Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Beyond Schema Conformance with Executable Verification for Scientific Configuration Repair

wasif masood

wasif.masood@empirischtech.at

Empirisch Tech GmbH https://orcid.org/0000-0003-3231-8104

shoaib noor

Empirisch Tech GmbH

Research Article

Keywords: Scientific configuration repair, Executable verification, Numerical reference conformance, Large language models, Execution-based evaluation, Verifiable coding, Scientific software, Schema validation

Posted Date: November 17th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11746019/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

Beyond Schema Conformance with Executable Verification for Scientific Configuration Repair

Wasif Masood
Empirisch Tech GmbH
wasif.masood@empirischtech.at

Shoaib Noor
Empirisch Tech GmbH
shoaib.noor@empirischtech.at

Abstract

Large language models can generate structurally valid scientific configuration files, but whether those configurations reproduce the intended computation is rarely tested. We study this gap on Palace, a finite-element solver for computational electromagnetics, using an executable verification hierarchy that measures schema conformance, operational acceptance by the solver, and numerical reference conformance. We construct a source-grounded fine-tuning corpus and a configuration-repair benchmark containing both shipped cases represented in the training corpus and newly authored cases created after the model and retrieval index were frozen. Fine-tuning provides the largest improvement, while the RAG-proxy setting and schema-guided repair provide additional gains. However, these improvements transfer much more strongly at the structural level than at the numerical level. On newly authored cases, the full pipeline reaches 89% schema conformance and 81% solver acceptance, but only 41% reference conformance. We further show that some failures arise from insufficient task specification and that blind LLM judges do not consistently reproduce execution-based labels. These results suggest that scientific configuration generation and repair should be evaluated against executable numerical references rather than schema conformance or model judgment alone.

1 Introduction

Large language models are increasingly used to generate structured configuration artifacts for continuous integration, infrastructure provisioning, network management, and scientific workflows. Existing methods improve reliability through constrained decoding, schema validation, retrieval, and iterative repair [Willard and Louf, 2023, Geng et al., 2023, Lewis et al., 2020, Ghaleb, 2026, Zhang et al., 2026]. These methods can substantially improve parsing, schema conformance, and tool acceptance. However, studies of continuous integration and infrastructure configurations show that a structurally valid artifact may still fail to reproduce the expected workflow, deploy successfully, or satisfy the intended requirements [Ghaleb, 2026, Zhang et al., 2026]. Structural conformance is therefore necessary but does not establish that a generated configuration produces the intended behavior.

Stronger evaluations use executable or formal checks. Software repair benchmarks evaluate generated changes through tests [Jimenez et al., 2024, Mündler et al., 2024], while Cornetto evaluates network configuration repair against formally verified behavioral specifications [Protogeros et al., 2026]. In scientific computing, CFDLLMBench [Somasekharan et al., 2026] evaluates generated CFD workflows through executability and numerical accuracy, while ChatCFD [Fan et al., 2026] reports a substantial gap between successful execution and physical fidelity. These results show that running a scientific workflow successfully does not by itself establish that the intended computation has been reproduced.

NeurIPS 2026 Workshop on AI for Verifiable Coding.

A second unresolved issue is case exposure. Evaluation on cases represented in a model's training or adaptation data may overstate its ability to transfer to new cases. Recent work on decontaminated software engineering benchmarks reports substantial differences between established benchmarks and fresh tasks collected after model training [Badertdinov et al., 2025]. For scientific configuration repair, it remains unclear whether improvements from domain fine-tuning, retrieval, and validator feedback persist when the underlying physical cases are newly authored rather than represented in the fine-tuning corpus. It is also unclear how repair performance should be interpreted when the prompt omits information needed to identify a unique physical target. We formalize this distinction through a specification sufficiency criterion that asks whether the information visible to the model determines the benchmark behavior up to the comparator tolerance.

We study these questions using Palace [AWS Center for Quantum Computing, 2025], an open-source finite-element solver for computational electromagnetics. Palace provides a schema, a command-line validator, executable examples, and stored regression outputs. We use these artifacts to define three increasingly strong outcomes comprising schema conformance, operational acceptance, and numerical reference conformance. We also construct two evaluation sets, one derived from shipped cases represented in the fine-tuning corpus and one from newly authored cases created after the model and retrieval index were frozen. This design measures both the gap between structural and numerical correctness and the extent to which repair performance transfers from case-exposed configurations to newly authored cases. We release both evaluation sets, with their meshes and reference outputs, and the QLoRA adapter under the Apache-2.0 license.[1]

2 Executable verification contract

Let c be a candidate repair, S the fixed Palace binary, $O(S(c))$ its selected outputs, and y^* the trusted regression reference. We write $y \simeq_\epsilon y^*$ when the strict comparator accepts y under the stated structural requirements and numerical tolerances.

2.1 Cumulative verification levels

We record three raw indicators:

$$T_0(c) = \mathbf{1}[\mathrm{Schema}_S(c) = \mathrm{pass}]\,, \tag{1}$$

$$T_1(c) = \mathbf{1}[\mathrm{DryRun}_S(c) = \mathrm{pass}]\,, \tag{2}$$

$$T_2(c; y^*) = \mathbf{1}[\mathrm{Run}_S(c) = \mathrm{pass} \ \wedge \ O(S(c)) \simeq_\epsilon y^*]\,. \tag{3}$$

T_0 checks the schema, T_1 requires `palace --dry-run` to succeed, and T_2 requires a completed run whose numerical CSV outputs conform to the reference. We report

$$H_0 = T_0, \qquad H_1 = T_0 \wedge T_1, \qquad H_2 = T_0 \wedge T_1 \wedge T_2. \tag{4}$$

We call these structural conformance, operational acceptance, and reference conformance. H_2 is an instance-level judgment over selected benchmark outputs, not a formal proof covering all program behaviors or physical properties. The relation $\simeq_\epsilon$ uses rel_tol $= 10^{-3}$ and abs_tol $= 10^{-12}$ (validated in Section 3.1) with $|y - y^*| \le$ abs_tol + rel_tol$|y^*|$. The comparator requires every reference CSV to be present, matching header sets and row counts, unique header names, and finite numeric cells. Because $\simeq_\epsilon$ compares selected quantities of interest rather than cell-by-cell fields, and the adaptive-mesh-refinement error indicators are checked only for presence, the hierarchy extends to non-deterministic or adaptive solvers. Such solvers would require comparing converged quantities of interest under a tolerance calibrated from repeated runs of the reference configuration.

2.2 Specification sufficiency and benchmark interpretation

A repair instance is $\tau = (x, c^-, z, y^*)$, where x is the repair request, c^- is the perturbed input configuration obtained by applying one of the benchmark's deterministic perturbations to a reference configuration, z is any auxiliary material visible to the model, and y^* is the benchmark reference

[1] Benchmark: `https://huggingface.co/datasets/empirischtech/Palace-Config-Repair`. Fine-tuned adapter: `https://huggingface.co/empirischtech/Qwen2.5-Coder-32B-Palace-LoRA`. The shipped cases are redistributed from Palace v0.14.0 under its Apache-2.0 license and the simulations were produced on NumericalAI (numericalai.net).

Table 1: The six perturbation classes used in $\mathcal{D}_{\text{ship}}$ and $\mathcal{D}_{\text{new}}$.

Class	Change made to the configuration	First failed outcome
`drop_required_section`	remove a required section	H_0
`wrong_type`	replace a number with a string	H_0
`bad_enum`	replace an allowed value with an invalid value	H_0 or H_1
`unit_slip`	multiply `Model.L0` by 10^3, 10^{-3}, or 10^6	H_2
`dangling_ref`	reference an identifier absent from the mesh	H_2
`multi_error`	combine two perturbations	varies

output. Here, *perturbed* means altered relative to the benchmark target and it does not necessarily imply that the configuration is schema-invalid or rejected by the solver. The perturbation classes are summarized in Table 1.

Let $\mathcal{Y}(x, c^-, z)$ denote the set of benchmark-output behaviors compatible with the information visible to the model, and let $\mathcal{E}_\epsilon(y^*) = \{y : y \simeq_\epsilon y^*\}$ denote the set of outputs that match the benchmark reference within the comparator tolerance. We define

$$\mathrm{Suff}(\tau) = \mathbf{1}\left[\mathcal{Y}(x, c^-, z) \neq \varnothing \ \wedge \ \mathcal{Y}(x, c^-, z) \subseteq \mathcal{E}_\epsilon(y^*)\right], \tag{5}$$

which assesses whether the information available to the model is sufficient to determine the behavior expected by the benchmark. The nonempty-set condition requires that at least one behavior be compatible with the visible information, while the subset condition requires every compatible behavior to agree with y^* within the comparator tolerance. Thus, $\mathrm{Suff}(\tau) = 1$ only when the visible information rules out materially different behaviors that would fail the benchmark comparator.

The purpose of this condition is to distinguish failure to repair a sufficiently specified task from failure caused by missing target information. When $\mathrm{Suff}(\tau) = 0$, failure at H_2 cannot be attributed solely to the repair system because multiple numerical behaviors remain compatible with the information visible to the model. In such cases, clarification, additional context, or abstention may be more appropriate than automatic repair. We use (5) explicitly when constructing and interpreting the benchmark. The 12 length-scale tasks (details in Section 3.2) in each evaluation set have $\mathrm{Suff}(\tau) = 0$ by construction, which motivates reporting reference conformance both on the full evaluation set and on a filtered set that excludes these known specification-insufficient tasks.

3 From solver artifacts to a corpus and a benchmark

All corpus and evaluation artifacts use $S := \text{Palace v0.14.0}$. The source collection comprises 121 documentation chunks, 435 flattened schema-option records, and 60 example records. After synthesis and filtering, the final corpus contains 3,296 training conversations and 199 validation conversations.

The evaluation benchmark starts from six unperturbed reference configurations, which together form the reference set $\mathcal{D}_{\text{ref}}$. Three are Palace shipped configurations whose underlying examples occur in the fine-tuning corpus. The remaining three are newly authored case-held-out configurations created after the model and retrieval index were frozen.

Each reference configuration is expanded using the six deterministic perturbation classes shown in Table 1, with three fixed variants per class. The six classes are selected so that failures are distributed across the verification hierarchy rather than concentrated at the schema level. The classes `drop_required_section`, `wrong_type`, and `bad_enum` fail structurally, whereas `unit_slip` and `dangling_ref` leave the configuration valid and executable and fail only at H_2. The `multi_error` class combines two perturbations to test whether the repair pipeline can resolve more than one issue in the same configuration.

The expansion produces $6 \times 3 = 18$ repair tasks per reference configuration. Since each group contains three reference configurations, the $18 \times 3 = 54$ tasks derived from the shipped configurations form $\mathcal{D}_{\text{ship}}$, while the 54 tasks derived from the newly authored configurations form $\mathcal{D}_{\text{new}}$. The complete perturbed evaluation set is $\mathcal{D}_{\text{pert}} = \mathcal{D}_{\text{ship}} \cup \mathcal{D}_{\text{new}}$, containing $54 \times 2 = 108$ repair tasks. Figure 1 in Appendix C summarizes the construction of these evaluation sets.

3.1 Execution oracle validation

We first verify that the execution oracle and grading pipeline behave as intended on the benchmark artifacts using the six unperturbed configurations in $\mathcal{D}_{\text{ref}}$ and the 108 perturbed configurations in $\mathcal{D}_{\text{pert}}$. All six configurations in $\mathcal{D}_{\text{ref}}$ pass H_0, H_1, and H_2. The relative differences between the solver-generated outputs and the stored reference outputs used for T_2 validation are all well below rel_tol $= 10^{-3}$.

All 108 perturbed configurations in $\mathcal{D}_{\text{pert}}$ fail H_2 before repair. Two perturbed configurations in $\mathcal{D}_{\text{new}}$ pass raw T_2 while failing T_0, confirming the need to define H_2 as the joint satisfaction of T_0, T_1, and T_2. The configurations in $\mathcal{D}_{\text{ref}}$ serve as positive controls, while the 108 perturbed configurations in $\mathcal{D}_{\text{pert}}$ serve as negative controls for H_2.

3.2 Repair task construction

Each of the 108 perturbed configurations in $\mathcal{D}_{\text{pert}}$ defines one repair task. This yields 18 tasks per case and 54 tasks in each of $\mathcal{D}_{\text{ship}}$ and $\mathcal{D}_{\text{new}}$.

The 12 tasks in each of $\mathcal{D}_{\text{ship}}$ and $\mathcal{D}_{\text{new}}$ modify `Model.L0`, the Palace parameter that converts mesh units into physical length units. Changing `Model.L0` changes the physical size of the simulated geometry even when the configuration remains schema valid and executable. Nine tasks contain only this length-scale change, while three combine it with another perturbation. For these tasks, the prompt does not specify the correct value of `Model.L0` or provide enough information to determine it uniquely. They therefore satisfy $\text{Suff}(\tau) = 0$ under (5). Their H_2 outcomes consequently depend not only on repairing the configuration but also on recovering a benchmark value that is not specified in the task. We therefore report results on all 54 tasks and separately on the $54 - 12 = 42$ tasks that do not modify `Model.L0`, which separates repair capability from specification sufficiency.

4 Systems evaluated

We compare five system settings on both $\mathcal{D}_{\text{ship}}$ and $\mathcal{D}_{\text{new}}$. The three shipped reference configurations underlying $\mathcal{D}_{\text{ship}}$ are represented in the fine-tuning corpus, whereas the three configurations underlying $\mathcal{D}_{\text{new}}$ were authored after the model and retrieval index were frozen. Evaluating both sets therefore distinguishes performance on case-exposed examples from transfer to newly authored cases.

We use Qwen2.5-Coder-32B-Instruct [Hui et al., 2024] as the stock model and adapt it to Palace using 4-bit QLoRA [Dettmers et al., 2023] and the training corpus described in Section 3. The retrieval component uses a BM25 index [Robertson and Zaragoza, 2009] containing only the 121 documentation chunks and 435 schema-option records. The 60 example records are excluded, so no reference configuration underlying $\mathcal{D}_{\text{ship}}$ or $\mathcal{D}_{\text{new}}$ is retrievable at inference time. Case exposure therefore occurs only through the fine-tuning corpus, which contains the three shipped configurations underlying $\mathcal{D}_{\text{ship}}$ and none of those underlying $\mathcal{D}_{\text{new}}$. Retrieval-enabled settings also use a deterministic calculator so that basic arithmetic and unit conversions are handled deterministically rather than left to model generation. In preliminary runs without the calculator, the model made basic arithmetic and unit-conversion mistakes. Because retrieval and calculator access are introduced together, we treat this condition as a RAG proxy and do not attribute its effect to either component separately.

When repair is enabled, a generated configuration that fails schema validation is returned to the model together with the validation errors for one additional attempt. Table 2 summarizes the five settings. All settings use the same tasks, prompts, and decoding parameters and are evaluated separately on both $\mathcal{D}_{\text{ship}}$ and $\mathcal{D}_{\text{new}}$.

5 Results

We evaluate all five local system settings separately on $\mathcal{D}_{\text{ship}}$ and $\mathcal{D}_{\text{new}}$. The central question is whether gains observed on case-exposed configurations transfer to newly authored cases, and whether that transfer changes as the verification requirement becomes stronger.

Table 2: The five system settings evaluated on $\mathcal{D}_{\text{ship}}$ and $\mathcal{D}_{\text{new}}$.

Setting	Description
stock	Base model without fine-tuning, retrieval, calculator, or repair
stock + RAG proxy	Base model with Palace retrieval and the deterministic calculator
fine-tune	Palace fine-tuned model without retrieval, calculator, or repair
fine-tune + RAG proxy	Fine-tuned model with Palace retrieval and the deterministic calculator
fine-tune + RAG proxy + repair	Fine-tuned model with retrieval, calculator, and one schema-guided repair attempt

Table 3: Outcomes for the five local system settings on $\mathcal{D}_{\text{ship}}$ and $\mathcal{D}_{\text{new}}$. The first three outcome columns report all 54 tasks. The final column excludes the 12 tasks that modify `Model.L0`.

System	H_0/54	H_1/54	H_2/54	Filtered H_2/42
Case-exposed tasks $\mathcal{D}_{\text{ship}}$				
stock	23 (43%)	21 (39%)	8 (15%)	8 (19%)
stock + RAG proxy	25 (46%)	17 (31%)	6 (11%)	6 (14%)
fine-tune	46 (85%)	45 (83%)	26 (48%)	26 (62%)
fine-tune + RAG proxy	51 (94%)	49 (91%)	32 (59%)	32 (76%)
fine-tune + RAG proxy + repair	**52 (96%)**	**50 (93%)**	**34 (63%)**	**34 (81%)**
Case-held-out tasks $\mathcal{D}_{\text{new}}$				
stock	2 (4%)	2 (4%)	2 (4%)	2 (5%)
stock + RAG proxy	14 (26%)	12 (22%)	7 (13%)	7 (17%)
fine-tune	41 (76%)	41 (76%)	10 (19%)	10 (24%)
fine-tune + RAG proxy	38 (70%)	34 (63%)	15 (28%)	15 (36%)
fine-tune + RAG proxy + repair	**48 (89%)**	**44 (81%)**	**22 (41%)**	**22 (52%)**

5.1 Performance on the case-exposed and newly authored task sets

Table 3 reports structural conformance, operational acceptance, and reference conformance for both evaluation sets.

The first finding is a clear performance gap between $\mathcal{D}_{\text{ship}}$ and $\mathcal{D}_{\text{new}}$. For the full pipeline, H_0 decreases only from 52/54 to 48/54, while H_2 falls more substantially from 34/54 to 22/54. We also evaluate the filtered sets after removing the 12 `Model.L0` tasks known to have $\text{Suff}(\tau) = 0$ under (5). Reference conformance still drops from 34/42 on $\mathcal{D}_{\text{ship}}$ to 22/42 on $\mathcal{D}_{\text{new}}$. The performance gap therefore cannot be explained only by the deliberately underspecified length-scale tasks. The pipeline transfers the structural and operational conventions of Palace more reliably than the case-specific repair decisions needed to reproduce the intended numerical behavior.

The same pattern is already visible after fine-tuning alone. Fine-tuning reaches 26/54 at H_2 on $\mathcal{D}_{\text{ship}}$ but only 10/54 on $\mathcal{D}_{\text{new}}$. Since the underlying shipped configurations occur in the fine-tuning corpus, this is consistent with case exposure contributing to repair performance. The result does not isolate memorization because the two sets contain different physical cases, but it shows that evaluation only on $\mathcal{D}_{\text{ship}}$ would substantially overstate transfer to newly authored configurations.

A second finding is that operational acceptance remains a weak proxy for reference conformance. Under the full pipeline, 50/54 repair tasks in $\mathcal{D}_{\text{ship}}$ produce outputs that pass H_1, but only 34/54 pass H_2. The gap is larger on $\mathcal{D}_{\text{new}}$, where 44/54 pass H_1 but only 22/54 reproduce the reference. Successful solver acceptance therefore becomes even less informative about numerical correctness on newly authored cases.

5.2 Effect of fine-tuning, the RAG proxy, and repair

Fine-tuning is the dominant intervention across both evaluation sets. On $\mathcal{D}_{\text{ship}}$, it increases H_2 from 8/54 for the stock model to 26/54. On $\mathcal{D}_{\text{new}}$, the corresponding increase is from 2/54 to 10/54. This shows that domain adaptation substantially improves Palace repair performance for the local Qwen model, although the gain transfers much less strongly to newly authored cases.

Table 4: Outcomes for zero-shot frontier models on $\mathcal{D}_{\text{new}}$ together with the stock and full local settings. The final column excludes the 12 tasks that modify `Model.L0`.

System	H_0/54	H_1/54	H_2/**54**	**Filtered** H_2/**42**
stock	2 (4%)	2 (4%)	2 (4%)	2 (5%)
fine-tune + RAG proxy + repair	48 (89%)	44 (81%)	22 (41%)	22 (52%)
GPT-5	5 (9%)	5 (9%)	3 (6%)	3 (7%)
DeepSeek-V4-Pro	16 (30%)	16 (30%)	14 (26%)	13 (31%)
Claude Opus 5	54 (100%)	54 (100%)	36 (67%)	34 (81%)

The RAG-proxy setting has a more conditional effect. When added to the fine-tuned model, H_2 increases from 26/54 to 32/54 on $\mathcal{D}_{\text{ship}}$ and from 10/54 to 15/54 on $\mathcal{D}_{\text{new}}$. In contrast, adding the same RAG proxy to the stock model lowers H_2 from 8/54 to 6/54 on $\mathcal{D}_{\text{ship}}$ while increasing it from 2/54 to 7/54 on $\mathcal{D}_{\text{new}}$. Because this setting combines retrieved Palace context with deterministic calculation, these comparisons establish the effect of the combined RAG-proxy setting rather than retrieval alone.

Schema-guided repair provides a further gain, particularly on the newly authored cases. It raises H_2 from 32/54 to 34/54 on $\mathcal{D}_{\text{ship}}$ but from 15/54 to 22/54 on $\mathcal{D}_{\text{new}}$. The larger improvement on $\mathcal{D}_{\text{new}}$ suggests that validator feedback helps recover a meaningful subset of errors that remain after fine-tuning and the RAG-proxy setting. However, it cannot resolve configurations that already pass the structural checks but still encode the wrong numerical behavior.

Taken together, the system comparison shows a progression in capability. Fine-tuning provides the largest gain, the RAG-proxy setting provides additional gains, and validator feedback repairs a narrower set of remaining errors. None of these interventions closes the gap between operational acceptance and reference conformance.

5.3 Frontier models on the newly authored task set

To place the local results in context, Table 4 compares three frontier model endpoints with the stock model and the full local pipeline on $\mathcal{D}_{\text{new}}$. The frontier models are used as zero-shot repair generators and receive the same repair requests but no fine-tuning, retrieval, calculator, or schema-guided repair attempt.

The frontier results show that stronger general-purpose models can substantially improve repair on newly authored cases, but they do not eliminate the distinction between structural and numerical correctness. Claude Opus 5 passes H_0 and H_1 on all 54 tasks, yet only 36/54 outputs satisfy H_2. After excluding the known length-scale tasks, it reaches 34/42 at H_2, compared with 22/42 for the full local pipeline. The remaining gap despite perfect structural and operational performance provides further evidence that the difficulty at H_2 is not simply malformed syntax or inability to satisfy the solver interface.

The frontier models also span a wide range of reference-conformance performance on $\mathcal{D}_{\text{new}}$. GPT-5 reaches 3/54 at H_2, DeepSeek-V4-Pro reaches 14/54, and Claude Opus 5 reaches 36/54. This variation suggests that base model capability matters for the task. However, the experiment does not isolate model size because the endpoints differ in training data, architecture, inference behavior, and provider settings. The stronger frontier performance should therefore be interpreted as evidence that the local 32B pipeline has remaining capability limitations, not as proof that parameter count alone explains the gap.

6 LLM judge audit against execution

We next ask whether an LLM judge can substitute for execution-based reference comparison. Each judge receives a produced configuration and a one-line case description, but no reference output or execution trace. We evaluate Claude Opus 5, GPT-5, and DeepSeek-V4-Pro against the execution-based H_2 labels. The panel is cross-family with respect to the Qwen generator, reducing one form of self-preference [Panickssery et al., 2024].

Table 5: LLM judge performance against execution-based H_2 labels for system-task outputs from $\mathcal{D}_{\text{ship}}$ and $\mathcal{D}_{\text{new}}$. FP and FN rates are calculated over H_2 negative and H_2 positive outputs, respectively.

Panel	Judge	Accuracy	Balanced acc.	FP rate	FN rate
$\mathcal{D}_{\text{ship}}$	Claude Opus 5	65%	59%	1%	82%
	DeepSeek-V4-Pro	60%	61%	46%	31%
	GPT-5	55%	48%	15%	89%
	majority vote	61%	55%	10%	79%
$\mathcal{D}_{\text{new}}$	Claude Opus 5	92%	95%	10%	0%
	DeepSeek-V4-Pro	66%	69%	37%	25%
	GPT-5	63%	67%	41%	25%
	majority vote	80%	86%	25%	4%

The audit covers every system-task output for which the local system produced a configuration. With five systems and 54 tasks, each evaluation set contains $5 \times 54 = 270$ possible system-task outputs. Of these, 255 outputs in $\mathcal{D}_{\text{ship}}$ and 260 in $\mathcal{D}_{\text{new}}$ contain a produced configuration and can therefore be evaluated by the LLM judges. The remaining 15 and 10 attempts, respectively, produced no judgeable configuration and are excluded. Each of the 515 produced configurations receives one verdict from each of the three judges, giving $515 \times 3 = 1{,}545$ verdicts in total.

No judge reproduces the execution labels consistently across both evaluation sets. The clearest example is Claude Opus 5. On $\mathcal{D}_{\text{ship}}$, its 1% false positive rate is accompanied by an 82% false negative rate, meaning that it rejects most system-task outputs that actually satisfy H_2. On $\mathcal{D}_{\text{new}}$, the same judge reaches 95% balanced accuracy with no false negatives. Judge reliability therefore changes substantially between the two sets.

The other judges show different error patterns. DeepSeek-V4-Pro has false positive rates of 46% on $\mathcal{D}_{\text{ship}}$ and 37% on $\mathcal{D}_{\text{new}}$, while GPT-5 misses 89% of the H_2 positive outputs on $\mathcal{D}_{\text{ship}}$ but only 25% on $\mathcal{D}_{\text{new}}$. Majority voting does not remove this variation. Its balanced accuracy changes from 55% on $\mathcal{D}_{\text{ship}}$ to 86% on $\mathcal{D}_{\text{new}}$.

These results show that blind LLM judging is not a stable substitute for the execution oracle in this benchmark. A judge that performs well on one set of physical cases can behave very differently on another. When an objective solver-derived label is available, execution-based reference comparison therefore provides a more dependable correctness signal.

7 Conclusion

Scientific configuration repair requires stronger checks than schema conformance or solver acceptance. On Palace, both consistently overstate numerical reference conformance, with the gap becoming larger on newly authored cases. This shows that improvements on case-exposed examples do not fully transfer to unseen configurations.

The study also shows that execution can verify agreement with a known reference but cannot recover a target that the specification does not provide. Blind LLM judges likewise do not reliably replace the execution oracle. Together, these results support combining schema checks, solver validation, and execution-based reference comparison when evaluating generated scientific configurations.

Acknowledgments and Disclosure of Funding

The authors would like to thank the Austrian Federal Ministry of Finance and the Austrian Research Promotion Agency (FFG) for funding this work through the LUMEIK-5G project under the grant agreement n. FO999923407. LUMEIK-5G is funded under the research and technology development of gigabit applications as part of the light-house projects of Breitband Austria 2030: GigaApp, which was initiated by the Austrian Federal Ministry of Agriculture, Regions, and Tourism.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021. doi: 10.48550/arXiv.2108.07732. URL https://arxiv.org/abs/2108.07732.

AWS Center for Quantum Computing. Palace: 3d finite element solver for computational electromagnetics. Software release v0.14.0, August 2025. URL https://github.com/awslabs/palace/releases/tag/v0.14.0. Version 0.14.0.

Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. SWE-rebench: an automated pipeline for task collection and decontaminated evaluation of software engineering agents. In *Advances in Neural Information Processing Systems*, volume 38, pages 26420–26466, 2025. doi: 10.52202/085713-0788.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. doi: 10.48550/arXiv.2107.03374. URL https://arxiv.org/abs/2107.03374.

Xun Deng, Sicheng Zhong, Andreas Veneris, Fan Long, and Xujie Si. VerifyThisBench: generating code, specifications, and proofs all at once. *arXiv preprint arXiv 2505.19271*, 2025. doi: 10.48550/arXiv.2505.19271.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems*, volume 36, pages 10088–10115, 2023. doi: 10.52202/075280-0441.

E Fan, Kang Hu, Zhuowen Wu, Jiangyang Ge, Jiawei Miao, Yuzhi Zhang, He Sun, Weizong Wang, and Tianhan Zhang. Chatcfd: A large language model-driven agent for end-to-end computational fluid dynamics automation with structured knowledge and reasoning. *Advanced Intelligent Discovery*, 2:e202500174, 2026. doi: 10.1002/aidi.202500174.

Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. Grammar-constrained decoding for structured NLP tasks without finetuning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 10932–10952, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.674. URL https://aclanthology.org/2023.emnlp-main.674/.

Taher A. Ghaleb. Doc2CI: a multi-service study of CI configuration generation using large language models. *arXiv preprint arXiv:2608.01451*, 2026. doi: 10.48550/arXiv.2608.01451. URL https://arxiv.org/abs/2608.01451.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-Coder technical report. *arXiv preprint arXiv:2409.12186*, 2024. doi: 10.48550/arXiv.2409.12186. URL https://arxiv.org/abs/2409.12186.

Hongchao Jiang, Yiming Chen, Yushi Cao, Hung-yi Lee, and Robby T. Tan. CodeJudgeBench: benchmarking LLM-as-a-judge for coding tasks. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, pages 19416–19448. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.888.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474, 2020.

Niels Mündler, Mark Niklas Müller, Jingxuan He, and Martin Vechev. SWT-bench: testing and validating real-world bug-fixes with code agents. In *Advances in Neural Information Processing Systems*, volume 37, pages 81857–81887, 2024. doi: 10.52202/079017-2601.

Arjun Panickssery, Samuel R. Bowman, and Shi Feng. LLM evaluators recognize and favor their own generations. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-2197.

Ioannis Protogeros, Rufat Asadli, Benjamin Hoffman, and Laurent Vanbever. Benchmarking LLM-driven network configuration repair. *arXiv preprint arXiv:2604.22513*, 2026. doi: 10.48550/arXiv.2604.22513. URL https://arxiv.org/abs/2604.22513.

Stephen E. Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009. doi: 10.1561/1500000019.

Nithin Somasekharan, Ling Yue, Yadi Cao, Weichao Li, Patrick Emami, Pochinapeddi Sai Bhargav, Anurag Acharya, Xingyu Xie, and Shaowu Pan. CFDLLMBench: a benchmark suite for evaluating large language models in computational fluid dynamics. *Journal of Data-centric Machine Learning Research*, 3(13):1–40, 2026. URL https://openreview.net/forum?id=kTcH1MnkjY.

Brandon T. Willard and Rémi Louf. Efficient guided generation for large language models. *arXiv preprint arXiv:2307.09702*, 2023. doi: 10.48550/arXiv.2307.09702. URL https://arxiv.org/abs/2307.09702.

Ling Yue, Nithin Somasekharan, Tingwen Zhang, Yadi Cao, Zhangze Chen, Shimin Di, and Shaowu Pan. Foam-Agent: a large language model-based multi-agent framework for automating computational fluid dynamics workflows. *Computer Methods in Applied Mechanics and Engineering*, 461: 119271, 2026. doi: 10.1016/j.cma.2026.119271.

Tianyi Zhang, Shidong Pan, Zejun Zhang, Zhenchang Xing, and Xiaoyu Sun. Deployability-centric infrastructure-as-code generation: fail, learn, refine, and succeed through LLM-empowered DevOps simulation. *Proceedings of the ACM on Software Engineering*, 3(FSE):321–343, 2026. doi: 10.1145/3797142.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-2020.

A Related work

Configuration generation and repair. Recent work on configuration generation increasingly evaluates generated artifacts beyond syntactic validity. Doc2CI evaluates continuous integration configurations using schema validity and reference alignment and shows that improvements in structural correctness do not necessarily produce equivalent gains in reference agreement [Ghaleb, 2026]. In infrastructure as code, Zhang et al. [2026] find that syntactically correct templates frequently fail during deployment and use iterative execution feedback to improve generation. Cornetto [Protogeros et al., 2026] evaluates network configuration repair by formally checking the resulting network behavior against explicit specifications. These works establish that parsing and schema validity are insufficient when a configuration determines downstream behavior.

Our setting extends this principle to scientific solver configurations. Palace configurations encode continuous physical simulations in which a file can satisfy the schema, be accepted by the solver, and execute successfully while still producing numerical behavior different from the benchmark target. We therefore separate schema conformance, operational acceptance, and numerical reference conformance and evaluate the strongest outcome through solver execution against stored regression references.

Scientific workflow generation and evaluation. LLM-based automation of scientific simulation has increasingly incorporated executable evaluation. CFDLLMBench [Somasekharan et al., 2026] evaluates CFD knowledge, numerical reasoning, and end-to-end OpenFOAM workflow generation, with FoamBench measuring both executability and numerical agreement with reference solutions. ChatCFD [Fan et al., 2026] similarly distinguishes simulation execution from physical fidelity and reports a substantial gap between the two. Foam-Agent [Yue et al., 2026] combines structured retrieval, dependency-aware OpenFOAM file generation, and execution-guided correction, while separately measuring crash-free execution and solution fidelity. These studies show that successful execution is not sufficient evidence of physical correctness.

Our work focuses specifically on configuration repair and makes the transition from structural validity to numerical behavior explicit within a single verification hierarchy. The same repaired artifact is evaluated for schema conformance, solver acceptance, and reference conformance. This allows us to identify configurations that pass the first two levels but still reproduce the wrong numerical result.

Structured generation and verifier feedback. Structured generation methods constrain model output using finite-state or regular-expression-guided decoding [Willard and Louf, 2023] and grammar-constrained generation [Geng et al., 2023]. These methods improve syntactic reliability but do not establish behavioral correctness. More generally, VerifyThisBench [Deng et al., 2025] evaluates LLMs on tasks that jointly require specification, implementation, and formal verification, illustrating a stronger form of machine-checkable correctness than syntax alone.

Our repair pipeline uses schema validation as feedback rather than as the final correctness criterion. A schema-failing output receives one additional repair attempt, after which the configuration is still required to satisfy the solver and numerical reference checks. This distinguishes verifier-guided repair from approaches whose evaluation terminates once structural constraints are satisfied.

Execution-based evaluation and case exposure. Executable tests provide the standard correctness signal in code generation benchmarks such as HumanEval [Chen et al., 2021] and MBPP [Austin et al., 2021]. SWE-bench [Jimenez et al., 2024] extends execution-based evaluation to repository-level software engineering tasks, while SWT-Bench [Mündler et al., 2024] evaluates generated fixes through tests constructed around real software issues. SWE-rebench [Badertdinov et al., 2025] addresses a related evaluation problem by collecting fresh tasks that reduce the risk of benchmark exposure during model training.

Both issues are relevant to our design. We extend execution-based evaluation from unit tests to numerical reference reproduction and report results separately on two task sets. $\mathcal{D}_{\mathrm{ship}}$ is derived from Palace cases whose underlying configurations occur in the fine-tuning corpus, while $\mathcal{D}_{\mathrm{new}}$ is derived from newly authored cases created after the model and retrieval index were frozen. The comparison therefore measures whether gains observed on case-exposed configurations transfer to newly authored cases.

LLM judges. LLMs are widely used to evaluate model outputs [Zheng et al., 2023], but their judgments can be sensitive to model family, prompt structure, and characteristics unrelated to correctness. Self-preference is one documented source of bias [Panickssery et al., 2024]. CodeJudgeBench [Jiang et al., 2026] evaluates LLM judges across code generation, code repair, and test generation and reports substantial variation and instability across judges.

Our judge audit uses an execution-based label rather than human preference as the comparison target. The judges do not generate repairs. They predict whether configurations generated by the local systems satisfy H_2, and their decisions are compared directly with the numerical reference conformance labels. This allows false positive and false negative behavior to be measured against an independent executable oracle.

B Detailed LLM judge results

The main text reports judge performance separately for $\mathcal{D}_{\text{ship}}$ and $\mathcal{D}_{\text{new}}$. Here we further break down the results by underlying case to examine whether the large difference between the two evaluation sets is driven by particular physical cases.

B.1 Case-level results

Table 6: LLM judge accuracy and balanced accuracy by underlying case. The H_2^-/H_2^+ column reports the numbers of H_2-negative and H_2-positive system-task outputs. A / B denotes accuracy and balanced accuracy, respectively, in percent.

Set	Case	H_2^-/H_2^+	Claude Opus 5 A / B	DeepSeek-V4-Pro A / B	GPT-5 A / B	Majority A / B
$\mathcal{D}_{\text{ship}}$	spheres	63/25	88/78	64/69	59/50	76/71
$\mathcal{D}_{\text{ship}}$	rings	52/36	61/53	57/62	60/53	61/54
$\mathcal{D}_{\text{ship}}$	coaxial	34/45	46/52	59/62	43/50	44/51
$\mathcal{D}_{\text{new}}$	parallel plate	73/15	90/94	60/63	64/70	74/84
$\mathcal{D}_{\text{new}}$	layered capacitor	66/20	91/94	67/67	62/59	83/85
$\mathcal{D}_{\text{new}}$	concentric sphere	65/21	95/97	70/77	63/72	84/89

The case-level results show that the difference between $\mathcal{D}_{\text{ship}}$ and $\mathcal{D}_{\text{new}}$ is not merely an aggregate effect. Judge behavior varies substantially across individual cases. The strongest example is coaxial, where GPT-5 identifies none of the 45 H_2 positive outputs as correct and Claude Opus 5 identifies only two. In contrast, Claude Opus 5 produces no false negatives across any of the three newly authored cases. This variation supports the main finding that judge reliability is strongly dependent on the evaluated case.

B.2 Perturbation class effects

Judge behavior also varies across perturbation classes and does not follow a consistent pattern between $\mathcal{D}_{\text{ship}}$ and $\mathcal{D}_{\text{new}}$. For `unit_slip`, GPT-5 incorrectly accepts 8 of 42 H_2 negative outputs on $\mathcal{D}_{\text{ship}}$ but 25 of 45 on $\mathcal{D}_{\text{new}}$, while DeepSeek-V4-Pro changes in the opposite direction from 21 of 42 to 11 of 45. Claude Opus 5 also changes substantially on `bad_enum`, with 20 false negatives among 25 H_2 positive outputs on $\mathcal{D}_{\text{ship}}$ and none among 14 on $\mathcal{D}_{\text{new}}$. These comparisons are descriptive because the underlying cases and evaluation set change together. The `unit_slip` results additionally reflect the missing target information discussed in Section 2.2.

C Evaluation set construction

Figure 1 summarizes how the evaluation sets are constructed from the reference configurations.

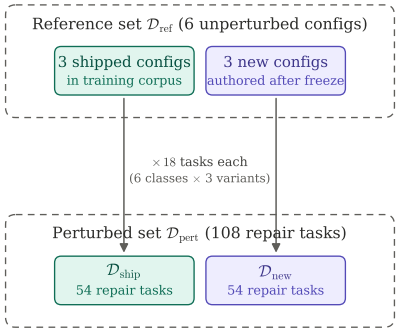


Figure 1: Construction of the evaluation sets. The six unperturbed reference configurations in $\mathcal{D}_{\text{ref}}$ are each expanded into $6 \times 3 = 18$ repair tasks, yielding $\mathcal{D}_{\text{ship}}$ and $\mathcal{D}_{\text{new}}$ with 54 tasks each and $\mathcal{D}_{\text{pert}} = \mathcal{D}_{\text{ship}} \cup \mathcal{D}_{\text{new}}$ with 108 tasks in total.