ADVERSARIAL SELF-PLAY FOR SCIENTIFIC-COMPUTING AGENTS: AN ARENA WHERE A MODEL WRITES FINITE-ELEMENT PROBLEMS IT CANNOT SOLVE

Anonymous authors
Paper under double-blind review

ABSTRACT

Static benchmarks for tool-using scientific agents saturate quickly and are vulnerable to contamination, while human-authored problems are expensive and rarely calibrated to the frontier of what an agent can do. We introduce an adversarial, zero-sum *arena* in which a single language-model agent plays both sides of a game about numerical finite-element computation: a GENERATOR designs a multiple-choice question, together with a ground-truth value, three distractors, and four verification scripts that independently reproduce every option with FEniCSx on a fixed, tagged mesh; a SOLVER, the same model, must answer the question, first without tools and then with a sandboxed Python/FEniCSx environment. The arena executes the generator's scripts, forfeits any round whose question is not fully verified, presents each question under four cyclic option orderings, and requires the solver to be correct on all four. Every invocation runs under fail-closed filesystem, CPU, and memory isolation, and every call is metered for time, tokens, and cost. We report a match of 866 rounds played by a single frontier coding model. Of the 666 verified questions, the solver answered 40% and failed 60%, despite a per-attempt tool-solver accuracy of 71%. The generator's strategy visibly evolves: median question length grows from 2.3k to 17k characters and questions progress from single PDEs to chains of six to eight coupled sub-problems; the no-tool solver, meanwhile, exploits structure in the distractors to answer 13% of verified questions without computation. We analyze these dynamics, the 23% forfeit rate caused by the generator's own verification scripts, and the cost profile of the match ($1.4k, 5.9B tokens). The arena yields a self-refreshing, verifiably correct, and difficulty-calibrated question bank—180 questions from this match—and a companion evaluation mode replays that bank against agents that must implement the finite-element method themselves, with no FEM or meshing package available. On these questions, which the tool-using model answered on all four presentations, the same model working without the library is correct on only 32.2%, and reproduces the library's value to the stated rounding on only 16.1%.

1 INTRODUCTION

Large language models (LLMs) equipped with code execution are increasingly used as assistants for scientific computing: they write simulation scripts, set up boundary conditions, and post-process numerical results. Measuring how good such agents actually are is difficult. Static benchmarks such as SciCode (Tian et al., 2024) or the scientific subsets of MMLU and GPQA (Hendrycks et al., 2021; Rein et al., 2024) are authored once, are quickly absorbed into training corpora (Sainz et al., 2023; Balloccu et al., 2024), and are not calibrated to the individual agent under test: a question that is trivial for one model may be impossible for another. Human-authored, verifiably correct numerical problems are also costly to produce, because every option in a multiple-choice item must be backed by an actual computation.

We take the position that the agent itself is the cheapest and best-calibrated source of hard, verifiable questions, *provided* that the arena around it refuses anything that is not machine-checkable. We

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

build such an arena for finite-element computation. The game is zero-sum and played by one model in two roles. The GENERATOR receives the history of the match and a fixed mesh of the square $(-5, 5)^2$ produced by Gmsh (Geuzaine & Remacle, 2009), and must emit a question, a design strategy, one ground-truth value, three distractors, and four Python scripts that compute those four values with FEniCSx (Baratta et al., 2023). The arena re-executes the scripts inside a sandbox and forfeits the round unless every script runs, prints exactly one finite number, and that number matches the declared option. A verified question then goes to the SOLVER, the same model, in two gates: four isolated no-tool calls, and, if any of those fails, four isolated tool-enabled calls with Python, FEniCSx, and a read-only copy of the mesh. Each gate shows the same four values under the four cyclic orderings of A–D and passes only if all four calls are correct. The generator scores $+1$ when the solver fails at the tool gate and -1 otherwise, including on forfeits.

This design has three properties that we consider essential for self-generated scientific benchmarks. First, **verifiability**: every option is the output of a script that the arena, not the generator, ran. Second, **calibration**: because generator and solver are the same model, the generator is rewarded exactly for the questions that sit at the boundary of the model's own ability, and the reward is re-estimated every round. Third, **robustness of the measurement**: the all-four-orderings rule removes positional bias and lucky guesses, the separation into no-tool and tool gates separates recall from computation, and fail-closed sandboxing (Landlock, cgroup v2, per-invocation memory ceilings, CPU pinning) makes the numbers reproducible.

We ran a match of 866 rounds with a single frontier coding model over about two weeks of wall-clock time. Our contributions are:

- **The arena.** A fully specified, open-source zero-sum game for self-play generation of verified multiple-choice questions about finite-element computation, with two solver gates, fail-closed isolation, and complete time/token/cost metering (§3).
- **A long self-play match and its analysis.** We characterize the 866-round match: outcome distribution, the escalation of the generator's strategy from single PDEs to multi-physics chains, the behaviour of the no-tool gate, the failure modes of the tool solver (wrong answers, silent sessions, out-of-memory kills), and the generator's forfeits (§5).
- **A question bank and an evaluation mode.** The match leaves a bank of verified questions with known difficulty; a companion evaluation mode replays those questions against an agent that must implement the finite-element method by hand, with no FEM library available (§3.3). From this match the bank holds 180 questions. Evaluating the same model on it shows that tool use is doing most of the work: accuracy drops from 100% with FEniCSx to 32.2% without it, and to 24.5% on coupled multi-physics questions (§5.6).

2 RELATED WORK

Self-play and self-generated curricula. Self-play produced superhuman game players (Silver et al., 2017), and asymmetric self-play between a task-proposer and a task-solver was proposed as an automatic curriculum for reinforcement learning (Sukhbaatar et al., 2018; Dennis et al., 2020; Wang et al., 2019). For language models, recent work lets a model propose and solve its own problems with a code executor as the sole source of reward (Zhao et al., 2025; Huang et al., 2025; Cheng et al., 2024), and self-improvement methods such as SPIN (Chen et al., 2024) and self-instruct style data generation (Wang et al., 2023; Xu et al., 2024) use the model to author its own training data. Our work shares the proposer/solver structure but differs in target and in verification: the questions are about numerical scientific computation on a fixed discretization, and every option must be reproduced by a script that the arena itself executes in a sandbox. We also use the game for *measurement* rather than for training: the match log is the artefact.

Adversarial and dynamic benchmarks. Dynamic benchmarking (Kiela et al., 2021) and adversarial data collection (Nie et al., 2020) use humans in the loop to find examples a model fails on. Our arena replaces the human with the model itself, which makes the loop cheap enough to run for hundreds of rounds, at the price of having to defend against the generator's own mistakes (Section 5.5).

Agents for code and science. Tool-augmented LLMs (Schick et al., 2023; Yao et al., 2023) and coding-agent benchmarks (Jimenez et al., 2024; Chan et al., 2025) evaluate whether a model can

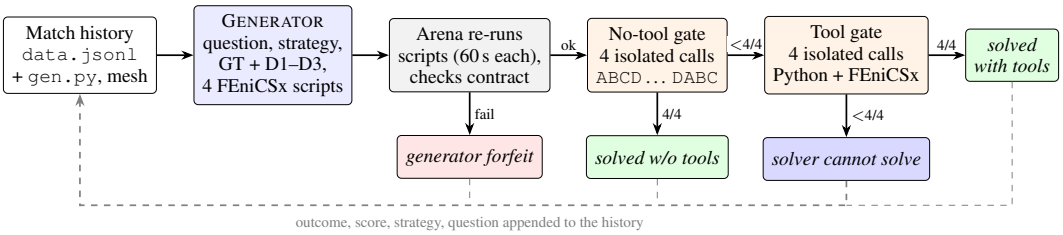


Figure 1: One round of the arena. Every agent call and script runs in its own fail-closed sandbox (Landlock, cgroup v2, 1536 MiB, two pinned cores). Only *solver cannot solve* (blue) scores $+1$ for the generator; the two solved outcomes (green) and a forfeit (red) score $+1$ for the solver.

write and run code to reach a goal. SciCode (Tian et al., 2024) targets research-level scientific programming with human-written tests. We focus specifically on finite-element modelling, where correctness hinges on boundary conditions, variational forms, quadrature, and post-processing choices that are easy to state and hard to get exactly right, and where a numerically verified ground truth is available by construction.

Multiple-choice artefacts. LLMs are sensitive to option ordering (Zheng et al., 2024) and can sometimes answer multiple-choice items from the options alone (Balepur et al., 2024). The arena's four-cyclic-ordering rule addresses the former; our no-tool gate directly measures the latter, and we find it to be a real effect even for numerically verified questions (Section 5.3).

3 THE ARENA

3.1 GAME

A match is a sequence of rounds between two roles played by the same model. Figure 1 summarizes one round.

Generator. The generator's prompt states the rules, the resource limits, the source of the mesh generator `gen.py` (a 10×10 square, target element size $h = 0.25$, physical tags for the domain and for the boundary), and a snapshot of recent outcomes. A compact history file with the round, outcome, running score, strategy, and question of every past round is in its workspace, and it is instructed to analyse this history programmatically and to balance two tactics: *deepening* (staying in a family that recently defeated the solver and making it more sophisticated) and *exploring* (pivoting to a new PDE type or numerical domain). It must produce six tagged blocks—strategy, question, ground truth, and three distractors—and four scripts `gt.py`, `d1.py`, `d2.py`, `d3.py`, each of which must run with the mesh as its only external input and print exactly one line `RESULT: <number>`. Distractors must arise from a *controlled mistake* (a wrong sign or boundary condition, an omitted term), and any rounding rule must be stated in the question.

Verification. The arena runs the four scripts itself, each with a 60 s limit, and marks the question *well defined* only if all six blocks are present, all four values are finite and pairwise distinct, all four scripts succeed, every printed result matches its declared block exactly, and the question text does not repeat an earlier round; question identity is the SHA-256 hash of the normalized question text, so rewording the options does not make a question new. Otherwise the generator forfeits, the reasons are stored, and the round still enters the database under a synthetic forfeit question, so every round is represented.

Solver gates. A well-defined question is shown to four isolated no-tool calls under the cyclic orderings `ABCD`, `BCDA`, `CDAB`, `DABC`. If all four answer correctly, the round ends as *solved without tools*. Otherwise the same four presentations go to four isolated tool-enabled calls that may run Python with FEniCSx 0.10 and Gmsh on a read-only copy of the mesh. The tool gate passes only if all four are correct (*solved with tools*); any wrong answer, missing answer, timeout, or out-of-memory kill fails it (*solver cannot solve*). The four calls of a gate run concurrently; once a failure is known, unfinished sibling calls are cancelled and stored as unavailable, so they count neither as a miss nor

as a success. A call that never emits a tagged `<<< X >>>` answer is scored as incorrect (answer parsing is detailed in Appendix A).

Scoring. Each round is zero-sum: the generator receives $+1$ only when the tool gate fails, and -1 when the solver succeeds at either gate or when the generator forfeits (Table 3 in Appendix A).

3.2 Isolation and metering

Every agent call and every generator script runs in its own playground under a fail-closed Landlock filesystem allowlist, a cgroup v2 scope killed atomically at cleanup, a 1536 MiB memory ceiling that is the same for every role, and two dedicated CPU cores; each agent call is limited to 1800 s. Every call is metered for time, tokens, and cost. Appendix A gives the details.

3.3 Evaluation mode

The match doubles as a question-bank generator, and the bank is only useful if we can say what its questions measure. The *collect* step keeps a question if, in some round, it passed two gates: a *difficulty* gate (at least one no-tool presentation returned a recorded wrong answer; unavailable, cancelled attempts count as neither miss nor success) and a *solvability* gate (all four tool presentations were correct). These are questions that the model could not answer from recall but could answer reliably with a finite-element library, so their ground truth has been independently reproduced five times: once by the generator's own script and four times by the tool solver. Collection reads stored rounds only and makes no model calls.

The *evaluate* step replays the bank against an agent that must *implement the finite-element method itself*: the tool-solver setting with the FEM stack removed but general-purpose numerics (NumPy, SciPy, SymPy, mpmath) left in place. Array operations and linear solves may come from a library; element definitions, quadrature, the weak form, assembly, and boundary conditions must be the agent's own code. No FEM or meshing package is installed or reachable, installers are stripped, and workspaces are scanned afterwards for downloaded packages, which *label* a record rather than void it. The read-only mesh comes with a standard-library reader that matches Gmsh exactly. Each question gets one call on a uniformly random arrangement of its options, under the solver's two cores and 1536 MiB but a 3600 s limit. Appendix A gives the full protocol.

4 Experimental setup

We report one match of 866 rounds played by a single frontier coding model (`composer-2.5`, accessed through its agent CLI) in all roles, with no topic hint, between 1 and 15 September 2026, across ten resumed sessions. The mesh is a fixed unstructured Gmsh triangulation of $(-5, 5)^2$ with target size $h = 0.25$: 1,940 nodes, 3,718 triangles, and 160 boundary segments, with physical tag 1 on the domain and a single tag 2 on the whole boundary (so any condition on an individual edge must be imposed geometrically, a detail the generator learned to exploit). Agents ran with the limits above; generator scripts had a 60 s limit. The environment for the generator and the solver used Python 3.11, FEniCSx 0.10.0, and Gmsh 4.15.2, on one workstation (Appendix B). The match cost \$1,421 and 5.9B tokens (356 h of cumulative agent time); Appendix B breaks this down.

The FEM-free evaluation (Section 5.6) uses the bank collected from this match with the same model, `composer-2.5`, as evaluator.

5 Results

5.1 Overall outcome

Table 1 gives the outcome distribution and Figure 2 the cumulative score and the rolling outcome mix. The solver finished ahead, $+64$ to -64, but this headline hides two opposite effects. Among the 666 verified questions, the generator *won* 401 (60%; 95% block-bootstrap CI 53–67%) and the solver only 265 (40%); the solver's lead comes entirely from the 200 generator forfeits (23% of rounds). Of the 265 solver wins, 85 were obtained without tools and 180 with tools.

Table 1: Outcomes of the 866-round match.

Outcome	Rounds	% of all	% of verified
Solver fails at tool gate (generator wins)	401	46.3	60.2
Solved with tools	180	20.8	27.0
Solved without tools	85	9.8	12.8
Generator forfeit	200	23.1	–
Total	866	100	666 verified

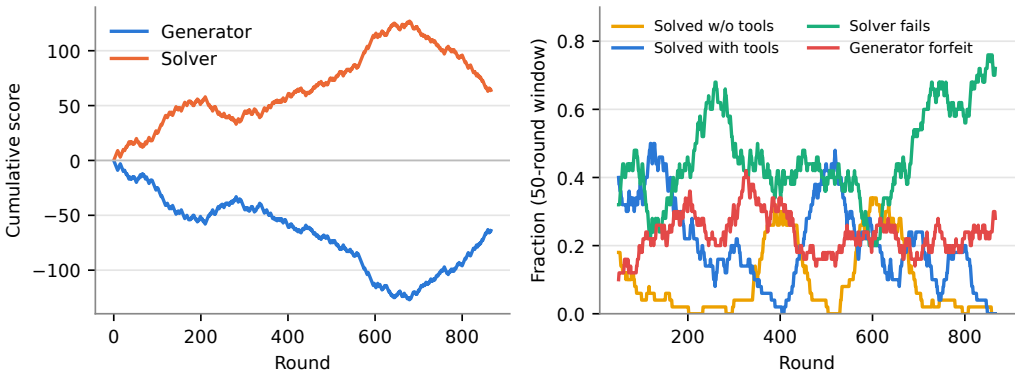


Figure 2: Left: cumulative zero-sum score of the two roles. Right: fraction of each outcome over a trailing 50-round window.

The all-four-presentations rule is a strong filter. Individual tool-solver attempts were correct 71% of the time (1,437 scored attempts), yet only 180 of the 581 rounds that reached the tool gate (31%) passed it. A per-attempt metric would have described this model as a competent finite-element user; the per-question metric shows that it is rarely *reliably* correct on questions written to its own frontier.

5.2 The generator's strategy escalates

The match is not stationary (Figure 2, right; per-block breakdown in Figure 6 and Table 4). In the first 200 rounds the generator mostly posed single steady problems—Poisson and anisotropic diffusion with mixed Dirichlet/Neumann/Robin data, plane-strain linear elasticity, Stokes—and asked for an integral or flux functional of the discrete solution; 70 of the 158 verified questions in that stretch were solved with tools. Around round 300 the generator switched to *coupled* problems: chains of two to three sub-problems (for example a Darcy solve feeding an SUPG-stabilized transport step feeding a Biot poroelasticity solve) whose composite functional depends on every stage. From round 600 the chains grew to six to eight coupled sub-problems, often with a time-stepped stage (BDF2 or Crank–Nicolson) and Simpson-weighted time integration of a functional. Figure 3 shows the effect: median question length rises from 2.3k characters in rounds 1–100 to 17k in rounds 701–800, and the median number of named sub-problems rises from 0 to 8. Tool-solver session time roughly doubles over the same period, and the solver's median time on questions it eventually got wrong (7.2 min) is nearly twice its time on questions it got right (3.9 min).

The escalation is driven by the reward. The generator's stated strategies reference the previous rounds explicitly ("Round 340 forfeited on the Darcy–SUPG–BDF2–Biot pivot, so this round retries that family with verified scripts and a rotated geometry"), and 606 of 866 strategies describe themselves as deepening against 227 as pivoting. The generator's win rate climbs from 44% of verified rounds in the first 100 to 98% in the last 66, where the solver won a single round; comparing rounds 1–300 (55%) with rounds 601–866 (76%), the increase is 21 points (95% block-bootstrap CI 4–34). The forfeit rate, by contrast, shows no significant trend across blocks ($\chi^2_8 = 13.0$, $p = 0.11$). The trade-off is that questions become long procedural specifications rather than conceptually hard problems; we return to this in Section 6.

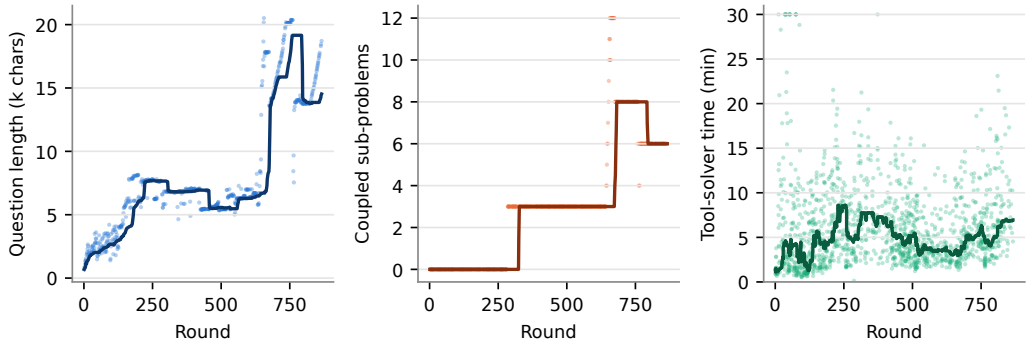


Figure 3: Escalation of the generator's questions over the match (verified rounds only; dark line is a trailing 50-round median). Left: question length. Middle: number of named coupled sub-problems in the question. Right: wall-clock time of individual tool-solver sessions.

Near-duplicate questions and shrinking option gaps. Two further trends accompany the escalation, and both are ways of exploiting the letter rather than the spirit of the contract. First, the duplicate check compares exact normalized question text, and the generator settles into long runs of *near*-duplicates: the median character-level similarity between consecutive verified questions (first 3,000 characters, difflib ratio) rises from 0.67 in rounds 1–100 to 0.98 in rounds 201–300 and 0.998 in rounds 801–866. The strategy of round 856 is typical: "Deepen the R833–R855 six-stage chain … by adding $\eta_{c^2T^2\psi^4\phi^{10}} \int_\Omega c_h^2 T_h^2 \psi_h^4 \phi_h^{10}\, \mathrm{d}x$ atop R855's ϕ^9 volume term", i.e. one more small weighted term appended to a functional that already carries several dozen. Second, the options crowd together: the median relative gap between the ground truth and its nearest distractor falls from 1.6×10^{-2} in rounds 1–100 to 5×10^{-4} in rounds 801–866, and in that final block 78% of verified questions have a distractor within 0.1% of the ground truth (e.g. 6709.4203 vs. 6709.4027 in round 856). Close options defeat elimination-by-magnitude in the no-tool gate and make the tool solver's answer sensitive to every small implementation choice, which is consistent with the inconsistent failures reported below. Both effects are legitimate under the current rules and suggest concrete amendments (Section 6).

5.3 The no-tool gate and distractor leakage

The no-tool gate is meant to filter out questions that can be answered by recall. It is cheap (32 s and about \$0.02 per presentation) and it is informative. Figure 5 (Appendix E) shows that on 406 of 666 verified questions the no-tool solver got none of the four presentations right, and that the distribution is strongly bimodal: a question is either answered on all four or on almost none, so the gate is behaving as a consistency test rather than as a four-fold lottery. Nevertheless, the no-tool solver answered 85 questions (13%) on all four presentations, with a noticeable spike in rounds 301–400 and 501–600 (Figure 6), precisely when the generator was introducing multi-stage chains.

Inspecting these transcripts shows the mechanism. The solver does not compute anything; it reasons about the *structure* of the distractors. Because the generator was instructed to construct each distractor by a controlled mistake such as omitting a term, the distractors are systematically biased relative to the ground truth (for example, three of four options are "missing one positive contribution" to an energy functional and the correct answer is the one that is not), and the solver identifies the correct value by elimination. This is a numerically verified version of the "answer from the options alone" artefact (Balepur et al., 2024). The generator learns to counter it—the no-tool solve rate collapses to 1 of 50 verified rounds after round 800—but the effect argues for a stronger distractor contract, for instance requiring distractors that bracket the ground truth or that arise from mistakes of both signs.

Individual no-tool presentations that produced an answer were correct 67% of the time (chance is 25%), but 24% of no-tool presentations produced no parseable answer at all, which we treat as incorrect; the no-answer rate rises to 52% in the final 66 rounds as questions become long.

5.4 How the tool solver fails

Of the 401 rounds the solver lost, 284 contained at least one presentation with a confidently wrong answer, and 117 were lost only through presentations that ended without any parseable answer: 16 hit the 1800 s session limit and the rest ended early with no output. Across all 1,437 scored tool-solver attempts, 100 (7%) were killed by the 1536 MiB memory ceiling, which the arena charges to the solver because the limit is stated in its prompt.

The wrong answers are spread almost evenly over the three distractors (91, 103, and 89 picks of D1, D2, D3), so no single kind of controlled mistake dominates. More telling is that the failures are largely *inconsistent* rather than systematic. In 167 of the 401 lost rounds at least one of the four presentations was answered correctly, and among the 150 lost rounds in which two or more presentations produced an answer, the answered values disagreed in 127 (lower bounds, since siblings are cancelled after the first failure). The solver does not consistently misread the problem; its long implementations land on the correct value only some of the time, which is what the all-four rule penalizes. Qualitatively, the wrong transcripts we inspected impose a condition on the whole tagged boundary rather than on the named edge, drop a stabilization or cross-coupling term, or evaluate a functional on the wrong field (not quantified).

5.5 Generator forfeits

The generator forfeited 200 rounds (23%), a rate that stayed between 15% and 32% per hundred rounds throughout the match. In 163 of them a verification script crashed when the arena re-ran it (Table 5). The strict contract is what keeps unverifiable questions out of the bank, but it spends nearly a quarter of the generator's budget on rounds that produce nothing.

5.6 Implementing the method by hand: FEM-free evaluation

Collection over the 866 rounds yields a bank of 180 questions, one per *solved with tools* round: 82 single-problem questions, mostly from the first half of the match, and 98 questions with two or more coupled sub-problems. Every one of them defeated the no-tool gate at least once (121 were missed on all four no-tool presentations) and was answered correctly on all four tool presentations, where the tool solver's median session took 3.4 min at a median $0.33 per presentation.

We replayed the whole bank against the same model acting as the FEM-free evaluator (Section 3.3); Table 2 and Figure 4 summarize the 180 sessions.

Without the library, accuracy collapses. The evaluator answered 58 of the 180 questions correctly (32.2%; 95% Wilson CI 25.8–39.4%), on questions where the tool solver had been correct on 4 of 4 presentations. It gave a wrong answer on 64, hit the 60-minute session limit on 31, and was killed by the 1536 MiB memory ceiling on 27, typically while building a large hand-assembled system. (One timed-out session had already written the correct letter; the arena scores any session that does not exit cleanly as incorrect.) Sessions were long: the median took 22.2 min, against 3.4 min for the tool solver on the same questions. The evaluation cost about $211 ($114 metered; the 58 timed-out or killed sessions report no usage and are priced by duration). Excluding the 34 flagged sessions changes little: 48 of 146 (32.9%).

Accuracy falls with question length and coupling. Accuracy is 41.5% on single-problem questions and 24.5% on coupled ones (Fisher exact $p = 0.02$), and it falls from 55.6% in the shortest quartile of questions to 15.6–17.8% in the two longest (Figure 4); correctly answered questions are markedly shorter (Mann–Whitney $p < 10^{-5}$). Questions from rounds 1–200, mostly single steady problems, are answered 47.1% of the time. The no-tool gate carries little extra signal: on the 121 questions that no no-tool presentation answered, accuracy is 30.6%, indistinguishable from the rest of the bank (Fisher $p = 0.50$). What makes a question hard without the library is therefore the amount of method to be reimplemented—mixed Taylor–Hood spaces, SUPG, BDF2 histories, Robin terms on named edges—rather than whether the answer can be guessed.

Correct letters are not always reproduced values. The multiple-choice format lets an agent that computed the wrong number still pick the right option. We therefore checked the evaluator's final

Table 2: FEM-free evaluation of the 180-question bank with `composer-2.5` (one call per question on a uniformly random arrangement of its options, 60-minute limit). The tool solver was, by construction of the bank, correct on all four presentations of every question. *Mem. kill*: the session was killed by the 1536 MiB memory ceiling before answering. *Before/after environment fix*: see Section 5.6.

Questions	n	Correct	Wrong	Timeout	Mem. kill	Med. time (min)
Single problem	82	34 (41%)	25	14	9	17.9
Coupled, 2–3 sub-problems	73	18 (25%)	27	15	13	26.9
Coupled, $\geq$4 sub-problems	25	6 (24%)	12	2	5	26.9
Length Q1 ($<$3.6k chars)	45	25 (56%)	9	9	2	11.8
Length Q2	45	18 (40%)	18	4	5	22.4
Length Q3	45	7 (16%)	19	10	9	31.3
Length Q4 ($\geq$6.6k chars)	45	8 (18%)	18	8	11	24.1
Before environment fix	123	42 (34%)	43	27	11	22.4
After environment fix	57	16 (28%)	21	4	16	20.5
Unflagged sessions only	146	48 (33%)	49	26	23	22.6
All	180	58 (32%)	64	31	27	22.2

message for its own computed value (a heuristic: numbers with at least three decimals that are not option texts). In 29 of the 58 correct answers the stated value reproduces the ground truth to the question's rounding rule; in 27 the evaluator reports a different value and chooses the ground truth as "the closest" option, and the same move accounts for 50 of the 64 wrong answers. By the strict criterion of reproducing FEniCSx's discrete answer, the evaluator succeeds on only 29 questions (16.1% of the bank; CI 11.5–22.2%). In 8 wrong answers, conversely, the evaluator's value reproduces a *distractor* to four decimals, i.e. it made exactly the controlled mistake that the generator had scripted, such as omitting a traction term or a history block. These counts rest on the heuristic above, which we have not validated against hand labels.

Successful transcripts show the intended behaviour, e.g. assembling P_1 stiffness from barycentric gradients and switching to a three-point rule after noticing that centroid lumping was "slightly low". On failed coupled questions a recurring pattern is a hand-written saddle-point solver for the Taylor–Hood block that converges slowly or not at all ("divergence of $\mathbf{u}$ stayed large"), after which the downstream stages inherit the error.

An environment defect, and the restriction flags. In the first 123 sessions (rounds up to 499), the agent's login shell re-sourced `/etc/profile`, which put an interpreter outside the Landlock allowlist first on `PATH`; `python` then failed, and `python3` resolved to a system interpreter without NumPy. We pinned `PATH` in the sandbox's shell profiles before the last 57 sessions. All 34 flags fall in the affected period: in 20 the agent, correctly finding the interpreter unusable, downloaded NumPy and SciPy; in 14 it compiled its own C linear solvers. None obtained a FEM or meshing package. Accuracy was 34.1% before the fix and 28.1% after, but the periods cover different, increasingly coupled parts of the bank, so this does not isolate the defect. Since it removed a permitted library, it can only have lowered accuracy, making our figures conservative.

6 Discussion and limitations

What kind of difficulty does self-play find? The generator discovered that the most reliable way to defeat a tool-using copy of itself is *specification length*: long chains of coupled sub-problems where every stage offers an opportunity for a plausible mis-implementation. This is a real weakness of current agents—it is exactly what a practitioner delegating a multi-physics workflow would care about—but it is not the only kind of difficulty, and the arena's reward does not distinguish a conceptually deep question from a long one. Rewards that penalize question length, or that pay for questions the solver fails *quickly* rather than after 25 minutes, would change the equilibrium.

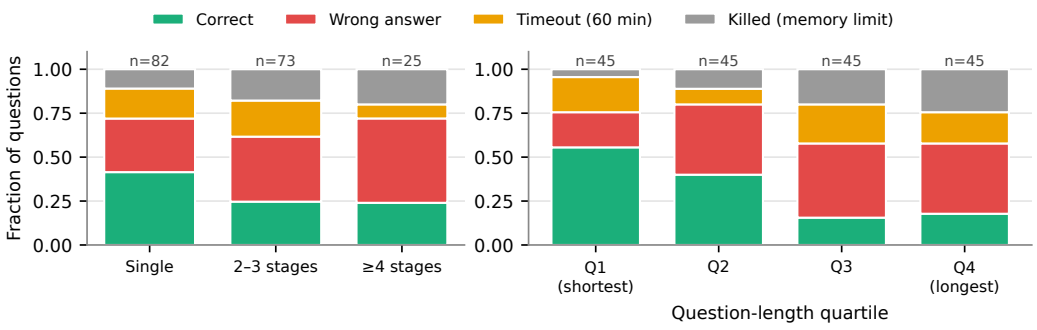


Figure 4: Outcome of FEM-free evaluator sessions by question type (left) and by quartile of question length (right).

Contract loopholes. The generator found two ways to win that the rules permit but that make the bank less useful: near-duplicate questions that differ from their predecessor by one term, and distractors placed within 0.1% of the ground truth (Section 5.2). Natural amendments are a similarity-based rather than exact-text duplicate check (for example on the question's embedding or on the set of sub-problems it names), and a minimum relative separation between options that the verification step can check mechanically.

What the FEM-free evaluation can and cannot certify. The evaluator's environment makes it impossible to import a FEM package from disk, but the agent keeps network access, so the post-hoc workspace scan is a detector rather than a guarantee; we report violations rather than silently dropping them. In our run the scan flagged 34 sessions, all during the environment defect and none with a FEM package; it also revealed that a C compiler was reachable, which the next version of the sandbox should either remove or explicitly permit. Because the answer is multiple choice, a correct letter does not imply a reproduced value (Section 5.6); a free-response variant that scores the reported number against the ground truth with a tolerance would remove this slack at the cost of the four-option calibration. The bank is also, by construction, the set of questions the tool solver answered on all four presentations, which excludes the hardest questions of the match: accuracy on the bank measures whether the method can be reproduced without the library on problems where the library is known to suffice, not the overall difficulty of the match.

Fixed mesh; verification cost. All questions live on one triangulation of a square, which keeps the ground truth reproducible and forces difficulty into the physics and numerics rather than geometry, but limits coverage; extending the arena to a small library of meshes is mechanical. The 23% forfeit rate is the price of the strict verification contract; letting the generator run and repair its scripts under the arena's exact limits before submission, or separating question design from script writing, are obvious mitigations we have not yet evaluated.

One match, one model. All results come from one 866-round match with one model, so they describe one trajectory of the game, not its distribution over seeds or models; using the same model in both roles also means the solver shares the generator's blind spots. Rounds are not independent (lag-1 autocorrelation of generator wins 0.24; near-duplicate runs), hence the block-bootstrap intervals for match-level rates, whereas the 180 evaluator sessions are independent and use binomial intervals and exact tests.

7 Conclusion

An adversarial self-play arena in which a model writes numerically verified finite-element questions for a tool-using copy of itself yields a difficulty-calibrated question bank at a few dollars per question and exposes reliability gaps that per-attempt accuracy hides. Without a finite-element library, the same model's accuracy on that bank falls from 100% to 32.2%, and to 16.1% when the library's value must be reproduced, with the steepest losses on long coupled problems.

Reproducibility statement

The arena, prompts, mesh generator, environment specifications, and the full JSONL log of the reported match (every question, script, attempt, and metered call) are provided as supplementary material. All numbers in Section 5 are computed from that log by a script included in the release. Everything is available at `https://anonymous.4open.science/r/fem-arena-7C3A`; the evaluation numbers are regenerated by `make_eval.py` directly from `evaluations.jsonl`.

Ethics statement

The work evaluates model capabilities on numerical mathematics and involves no human subjects or sensitive data. All agent invocations run under fail-closed filesystem, memory, and CPU isolation.

AI use statement

Language models are the object of study here: the generator, the solvers, and the evaluator are all played by the model named in the experimental setup, and every question, script, and answer they produced is part of the released log. Separately, we used an AI coding assistant to help build the arena's pipeline (orchestration, sandboxing, metering, and logging) and the analysis scripts that compute the reported statistics and figures, and to edit parts of the paper text for the final version. We did not use generative AI to formulate the research questions or to interpret results. We reviewed and tested all AI-assisted code, and every number in the paper is regenerated from the released logs by the included scripts. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

References

Nishant Balepur, Abhilasha Ravichander, and Rachel Rudinger. Artifacts or abduction: How do LLMs answer multiple-choice questions without the question? In *Annual Meeting of the Association for Computational Linguistics*, 2024.

Simone Balloccu, Patrícia Schmidtová, Mateusz Lango, and Ondřej Dušek. Leak, cheat, repeat: Data contamination and evaluation malpractices in closed-source LLMs. In *Conference of the European Chapter of the Association for Computational Linguistics*, 2024.

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. doi: 10.5281/zenodo.10447666.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Madry. MLE-bench: Evaluating machine learning agents on machine learning engineering. In *International Conference on Learning Representations*, 2025.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. Self-play fine-tuning converts weak language models to strong language models. In *International Conference on Machine Learning*, 2024.

Pengyu Cheng, Tianhao Hu, Han Xu, Zhisong Zhang, Yong Dai, Lei Han, Nan Du, and Xiaolong Li. Self-playing adversarial language game enhances LLM reasoning. In *Advances in Neural Information Processing Systems*, 2024.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. In *Advances in Neural Information Processing Systems*, 2020.

Christophe Geuzaine and Jean-François Remacle. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021.

Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Zhang, Jiaxin Huang, Haitao Mi, and Dong Yu. R-Zero: Self-evolving reasoning LLM from zero data. *arXiv preprint arXiv:2508.05004*, 2025.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024.

Douwe Kiela, Max Bartolo, Yixin Nie, Divyansh Kaushik, Atticus Geiger, Zhengxuan Wu, Bertie Vidgen, Grusha Prasad, Amanpreet Singh, Pratik Ringshia, et al. Dynabench: Rethinking benchmarking in NLP. In *Conference of the North American Chapter of the Association for Computational Linguistics*, 2021.

Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. Adversarial NLI: A new benchmark for natural language understanding. In *Annual Meeting of the Association for Computational Linguistics*, 2020.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level Google-proof Q&A benchmark. In *Conference on Language Modeling*, 2024.

Oscar Sainz, Jon Ander Campos, Iker García-Ferrero, Julen Etxaniz, Oier Lopez de Lacalle, and Eneko Agirre. NLP evaluation in trouble: On the need to measure LLM data contamination for each benchmark. In *Findings of the Association for Computational Linguistics: EMNLP*, 2023.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, 2023.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017.

Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. In *International Conference on Learning Representations*, 2018.

Minyang Tian, Luyu Gao, Shizhuo Dylan Zhang, Xinan Chen, Cunwei Fan, Xuefei Guo, Roland Haas, Pan Ji, Kittithat Krongchon, Yao Li, et al. SciCode: A research coding benchmark curated by scientists. In *Advances in Neural Information Processing Systems*, 2024.

Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O. Stanley. Paired open-ended trailblazer (POET): Endlessly generating increasingly complex and diverse learning environments and their solutions. In *Genetic and Evolutionary Computation Conference*, 2019.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Annual Meeting of the Association for Computational Linguistics*, 2023.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. WizardLM: Empowering large language models to follow complex instructions. In *International Conference on Learning Representations*, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data. *arXiv preprint arXiv:2505.03335*, 2025.

Chujie Zheng, Hao Zhou, Fandong Meng, Jie Zhou, and Minlie Huang. Large language models are not robust multiple choice selectors. In *International Conference on Learning Representations*, 2024.

A IMPLEMENTATION DETAILS

Table 3: Round outcomes and scores.

Round outcome	Generator	Solver
All four no-tool presentations correct (*solved without tools*)	-1	$+1$
No-tool gate fails; all four tool presentations correct (*solved with tools*)	-1	$+1$
Tool gate fails, incl. timeout or OOM kill (*solver cannot solve*)	$+1$	-1
Verification contract fails or generator OOM (*generator forfeit*)	-1	$+1$

Isolation. Every agent call and every generator script runs in its own playground under a fail-closed Landlock filesystem allowlist inherited by all descendants, a cgroup v2 scope that is killed atomically at cleanup, a 1536 MiB memory ceiling with swap disabled, and dedicated CPU affinity (two cores per gate slot, two for the generator). No-tool playgrounds are empty and read-only; solver playgrounds contain only the mesh. The arena verifies and clears playground contents between rounds. Each agent call is limited to 1800 s. Credentials, caches, and temporary files live in private per-invocation directories, and the memory ceiling is the same for every role, so the generator can only pose a question that it can verify inside the budget the solver is later given to answer it. An OOM kill fails only the affected invocation: a no-tool OOM advances the round to the tool gate, a tool-solver OOM fails the tool gate, and a generator OOM is a forfeit. An optional monitor samples cgroups, CPU affinity, memory, process containment, and playground integrity while a match runs and reports violations without controlling the match.

Metering. Every call is metered: wall-clock time, the CLI's own API time, input, output, and cache token counts, and a dollar cost derived from published per-token rates, keyed by model family so that reasoning-effort variants of the same model share one price. Totals are aggregated per round and per match, and an unpriced call is recorded as unknown rather than as zero. The match log therefore supports cost accounting at the granularity of a single presentation. Accounting is best-effort by design: a call that is cancelled or killed by the timeout never emits its usage envelope, so its tokens are recorded as absent while its answer is still parsed from the raw output, and all reported costs are therefore lower bounds.

Answer parsing. Answers are read from the agent's structured output stream. Because the final text concatenates every assistant message including narration, the arena takes the last assistant message that carries a tagged `<<< X >>>` answer, which skips pre-tool narration and prefers a corrected answer over an earlier draft. For the evaluator, a session that does not exit cleanly is scored as incorrect even if a tag was already emitted.

FEM-free evaluation protocol. The *evaluate* step replays the bank against an agent that must *implement the finite-element method itself*. It is the tool-solver phase with the FEM stack removed but general-purpose numerics left in place: the agent runs in a dedicated Conda environment with Python 3.11, NumPy, SciPy (including `scipy.sparse` and its direct, Krylov, ARPACK and LAPACK solvers), SymPy, mpmath, and Matplotlib, and with no finite-element or mesh-generation package (FEniCSx/DOLFINx, UFL, Basix, FFCx, PETSc, SLEPc, `mpi4py`, scikit-fem, SfePy, NGSolve, Firedrake, Gmsh, pygmsh, meshio, PyVista). The line is drawn between the arithmetic and the method: array operations and linear solves may come from a library, whereas element definitions, quadrature, the weak form, assembly, and boundary conditions must be the agent's own code (Appendix C). Meshers are excluded along with the FEM stack because the question is posed on one specific discretization; a mesher could only produce a *different* mesh and hence a different answer. Instead each workspace receives the read-only mesh and a standard-library reader whose node coordinates and triangle connectivity match Gmsh's exactly on that file.

The restriction is enforced rather than requested. The arena refuses to start unless the evaluator environment contains no forbidden module; the installer tooling (pip, setuptools, wheel, `ensurepip`) is stripped from the environment after it is built; and the Landlock allowlist admits only that one Conda prefix, so the FEniCSx environment and the package cache can be neither imported nor executed. Network access cannot be removed, because the agent CLI needs it, so each workspace is

scanned afterwards for wheels, compiled extensions, and package metadata. A finding *labels* the record (`violated`, with the offending paths) rather than voiding it, so that exclusion is an analysis decision rather than one baked into the data.

Each question receives exactly one evaluator call, on an arrangement of its options drawn uniformly from all 24 permutations. The match gates use the four cyclic rotations because all four run together and the ground truth then sits at each letter exactly once; for a single call, a rotation would be a fixed, guessable pattern, so the evaluator's verdict is a plain *correct/incorrect* and the quantity of interest is accuracy over the bank. The arrangement is stored with the record, so every question can be replayed exactly. Four questions run side by side, each in its own workspace on one gate slot's two cores and under the same 1536 MiB ceiling as the solver, so a question's result never depends on another's and remains comparable to the tool solver's; the session limit is 3600 s, twice the solver's, because the method has to be written by hand. There is no per-script time limit inside the session. Progress is tracked per evaluator model, so several models can be evaluated side by side on the same bank.

B Cost and forfeit details

All runs used one workstation (Intel Core i9-13900, 32 hardware threads, 7.5 GiB of RAM available to the container). In total the arena made $3{,}380$ metered calls consuming 5.9B tokens (143M input, 84M output, 5.7B cache reads) at a cost of $\$1{,}421$, i.e. $\$1.64$ per round; 37% of the cost is the generator and 62% the tool solver, while the no-tool gate costs 1.5%. Cumulative agent time was 356 h, run mostly in parallel across the four gate slots; the match's own wall-clock time was 15.6 h of active play over the ten sessions. Only $3{,}380$ of the $5{,}854$ recorded invocations reported token usage, since cancelled siblings and timed-out calls do not, so these figures are lower bounds.

Cache reads are 96% of all tokens: the generator re-reads the growing history and the solver re-reads long questions and its own scripts. The tool solver dominates the cost ($\$878$, 3.7B tokens), followed by the generator ($\$522$, 2.2B tokens); the no-tool gate is negligible ($\$22$). The median generator session took 4.3 min and the median tool-solver session 4.7 min. A verified question that defeated the solver therefore cost about $\$3.5$ of arena spend on average, which compares favourably with the cost of authoring and numerically verifying an equivalent item by hand (total spend divided by the 401 solver-defeating questions; the 180 questions of the evaluation bank cost $\$7.9$ each on the same basis).

Forfeits. In 163 cases at least one of the four scripts exited with an error when the arena re-ran it; in 15 a script printed a non-finite result; in 14 a script was killed by the memory ceiling; in 8 the printed value did not exactly match the declared option, usually a formatting or rounding discrepancy. The forfeit is the arena's main defence against unverifiable questions, and it is strict by design—a question with a wrong or unreproducible ground truth would poison the bank—but it also means that nearly a quarter of the generator's budget ($\$522$ in total) produced no question. Most script failures come from the generator writing four increasingly long scripts for an eight-stage coupled problem within a single session; Section 6 discusses remedies. The four scripts of every round are archived with the match log, so each forfeit can be replayed.

C Agent prompts

The four prompt templates are reproduced verbatim. Placeholders in braces (session and code time limits, core and memory limits, script names, the contents of `gen.py`, the past-rounds snapshot, the question and options) are filled in by the arena at run time.

C.1 Generator

```
You are the GENERATOR in an adversarial, zero-sum multiple-choice game.

Your opponent is the SOLVER, an LLM agent (the same model as you). The SOLVER
can use FEniCSx 0.10.0 for finite element computations and gmsh 4.15.2 to work
with the provided mesh file, and is able to execute software to answer your
question.
```

```
Your goal is to iteratively develop a strong question-design strategy that
produces fair, well-posed multiple-choice questions whose correct answers are
genuinely hard to obtain even with FEniCSx-based computation. The orchestrator
will fully shuffle the options before the SOLVER sees the choices.

Scoring (zero-sum):
  - Solver solves with or without tools -> Solver +1, Generator -1
  - Solver cannot solve -> Solver -1, Generator +1
  - You forfeit if your response or required scripts are missing/malformed; any
    script fails or times out; a script result disagrees with its corresponding
    output block; your question repeats an earlier question; or any hard
    requirement is violated -> Solver +1, Generator -1.

Strict time limits:
  - Session response limit: {SESSION_TIMEOUT_S} seconds.
  - Code execution limit: {CODE_TIMEOUT_S} seconds per script.

Resource limits:
  - Your entire process tree is restricted to {CPU_CORE_LIMIT} CPU cores and
    {MEMORY_LIMIT_MIB} MiB of memory, including descendants and page cache.
  - Creating additional processes or threads does not grant additional cores;
    they share the same {CPU_CORE_LIMIT}-core allocation.
  - Exceeding the memory limit causes an OOM kill and an immediate Generator
    forfeit. Design and verify the question within both resource limits.

Hard requirements:
 1. No Internet usage.
 2. Only work in your workspace.
 3. Terminal use is limited to running Python scripts for two purposes:
    (a) selectively analyzing `data.jsonl`; and (b) constructing and verifying
    the current question by evaluating its ground-truth and distractor values.
    Run every such script with this command form:
    `conda run -s -n fenicsx-env python <script>.py`.
 4. Use the `gen.py` contents supplied in this prompt to understand the mesh,
    and use the read-only `rectangle.msh` in your workspace as the mesh input.
 5. Create these four Python scripts in your workspace: `{GT_SCRIPT}`,
    `{D1_SCRIPT}`, `{D2_SCRIPT}`, and `{D3_SCRIPT}`. Each script MUST be
    able to run with `rectangle.msh` as its only required external file,
    excluding Python's standard library and installed Python packages.
 6. `{GT_SCRIPT}` MUST correctly evaluate the question's numerical answer.
    Each distractor script MUST evaluate one plausible wrong value by making a
    controlled mistake such as using a wrong boundary condition/sign,
    omitting a required term/step, or applying another realistic error.
 7. Every script MUST run in the provided Conda environment (including gmsh
    4.15.2 and FEniCSx 0.10.0), use `rectangle.msh`, and print exactly one line
    of the form `RESULT: <finite-real-number>`.
 8. Emit <<<GROUNDTRUTH>>>, <<<DISTRACTOR1>>>, <<<DISTRACTOR2>>>, and
    <<<DISTRACTOR3>>> blocks. After trimming whitespace, each block's text MUST
    exactly match the `RESULT:` value printed by its corresponding script.
 9. The four results MUST be numerically distinct, and their texts MUST also
    be distinct after trimming. The ground-truth value must be the unique
    correct answer, and all three distractor values must be plausible.
10. Solving the proposed question should require writing
    and executing proper Python code; relying only on mental work should be
    insufficient or risky.
11. If options involve numerical values that depend on rounding, you MUST
    state the exact rounding rule explicitly in the question text
    (e.g., decimal places, significant figures, tie-breaking convention).
12. The question text must be clear, well-posed, and humanly understandable:
    do not include meta commentary, trap explanations, or strategy hints;
    provide only the problem statement and required numeric/spec details; use
    standard mathematical/scientific conventions by default, define only
    nonstandard or potentially ambiguous notation/assumptions, and avoid
    ambiguous wording or underspecified abstractions.
13. Emit a concise strategy in <<<STRATEGY>>> (1-3 short bullets or
    sentences) describing how this round's question is designed to challenge
    a tool-using SOLVER while staying fair and well-posed.
14. Before designing this round, use the snapshot below and `data.jsonl` to
    learn from past rounds. The snapshot lists recent round outcomes. Each
    line of `data.jsonl` contains one completed round's `round`, `outcome`,
    `generator_total_score`, `strategy`, and `question`. Outcomes are one of:
    `solved without tools`, `solved with tools`, `solver cannot solve`, or
    `generator forfeit`. Analyze the history programmatically, and bring a
    compact internal summary into your context. Summarize relevant patterns,
    successful and unsuccessful strategies, and the SOLVER's apparent
    weaknesses. Across the match, strategically balance two distinct
    approaches based on the SOLVER's performance: (1) Deepening--staying
    within a problem family that previously challenged the SOLVER and
    making the problem itself more sophisticated, for instance through
    different boundary or interface conditions, a richer physical setting,
    or more intricate governing equations, and (2) Exploring--pivoting to
```

```
    new problem families, PDE types, or numerical domains to probe how the
    SOLVER behaves on unfamiliar terrain. You have the autonomy to decide
    when to deepen and when to pivot based on the history, but you must
    employ both tactics across the match and avoid getting trapped in a
    single problem family. Keep every problem fair and well-posed.
 15. Do not repeat the exact question text of an earlier round in this match;
    a repeated question causes an automatic forfeit. However, do not
    permanently abandon older topics. You are strongly encouraged to revisit
    previous problem families later in the match, applying the new insights,
    patterns, and history you have gathered from intervening rounds to attack
    the old topic from a fresh, more sophisticated angle.

The full contents of the mesh generation file `gen.py` are provided below.
Use them as the authoritative description of how `rectangle.msh` was created;
`gen.py` itself is not present in your workspace.

--- BEGIN gen.py ---
{GEN_PY}
--- END gen.py ---

Past rounds snapshot:
{PAST_ROUNDS_SNAPSHOT}

Output format -- emit EXACTLY these tagged blocks, in this order, and NOTHING
else (no preamble, no explanation, no markdown fences):

<<<STRATEGY>>>
... concise strategy for this round ...
<<</STRATEGY>>>

<<<QUESTION>>>
... the question text (Use LaTeX for display math). ...
<<</QUESTION>>>

<<<GROUNDTRUTH>>>
... exact numeric text printed by {GT_SCRIPT} ...
<<</GROUNDTRUTH>>>

<<<DISTRACTOR1>>>
... exact numeric text printed by {D1_SCRIPT} ...
<<</DISTRACTOR1>>>

<<<DISTRACTOR2>>>
... exact numeric text printed by {D2_SCRIPT} ...
<<</DISTRACTOR2>>>

<<<DISTRACTOR3>>>
... exact numeric text printed by {D3_SCRIPT} ...
<<</DISTRACTOR3>>>
```

C.2 NO-TOOL SOLVER

```
question_text = """
Answer the following question.

{QUESTION}

A. {OPTION_A}
B. {OPTION_B}
C. {OPTION_C}
D. {OPTION_D}

Your response should be in the form <<< X >>>, where X is either A, B, C, or D.

Strict time limits:
  - Session response limit: {SESSION_TIMEOUT_S} seconds.

Hard requirements:
  1. No Internet usage.
  2. Answer from your own reasoning alone. You must not use any tools:
     do not read or write files, run code, or execute shell/terminal
     commands. Rely solely on the information given in this prompt.

The full contents of the mesh generation file `gen.py` are provided below.
Treat this as your only source of information about the mesh.

--- BEGIN gen.py ---
{GEN_PY}
```

```
--- END gen.py ---
"""
```

C.3 TOOL SOLVER

```
question_text = """
Use FEniCSx 0.10.0 to answer the following question.

{QUESTION}

A. {OPTION_A}
B. {OPTION_B}
C. {OPTION_C}
D. {OPTION_D}

Your response should be in the form <<< X >>>, where X is either A, B, C, or D.

Strict time limits:
  - Session response limit: {SESSION_TIMEOUT_S} seconds.

Resource limits:
  - Your entire process tree is restricted to {CPU_CORE_LIMIT} CPU cores and
    {MEMORY_LIMIT_MIB} MiB of memory, including descendants and page cache.
  - Creating additional processes or threads does not grant additional cores;
    they share the same {CPU_CORE_LIMIT}-core allocation.
  - Exceeding the memory limit causes an OOM kill and counts as failing to solve
    the question.

Hard requirements:
  1. No Internet usage.
  2. Only work in your workspace.
  3. The only interaction with terminal is to run Python code in this command
     form: `conda run -s -n fenicsx-env python <script>.py`.
  4. Use the `gen.py` contents supplied below to understand the mesh, and use
     the read-only `rectangle.msh` in your workspace as the mesh input.

The full contents of the mesh generation file `gen.py` are provided below.
Use them as the authoritative description of how `rectangle.msh` was created;
`gen.py` itself is not present in your workspace.

--- BEGIN gen.py ---
{GEN_PY}
--- END gen.py ---
"""
```

C.4 FEM-FREE EVALUATOR

```
question_text = """
Answer the following question.

{QUESTION}

A. {OPTION_A}
B. {OPTION_B}
C. {OPTION_C}
D. {OPTION_D}

Your response should be in the form <<< X >>>, where X is either A, B, C, or D.

Strict time limits:
  - Session response limit: {SESSION_TIMEOUT_S} seconds.

Resource limits:
  - Your entire process tree is restricted to {CPU_CORE_LIMIT} CPU cores and
    {MEMORY_LIMIT_MIB} MiB of memory, including descendants and page cache.
  - Creating additional processes or threads does not grant additional cores;
    they share the same {CPU_CORE_LIMIT}-core allocation.
  - Exceeding the memory limit causes an OOM kill and counts as failing to
    answer the question.

Hard requirements:
  1. No Internet usage.
  2. Only work in your workspace.
  3. The only interaction with terminal is to run Python code in this command
     form: `python <script>.py`.
  4. You must implement the finite element method yourself. General-purpose
```

```
    scientific libraries are available and permitted, and you should use them
    freely: the Python standard library, NumPy, SciPy (including
    `scipy.sparse` and `scipy.sparse.linalg`), SymPy, mpmath, and Matplotlib.
  5. No finite element or mesh-generation package is available or permitted,
    including but not limited to FEniCSx, dolfinx, UFL, Basix, FFCx, PETSc,
    SLEPc, mpi4py, gmsh, pygmsh, meshio, PyVista, scikit-fem, SfePy, and
    NGSolve. The rule is about the method, not the arithmetic: the element
    definitions, quadrature, weak form, assembly, and boundary conditions
    must be your own code, while array operations and linear solves may come
    from a library.
  6. Do not install, download, vendor, or otherwise obtain any additional
    third-party package by any means. Attempting to do so voids your answer.
  7. Use the read-only `rectangle.msh` in your workspace as the mesh input.
    Do not generate a different mesh: the question is posed on this exact
    discretization, so a mesh you build yourself will give a different answer.

Reading the mesh:
  Since gmsh and meshio are unavailable, `read_msh.py` is provided read-only in
  your workspace. It parses the Gmsh MSH 4.1 ASCII container using only the
  standard library and does no finite-element work. Its output matches what
  gmsh itself reports for this file, and its plain lists convert directly with
  `numpy.array`.

    from read_msh import read_msh

    mesh = read_msh("rectangle.msh")
    mesh["points"] # [(x, y, z), ...] indexed 0..n-1, ascending node tag
    mesh["triangles"] # [(i, j, k), ...] indices into points
    mesh["triangle_tags"] # physical tag per triangle
    mesh["lines"] # [(i, j), ...] boundary segments
    mesh["line_tags"] # physical tag per line
    mesh["physical_names"] # {{(dim, tag): name}}

  You may read it, ignore it, or parse `rectangle.msh` yourself, but you must
  not modify it.

The full contents of the mesh generation file `gen.py` are provided below.
Use them as the authoritative description of how `rectangle.msh` was created;
`gen.py` itself is not present in your workspace, and you cannot run it because
it requires gmsh.

--- BEGIN gen.py ---
{GEN_PY}
--- END gen.py ---
"""
```

D EXAMPLE QUESTIONS

Early round (round 6, solved with tools). *Generator strategy:* "Rounds 4–5 were solved without tools on center-point evaluations, so this round deepens the mixed-boundary Poisson family from Round 3 by adding a symmetric anisotropic diffusion tensor and changing the target quantity to the domain integral. Distractors come from omitting off-diagonal anisotropy, enforcing Dirichlet data on the entire tagged boundary, and swapping the Dirichlet/Neumann sides."

> Consider the steady anisotropic diffusion problem on the square domain $\Omega = (-5,5) \times (-5,5)$: $-\nabla \cdot (K\nabla u) = f$ in Ω, with constant symmetric positive-definite tensor $K = \begin{pmatrix} 2.0 & 0.8 \\ 0.8 & 1.5 \end{pmatrix}$ and source term $f(x,y) = 1 + \frac{1}{2}\sin(\pi x/5)\cos(\pi y/5)$. The boundary conditions are $u = 0$ on $\{x = -5\} \cup \{x = 5\}$ and $(K\nabla u)\cdot \mathbf{n} = 0$ on $\{y = -5\} \cup \{y = 5\}$. Equivalently, find $u_h \in V$ such that $\int_\Omega K\nabla u_h \cdot \nabla v \, \mathrm{d}x = \int_\Omega f v \, \mathrm{d}x$ for all $v \in V$, with V the continuous piecewise linear functions satisfying $u = 0$ on $\{x = \pm 5\}$. Using the provided mesh file `rectangle.msh` (2D domain physical tag 1, 1D boundary physical tag 2) and P_1 Lagrange finite elements, compute u_h. What is $\int_\Omega u_h \, \mathrm{d}x$? Report the value rounded to **4 decimal places** using standard rounding (round half away from zero).
>
> Options: **436.1360** (ground truth), 416.4876, 205.4933, 590.7088.

The no-tool gate failed; all four tool presentations and, later, the FEM-free evaluator answered correctly.

Late round (round 856, solver fails). A 17k-character "six-stage coupled multiphysics computation" on the same mesh, which opens with the instruction to "impose boundary conditions only on geometric edges named by coordinate constraints (e.g. $\{x = -5\}$), not on every facet with tag 2". Its stages are: (A) steady Stokes–Oseen flow with Taylor–Hood elements, a Robin slip term on two edges and a pinned pressure gauge; (B) a steady anisotropic chemical potential coupled to the flow; (C) isotropic linear elasticity with a body force from stage B; (D) transient anisotropic heat with BDF2 time marching coupled to B and C; (E) BDF2–SUPG transport with skew-symmetric advection; and (F) a variable-coefficient Helmholtz reaction–diffusion problem. The target J_h combines Simpson-weighted time integrals over the heat and transport histories with about forty weighted volume integrals of products of the computed fields, reported to four decimals with round-half-to-even. Options: 6709.4203 (ground truth), 6708.8688, 6709.4027, 6661.9380; the distractors omit the heat-history Simpson block, the transport-history Simpson block, or one coupling term. The tool-solver presentation `DABC` answered 6709.4027, the distractor that omits the transport-history block and lies 0.0003% from the ground truth; its three siblings were cancelled as soon as that failure was known.

E Additional statistics

Table 4 breaks the match down by blocks of 100 rounds, and Figure 5 shows how many of the four no-tool presentations were answered correctly per verified question.

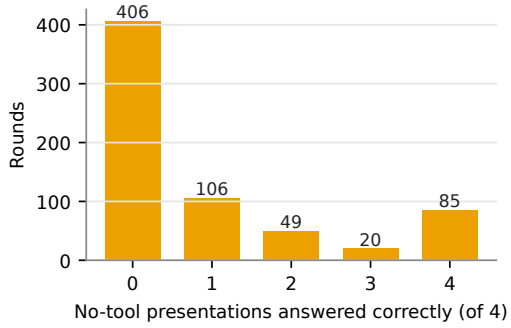


Figure 5: No-tool gate: number of the four presentations answered correctly, over the 666 verified questions.

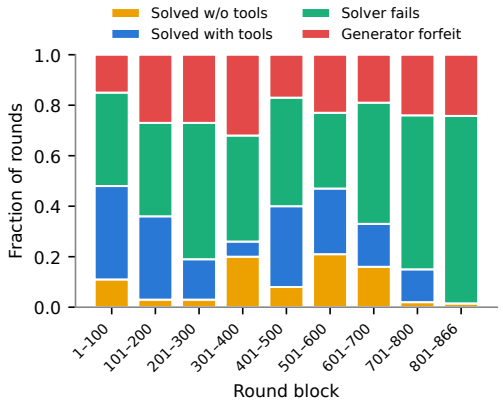


Figure 6: Outcome mix per block of 100 rounds.

Table 4: Outcomes, generator net score, median question length of verified questions, and spend per block of 100 rounds.

Rounds	w/o tools	with tools	solver fails	forfeit	gen. net	med. length (k)	cost ($)
1–100	11	37	37	15	−26	2.3	128
101–200	3	33	37	27	−26	5.5	176
201–300	3	16	54	27	+8	7.6	188
301–400	20	6	42	32	−16	6.8	162
401–500	8	32	43	17	−14	5.6	169
501–600	21	26	30	23	−40	6.3	148
601–700	16	17	48	19	−4	13.9	159
701–800	2	13	61	24	+22	17.0	187
801–866	1	0	49	16	+32	14.5	105
All	85	180	401	200	−64	6.8	1421

Table 5: Reasons for the 200 generator forfeits (first failing check per round).

Reason	Rounds
A verification script exited with an error	163
A script printed a non-finite result	15
A script was killed by the memory ceiling	14
Printed result did not match the declared option	8