Why FeenoX is different

Jeremy Theler

2025-12-01-26f4122

Abstract

This white paper answers the question "why is FeenoX different?" First of all, we have to say what FeenoX is. Then, we must state what we are comparing FeenoX against so we can finally explain why it is indeed different. TL; DR: FeenoX is different because there is nothing else that matches 100% FeenoX's design and implementation.

1 What FeenoX is

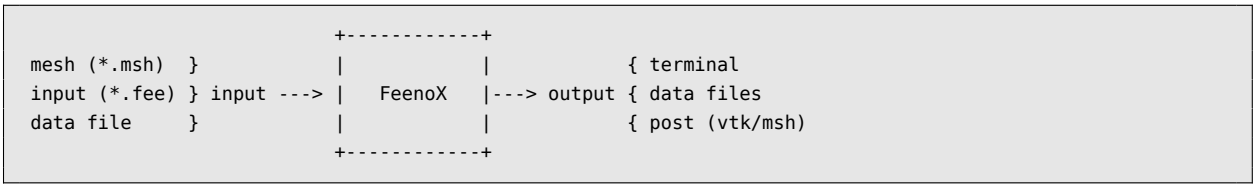
As explained in its website, FeenoX is…

> …a cloud-first free no-fee no-X uniX-like finite-element(ish) computational engineering tool.

or, in similar words,

> …a tool to solve engineering-related problems using a computer (or many computers in parallel) with a particular design basis.

We will go into (some) of the details in sec. 3, but it is helpful to think of FeenoX as a transfer function between one or more input files and zero or more outputs:



Here, FeenoX is a binary executable which can be easily installed in a Debian or Ubuntu GNU/Linux server (remember it is a **cloud-first** tool) with

```
apt install feenox
```

Let us consider for instance the following input file that solves linear elasticity:

```
PROBLEM mechanical MESH box.msh

T(x,y,z) = sqrt(x^2 + y^2) + z   # non-trivial temperature distribution
E = 200e3 * (1-T(x,y,z)/300)     # temperature-dependent Young modulus
nu = 0.3

BC left fixed           # left face fully fixed
```

```
BC top  p=1e4*(1+x)     # top face with a trapezFoidal load

WRITE_RESULTS FORMAT vtu
```

Some preliminary notes:

- The input file is made of simple self-descriptive English keywords
  - Note that if one asks an LLM to create an input—which is a straightforward task—hallucinations are easily detected.
- The type of problem being solved is chosen with the `PROBLEM` keyword at runtime. The same executable can solve several problems, namely (so far in version 1.2 as the time of this writing)
  - `laplace`: the Laplace equation
  - `thermal`: heat conduction
  - `mechanical`: mechanical elasticity
  - `modal`: mechanical modal analysis
  - `neutron_diffusion`: core-level multi-group neutron diffusion
  - `neutron_sn`: core-level multi-group neutron transport with discrete ordinates
- Material properties and boundary conditions can be expressions of x, y and z
- Input mesh is in Gmsh's format `.msh`
- Output results are in Paraview's `.vtk`/`.vtu` formats

2 What can FeenoX do that cannot be done with Tool $\mathcal{X}$?

In principle, nothing. FeenoX provides a Turing-complete system so theoretically any computable problem can be solved. Chances are that Tool $\mathcal{X}$ also provides a Turing-complete system as well. So the question is ill-posed. The real question we need to ask is:

> How much engineering effort is needed in order to solve problem y with FeenoX and with Tool $\mathcal{X}$?

Well, I do not know anything about Tool $\mathcal{X}$. But I do know everything about FeenoX and first of all, FeenoX is free[1] software and open source so, again in principle, any engineer has the freedom to modify (or the freedom to hire someone to modify) the code to suit her needs.

But secondly and more importantly, FeenoX has been designed and implemented in such a way that solving problem y can be performed by different engineers as flexibly and efficiently as possible (i.e. in different ways for each engineer). Let us dive into some details in the next section.

> For instance, given the question "what is the most efficient way of solving a maze?" there might be people that just say "Dijkstra" and move on. These are the same people that think they know more than they do and do not know what they do not know—especially regarding engineering challenges—as illustrated in FeenoX's tutorial about solving mazes without AI.

3 Design and implementation

FeenoX is a third-system effect. A first idea came up in the late 2000s. Since it had some good features, a second version with a lot of features appeared in the early 2010s. After almost one decade later, with all

[1] As in *freedom*, not as in *price*.

the lessons learned, the third version—named FeenoX—was re-written from scratch as a subject of a PhD thesis in Nuclear Engineering "Neutron transport in the cloud."[2]

The design went through a pattern which is common in software development, namely

1. A customer writes a document named "Software Requirements Specification" which acts as a request for quotation of an alleged tender where the requirements that the provider has to fulfill are clearly stated and explained.
2. One or more providers write a document named "Software Design Specification" that acts as an technical offer for the potential tender where each customer requirement is addressed and the way the provider proposes to design and implement each feature is explained.

For FeenoX, these two documents—the SRS and the SDS—were written by the same person. But nevertheless, they help organize and understand the rationale behind the design and the implementation.

The main items of FeenoX's design basis are

1. Cloud-first remote execution
2. Scalability with distributed-memory parallelization (MPI)
3. Self-descriptive English keyword-based plain-text input file
4. Scriptability and traceability
    a. Parametric and optimization studies
5. Straightforward integration with
    a. Web-based UX/UIs
    b. Large Language Models
    c. Rest APIs
    d. Standard post-processing tools
    e. Markdown and LaTeX reporting

Technically speaking, FeenoX is a back end which can use zero or more front ends (fig. 1). These front ends can range from a graphical user interface that creates a proper input file given the user point-and-click choices down to an agent using an augmented Large Language Model to create the input from a problem formulated in natural (engineering) language.

Due to the way FeenoX is designed to read a text-based input file (and eventually mesh and data files), it is perfectly suited to be run in remote cloud servers. The server just needs to get the `feenox` executable, either by (from easier and less flexible to more complex and more flexible)

- installing it from the distribution's APT repositories with `apt install feenox`, or
- downloading a pre-compiled binary for the proper architecture, or
- downloading a source tarball and compiling it (potentially with specific optimization options), or
- cloning the Git repository, bootstrapping and compiling it locally (with even more options).

and either get or create on the fly the input and the mesh files. These are actual files that can be uploaded from a client to a server, copied from server to server and/or stored in any kind of medium until needed. The same idea applies to the output, which being a set of files, can be downloaded, copied or stored at will. There are no binary, database nor proprietary lock ins. Hence, the implementation of a remote REST API to handle file transfers from the client to the server back and forth with a (possibly parallel) execution of `feenox` in the middle is (almost) straightforward.[3]

[2] Due to local law the text of the thesis had to be written in Spanish. Yet nowadays, it should be pretty straightforward to translate the text into English.

[3] Especially in times of AI-assisted programming, a.k.a. "vibe coding."

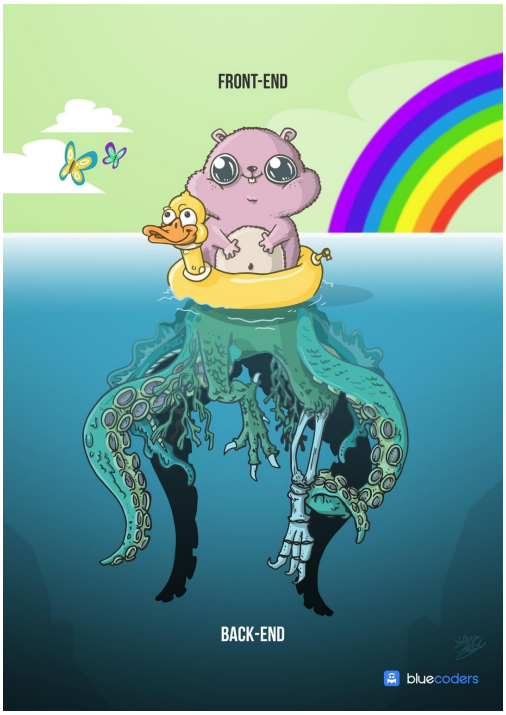


Figure 1: Conceptual illustration of the difference between a front end and a back end ©bluecoders.

> The last three paragraphs illustrate item 1 (cloud-first remote execution) and 5.c (REST APIs).

Since the algebraic solvers in FeenoX are based on PETSc—if the server has enough computational power– it can be launched in parallel using the MPI standard. This architecture allows a large range of problem sizes to be tackled, from simple coarse cases up problems to millions of degrees of freedom since—as long as the server is part of an HPC cluster—the domain can be split not only among several processors but also among several nodes. This "distributed memory parallelization" approach is not bounded by the maximum memory of a single host as the "shared memory parallelization" scheme used by OpenMP.

> The last paragraph illustrates item 2 (scalability with distributed-memory parallelization).

A great deal of effort was put into designing the syntax of the input files. The SDS discusses several "rules" for this syntax, but one of the most important ideas is the one from Alan Kay: "Simple things should be simple, complex things should be possible." Another one is that the input file should try to match as closely as possible the "engineering" formulation of the problem. The inputs needed to solve two NAFEMS benchmark problems in fig. 2 and fig. 3 illustrate these concepts.

> The last paragraph and figures 2-3 illustrate item 3 (self-descriptive English keyword-based plain-text input file).

Since the input files are based on English words and algebraic expressions, they are amenable to be traced by version control systems like Git. And since the mesh files come from Gmsh (actually from any other mesher or converter that can write in `.msh` format) that also uses English keywords in its input `.geo` language

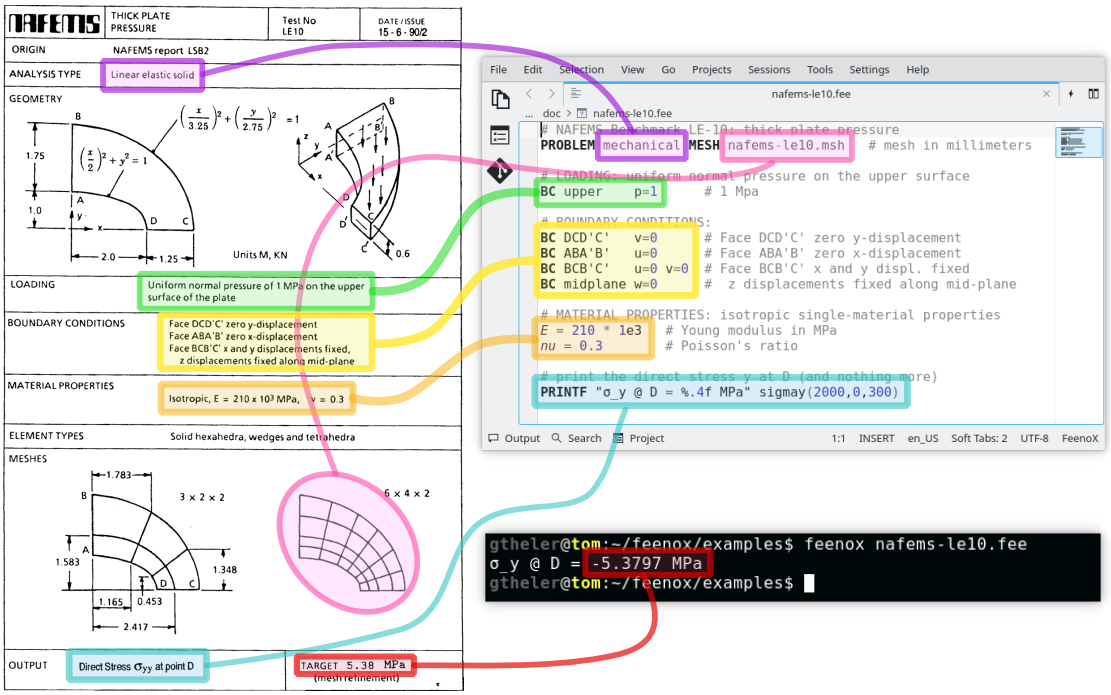


Figure 2: One-to-one correspondence between the “engineering” formulation and the FeenoX input file for the NAFEMS LE10 problem.

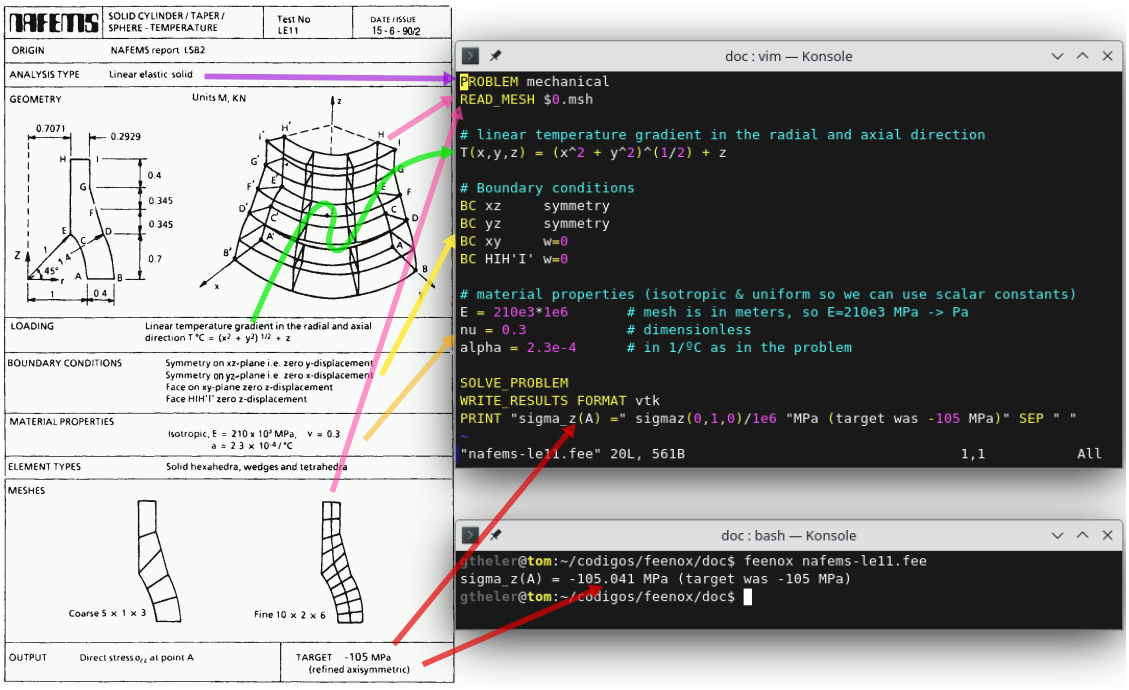


Figure 3: The NAFEMS LE11 problem asks for an algebraically-defined temperature distribution which can be written as an ASCII expression in the input file.

(or in its Python API), in principle the set of files that completely define a problem can be tracked with Git-like tools. Even more, FeenoX's source code is also tracked with Git and the `feenox` executable reports its version and checksum when invoked with `-v` so the same set of results can be deterministically obtained:

```
$ feenox --versions
FeenoX v1.2.3-gf95b352
a cloud-first free no-fee no-X uniX-like finite-element(ish) computational engineering tool

Last commit date   : Mon Sep 22 13:17:58 2025 -0300
Build architecture : linux-gnu x86_64
Compiler version   : gcc (Debian 14.2.0-19) 14.2.0
Compiler expansion : gcc -Wl,-z,relro -Wl,-z,now -I/usr/lib/x86_64-linux-gnu/mpich/include -L/usr/lib/ ↩
     x86_64-linux-gnu/mpich/lib -lmpich
Compiler flags     : -O3 -flto=auto -no-pie
GSL version        : 2.8
SUNDIALS version   : N/A
PETSc version      : PETSc Development Git Revision: v3.24.1-181-gbad27f0b03b Git Date: 2025-11-24  ↩
     16:54:37 +0000
PETSc arch         : arch-linux-c-debug
PETSc options      : --download-mumps --download-scalapack --download-slepc --with-bison=0 --with-c2html ↩
     =0 --with-debugging=1 --with-x=0 --download-parmetis --download-metis
SLEPc version      : N/A
```

A set of scripts calling

a. the mesher (or other pre-processing tools),
b. `feenox`
c. post-processing tools

can help to get systematically traceable results. And even more so, since the output from FeenoX is 100% determined by the user to the extent that without instructions to create outputs (i.e. `PRINT`, `WRITE_RESULTS`, etc.) then there will not be any output whatsoever. There is no need to for complex regular expressions or weird filter to uncover numerical results buried inside thousands of useless lines of data. And since the post-processing formats are standard (VTU/VTK or `.msh`) then there are plenty of high-quality (and most of the time open source) libraries for further post-processing the results and obtaining just the numbers that the engineer needs, giving more importance to engineering time over computer time. Plus, again since the output is user-defined, programmatically creating plots, tables and even full engineering reports is straightforward.

Moreover, FeenoX can expand command-line arguments into strings in the input file which—since everything is an expression—may be parsed as numerical values. For example, consider the following input that solves a rectangular cantilevered beam and computes the difference of the maximum displacement with the value predicted by Euler's theory:

```
PROBLEM mechanical 3D MESH cantilever-$1-$2.msh

E = 2.1e11          # Young modulus in Pascals
nu = 0.3            # Poisson's ratio
F = 1000

BC left   fixed
BC right  Fz=-F  # traction in Pascals, negative z

# compute the vertical displacement using Euler's theory
L = 0.5    # length
b = 0.05   # base
```

```
h = 0.02   # width

I = b*h^3/12
euler = -F*L^3/(3*E*I)

# error in z-displacement (components are u,v,w) at the tip vs. number of nodes
PRINT nodes %e 100*abs((w(L,0,0)-euler)/euler) "\# $1 $2"
```

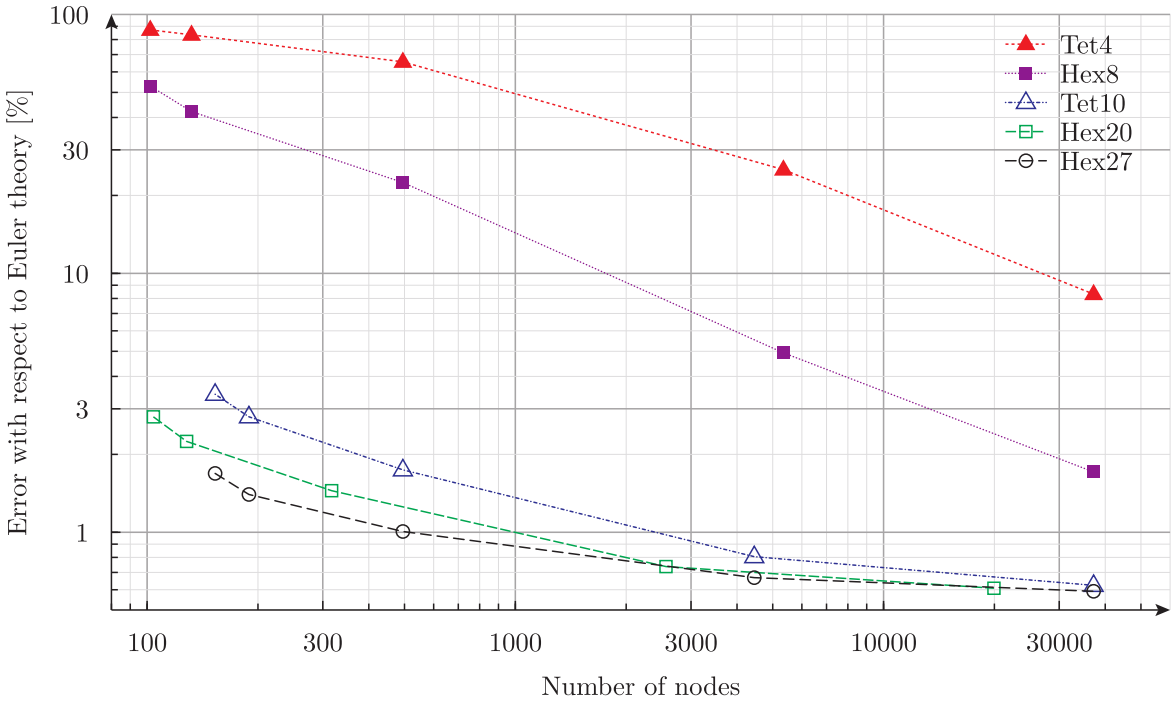


Figure 4: Parametric study of the difference between FEA and Euler's beam theory.

The occurrences of `$1` and `$2` in the input file are expanded to the arguments given in the command line after the path to the input file. Hence, if we call Gmsh & FeenoX as

```
for e in tet4 tet10 hex8 hex20 hex27; do
  for c in 1 0.75 0.5 0.2 0.1; do
    gmsh -3 cantilever-$e.geo -clscale $c -o cantilever-$e-$c.msh -v 0
    feenox beam-euler.fee $e $c | tee -a beam-euler-$e.dat
  done
done
```

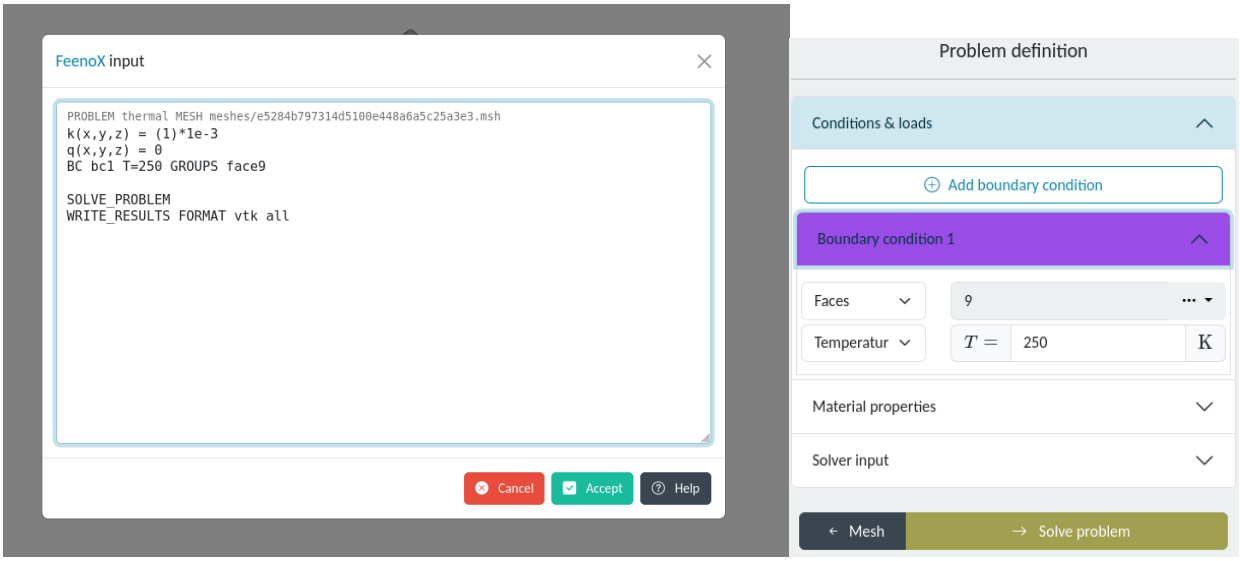
we can perform a parametric mesh convergence study. A result similar to fig. 4 can be easily created by standard open-source tools such as Gnuplot, Pyxplot or Matplotlib. Also, the combination of these parametric-ready features and the possibility of having sources (and material properties and boundary conditions) as algebraic expressions allow the verification of the solver with the Method of Manufactured solutions. Finally, by replacing a pre-defined parametric run with an optimization scheme one can smartly iterate over the parameter space to obtain optimal engineering designs.

> The last four paragraphs illustrate item 4 (parametric and optimization studies) and 4.a (scriptability and traceability).

This design philosophy also enhances the creation of several front ends to match different needs. For instance, CAEplex is a web app that is integrated into the cloud-base CAD tool Onshape.

SunCAE is another front end that provides an open-source web interface that helps the user to create the mesh the CAD, create FeenoX's input file, run the program and post-process the results. Even more, the input file can be edited online from the browser and, if the changes are simple enough, the web interface updates automatically. So for instance, if a boundary condition for a thermal problem says $T = 300$ K then the input file would look like

```
BC bc1 T=300 GROUPS face9
```



(a) FeenoX input file in the online editor (b) Updated interface with the new value

Figure 5: Illustration of the two-way connection between the UX and the input file in SunCAE

But if the user changes it to 250 and clicks "Accept" then the interface will show the updated value (fig. 5). Further examples of FeenoX usage through SunCAE:

- Overview & Tutorial
- Tutorial #2: NAFEMS LE10
- Tutorial #3: heat conduction

> The last three paragraphs illustrate item 5.b (straightforward integration with web-based UX/UIs).

The fact that `feenox` is an executable (instead of a library) and that it reads the problem to be solved from an input file at runtime (instead of having to recompile each time) allows LLMs to assist the cognizant engineer in the process of stating the problem formulation in FeenoX's terms. Because the input syntax is designed to look like engineering textbook descriptions (e.g., `BC right T=300`), LLMs can generate valid FeenoX inputs with high accuracy. Unlike proprietary "input decks" full of magic numbers, or GUI-based workflows that are invisible to an LLM, FeenoX provides a semantic bridge between natural language and computational physics. Furthermore, the feedback loop is tight: an LLM can write an input file, run FeenoX, capture the standard error output, detect syntax errors (e.g., a missing variable), correct the input, and run it again—all without human intervention. This enables "Agentic workflows" where the AI acts as a junior engineer, performing the setup and initial debugging before the human reviews the results.

> The last paragraph illustrates item 5.a (straightforward integration with LLMs).

This repository is an example of Kay's idea that complex thing should be possible. It contains an example of scripts and input files that estimate environmentally-assisted fatigue in piping joints of a nuclear power plant using

- transient heat conduction to evaluate temperature distributions in the bulk of the pipes,
- stress linearization to compute principal and secondary stress intensities according to ASME III,
- modal analysis to incorporate seismic loads through floor response spectra,
- the EPRI methodology to estimate environmentally-assisted fatigue at stress classification lines.

Everything is Git-tracked and all the results are deterministically computed by running a single script `run.sh`. By using specific LaTeX macros, reports containing references to isometric blue prints, time-dependent plots, spatial temperature distributions and result tables can be created programmatically (such as the one shown in fig. 6) without needing error-introducing manual interventions.

> The last two paragraphs illustrate item 5.e (straightforward integration with Markdown and LaTeX reporting).

4 The Unique Value Proposition

Taking into account the features discussed so far, it is left to the reader to consider the differences that FeenoX proposes with respect to traditional computational codes. In particular, to think about how the workflows in conceptual and basic engineering design of a nuclear reactor can be improved by being able to model all the physics involved in a nuclear reactor

a. within the same general modeling paradigm, and
b. using the particular paradigm proposed by FeenoX.

4.1 Unfair advantages

Traditional computational codes in the nuclear industry…

- are not designed to be executed remotely through web interfaces (making it harder for potential users to test the software)
- are not prone to be executed on remote servers (one has to buy hardware instead of renting)
- mix the front and the back ends (leading to pretty inflexible workflows)
- do not have an explicit input file or API (hard to track inputs)
- rely on binary data files (untraceable results)
- use numerical input "decks" (LLMs are limited if not useless)
- lack mechanisms for performing parametric studies (leading to complex and cumbersome optimization schemes)
- insist in interactive interfaces limiting the scripting capabilities (forcing mandatory error-prone human intervention)
- parallelize the execution with SMP which is limited by a single-node RAM (instead of using DMP which is not bounded)
- do not provide a generic way of giving non-trivial material properties and boundary conditions (while in FeenoX everything is an expression and constants are particular cases)
- can only be used by one single limited UI (which usually looks old and clumsy)

Pair A	Pair B	Applied Cycles A	Applied Cycles B	M+B STRESS (psi)	K_e	Total Stress (psi)	S_{alt} (psi)	N_n	n_n	U_n	Max. Metal Temp. (°F)	DO (ppm)
694	447	5	20	125542.9	2.580	144164.4	220490.4	140.005	5	0.0357	566.6	0.150
699	447	50	15	121622.8	2.405	139047.0	198300.6	178.958	15	0.0838	566.6	0.550
699	1021	35	20	104691.5	1.653	126037.5	124507.0	582.468	20	0.0343	600.4	0.550
699	899	15	50	89695.4	1.000	102302.8	57864.5	6339.47	15	0.0024	336.1	0.550
695	899	5	35	84993.9	1.000	98798.6	55882.4	7027.83	5	0.0007	336.1	0.550
185	899	20	30	68222.2	1.000	76465.1	43250.2	15549.1	20	0.0013	336.1	0.550
1432	899	20	10	66665.7	1.000	83098.8	47002.3	11892.7	10	0.0008	336.1	0.550
1432	1653	10	100	49437.0	1.000	61950.9	33687.5	35734.8	10	0.0003	103.0	0.522
1296	1653	20	90	32478.6	1.000	38719.1	22025.4	154852	20	0.0001	366.2	0.522
1136	1653	20	70	27045.6	1.000	33751.1	19388.7	258499	20	0.0001	417.7	0.522
2215	1653	100	50	25255.9	1.000	25668.1	15147.6	1.15E+06	50	0.0000	547.0	0.522
2215	1213	50	20	22343.7	1.000	25298.3	14929.4	1.30E+06	20	0.0000	547.0	0.050
2215	1562	30	20	22047.7	1.000	24970.1	14735.7	1.46E+06	20	0.0000	547.0	0.050
2215	1	10	20	11956.0	1.000	12255.6	7232.5	1.00E+11	10	0.0000	547.0	0.150
1347	1	20	10	3786.5	1.000	4173.0	2412.1	1.00E+11	10	0.0000	450.0	0.150
1347	1595	10	20	3408.0	1.000	3430.2	1963.3	1.00E+11	10	0.0000	398.7	0.050
960	1595	20	10	241.8	1.000	259.9	146.0	1.00E+11	10	0.0000	299.5	0.050
960	960	5	5	0.0	1.000	0.0	0.0	1.00E+11	10	0.0000	299.5	0.050
										TOTAL CUF =	**0.1596**	

(a) A table published by a multi-billion-dollar agency (probably using Word)

j	A_j	B_j	$n(A_j)$	$n(B_j)$	MB'_j [ksi]	$k_{e,j}$	S'_j [ksi]	$S_{\mathrm{alt},j}$ [ksi]	N_j	n_j	U_j	$T_{\mathrm{max},j}$ [°F]
1	447	694	20	5	125.5	2.580	144.2	220.400	1.40×10^{2}	5	3.57×10^{-2}	566.6
2	447	699	15	50	121.6	2.405	139	198.300	1.79×10^{2}	15	8.38×10^{-2}	566.6
3	699	1020	35	20	104.7	1.653	126.5	124.900	5.77×10^{2}	20	3.47×10^{-2}	599.2
4	699	899	15	50	89.7	1.000	102.3	62.640	5.02×10^{3}	15	2.99×10^{-3}	336.1
5	695	899	5	35	84.99	1.000	98.8	59.750	5.77×10^{3}	5	8.67×10^{-4}	336.1
6	899	1432	30	20	66.67	1.000	83.1	50.360	9.56×10^{3}	20	2.09×10^{-3}	634.2
7	184	899	20	10	68.23	1.000	76.76	46.440	1.24×10^{4}	10	8.09×10^{-4}	600.0
8	184	1641	10	100	51.22	1.000	55.83	33.630	3.59×10^{4}	10	2.78×10^{-4}	634.2
9	1296	1641	20	90	32.69	1.000	38.94	22.110	1.53×10^{5}	20	1.31×10^{-4}	366.2
10	1134	1641	20	70	27.31	1.000	34.49	19.800	2.34×10^{5}	20	8.53×10^{-5}	419.2
11	1641	2215	50	100	25.47	1.000	25.89	15.270	1.07×10^{6}	50	4.66×10^{-5}	547.0
12	1213	2215	20	50	22.34	1.000	25.3	14.930	1.31×10^{6}	20	1.53×10^{-5}	547.0
13	1630	2215	100	30	24.88	1.000	25.2	14.870	1.35×10^{6}	30	2.22×10^{-5}	547.0
14	1347	1630	20	70	16.71	1.000	17.12	9.798	3.72×10^{9}	20	5.38×10^{-9}	398.7
15	960	1630	20	50	13.54	1.000	13.95	8.405	7.76×10^{10}	20	2.58×10^{-10}	634.2
16	1595	1630	20	30	13.3	1.000	13.69	7.690	1.00×10^{11}	20	2.00×10^{-10}	299.4
17	1	1630	20	10	12.92	1.000	12.95	7.469	1.00×10^{11}	10	1.00×10^{-10}	450.0
18	1	1596	10	100	12.92	1.000	12.95	7.469	1.00×10^{11}	10	1.00×10^{-10}	450.0
19	1562	1596	20	90	2.829	1.000	0.2345	0.132	1.00×10^{11}	20	2.00×10^{-10}	299.4
										CUF total =	**0.1615**	

(b) The same results in a report written by a a small third-world consulting company (FeenoX+AWK+LaTeX)

Figure 6: Results of the same environmentally-assisted fatigue problem.

- do not follow the principle of "each program should solve a single problem and interact with others" (even though most of them were born within the Unix world)
- focus too much on Word & Excel (and too little on Markdown & AWK)

Given what we have been discussing so far (and the details discussed in the SRS and the SDS), there is no single piece of software which provides all the benefits that FeenoX does.

Table 1: While traditional commercial codes rely on legacy codebases and proprietary lock-ins, FeenoX leverages the modern open-source ecosystem.

Feature	Traditional Commercial Code	FeenoX
Interface	Heavy GUI (User must click manually)	Text-based (Human or AI can write it)
Scalability	Expensive licenses per core; limited by license server	Unlimited MPI scaling; limited only by hardware
Transparency	"Binary files &""Black Box" " solvers"	Plain text inputs & Open Source Code
Workflow	Monolithic (One tool does everything moderately well)	"Modular (Unix philosophy: connects with Gmsh, Paraview, Python)"
Reproducibility	Difficult (What version? Which settings?)	Native (Git-trackable inputs & versioned binaries)
AI Readiness	Low (Visual interfaces are hard for LLMs)	High (Text interfaces are native to LLMs)

4.2 What comes next

The goal is to provide a comprehensive suite of tools to aid engineering teams to design and license nuclear reactors within a modern philosophy that leverages state-of-the-art computational technologies (web, cloud, IA, etc.). The following is a list of potential development tasks that can move FeenoX closer to the aforementioned goal:

- Improved physics
  - fuel burn-up and xenon quasi-static dynamics in core-level neutronics
  - transient core-level neutronics
  - radiation heat transfer in the thermal problem (not just boundary conditions)
  - more complex non-linear mechanics, including radiation-induced creep
- New problems
  - cell-level neutronics
  - P_N and/or SP_N neutronics formulation
  - one-dimensional two-phase thermal hydraulics
  - computational fluid dynamics
  - radiation transport and shielding
  - wrappers for existing libraries (e.g. MoFEM, Sparselizard, Ratel, etc.) so they can be used thorugh FeenoX's input files
- UI/UXs
  - specific web-based interfaces
  - REST API
  - Python API
  - FreeCAD workbench
- Numerical improvements
  - leverage GPUs/APUs for assembly and solve

- Automatic Mesh Refinement
- handling multi-point constrains with
  - Lagrange multipliers
  - direct elimination