXiv:2509.11447*, 2025.

Alexander Hermann, Arman Shojaei, Ingo Scheider, and Christian Cyron. OASiS: An open-source multi-physics and multi-code framework for verified computer simulations. Software release, 2026. URL `https://github.com/Hereon-InstituteMS/OASiS`.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=c8P9NQVtmnO`.

Haoming Luo. *High Frequency Thermomechanical Study of Heterogeneous Materials with Interfaces*. PhD thesis, Université de Lyon, 2020.

Haoming Luo. AgentFEM: An ai-native open-source platform for finite-element computing. Software, 2026. URL `https://github.com/haoming-luo/agentfem`. Apache-2.0 licensed.

Haoming Luo, Anthony Gravouil, Valentina Giordano, and Anne Tanguy. Thermal transport in a 2d nanophononic solid: Role of bi-phasic materials properties on acoustic attenuation and thermal diffusivity. *Nanomaterials*, 9(10): 1471, 2019. doi: 10.3390/nano9101471.

Haoming Luo, Zahra Hooshmand-Ahoor, Konstantinos Danas, and Julie Diani. Numerical estimation via remeshing and analytical modeling of nonlinear elastic composites comprising a large volume fraction of randomly distributed spherical particles or voids. *European Journal of Mechanics A/Solids*, 101:105076, 2023. doi: 10.1016/j.euromechsol.2023.105076.

Nathan M. Newmark. A method of computation for structural dynamics. *Journal of the Engineering Mechanics Division*, 85(3):67–94, 1959.

Bo Ni and Markus J. Buehler. MechAgents: Large language model multi-agent collaborations can solve mechanics problems, generate new data, and integrate knowledge. *arXiv preprint arXiv:2311.08166*, 2023.

OpenAI. Harness engineering: Leveraging Codex in an agent-first world. OpenAI engineering article, 2026. URL `https://openai.com/index/harness-engineering/`.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Yupeng Qi, Ran Xu, and Xu Chu. FeaGPT: An end-to-end agentic-AI for finite element analysis. *arXiv preprint arXiv:2510.21993*, 2025.

Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. doi: 10.1016/j.jcp.2018.10.045.

Matthew W. Scroggs, Igor A. Baratta, Chris N. Richardson, and Garth N. Wells. Basix: A runtime finite element basis evaluation library. *Journal of Open Source Software*, 7(73):3982, 2022. doi: 10.21105/joss.03982.

Zhenghan Song, Yulong Liu, Cheng Wan, Chenjun Li, Lingfu Liu, Yunyi Li, and Congcong Yuan. Your simulation runs but solves the wrong physics: PDE-grounded intent verification for LLM-generated multiphysics simulation code. *arXiv preprint arXiv:2605.09360*, 2026.

Nilay Upadhyay and Wesley F. Reinhart. A constrained natural-language interface for variational multi-physics finite element simulations in FEniCS. *arXiv preprint arXiv:2606.10928*, 2026.