From Scientific Specifications to Executable Simulation Workflows: Representation and Completeness in LLM-Mediated Scientific Computing

Anonymous ACL submission

Abstract

Scientific simulation workflows are difficult to automate because physical specifications must be translated into software-specific inputs, mesh and solver controls, execution scripts, and validation routines. Large language model (LLM) agents provide a promising interface for this translation, but scientific simulation differs from ordinary code generation: a workflow can be syntactically valid and runnable while still drifting from the intended physical problem. We study LLM-mediated simulation workflow generation across OpenFOAM, SU2, CalculiX, and OpenPhase, covering FVM-based CFD, FEM solid mechanics, and grid-based phase-field simulation. For each task, we compare math-oriented Markdown, natural-language text, and tabular CSV specifications, and evaluate generated packages by workflow executability, physical faithfulness, and LLM-side efficiency. We further conduct an OpenFOAM ablation study to test how missing governing models, boundary conditions, mesh information, material parameters, numerical schemes, and time controls affect generated workflows. The agent reliably assembles multi-file workflow packages: all 12 base settings contain complete files, syntactically valid configurations, and mesh-ready setups. However, validation-ready scientific evidence is harder than file generation or short execution: 9/12 settings produce expected output artifacts and only 6/12 are validation-ready in the recorded run, often because the agent performs a smoke test or demonstration while leaving scripts for full execution. Our analysis shows that input representation and physical-information completeness act as control surfaces for reliability, shaping whether the agent preserves scientific constraints, exposes assumptions, and performs enough work for validation. These results argue for specification-aware, assumption-aware evaluation of LLM scientific agents rather than executability checks alone.

1 Introduction

Scientific simulation workflows are difficult to automate because they require a structured translation from physical intent to executable software artifacts. A typical workflow may involve selecting governing equations, defining geometry and mesh, assigning material parameters, specifying boundary and initial conditions, configuring numerical solvers, running the simulation, and extracting validation quantities. This translation is routine for experienced users but remains time-consuming, brittle, and highly dependent on software conventions.

LLM-based coding agents create a new opportunity: instead of asking users to manually assemble solver inputs and scripts, a user can provide a scientific specification and ask the model to construct the workflow. This turns scientific simulation into a language-mediated systems problem. The model must not only write code, but also connect natural language, mathematical notation, tabular parameters, file formats, command-line tools, solver logs, and post-processing outputs.

This setting is more demanding than ordinary code generation. A workflow can be syntactically valid and even executable, yet still solve the wrong physical problem. For example, an omitted mesh-scaling operation can silently change the effective Reynolds number; a smoke test can be mistaken for a completed transient simulation; or a generated post-processing script may exist without producing validation-ready drag, lift, displacement, or phase-field quantities. Thus, the central question is not merely whether LLMs can generate simulation files, but whether they can generate executable and physically faithful workflows from scientific specifications.

This work connects to research on LLM code generation and program synthesis, including function-level synthesis, competitive programming,

multilingual code generation, data-science code, and real-world software maintenance benchmarks (Chen et al., 2021; Austin et al., 2021; Hendrycks et al., 2021a; Lu et al., 2021; Ren et al., 2020; Li et al., 2022; Nijkamp et al., 2023; Cassano et al., 2023; Roziere et al., 2023; Lai et al., 2022; Jimenez et al., 2024; Xu et al., 2022). It also relates to agentic language-model systems that interleave reasoning, tool use, planning, feedback, and external execution (Yao et al., 2022; Schick et al., 2023; Yao et al., 2023; Shinn et al., 2023; Liu et al., 2023; Qin et al., 2024; Shen et al., 2023; Wang et al., 2023a). In scientific domains, prior work has developed scientific language models, STEM-reasoning benchmarks, and tool-augmented agents for chemistry, mathematics, machine-learning experimentation, and scientific coding (Beltagy et al., 2019; Lewkowycz et al., 2022; Taylor et al., 2022; Hendrycks et al., 2021b; Wang et al., 2023b; Bran et al., 2023; Boiko et al., 2023; Romera-Paredes et al., 2024; Huang et al., 2024; Tian et al., 2024; Microsoft Research AI4Science and Microsoft Azure Quantum, 2023). Our focus differs from these settings: the output is a multi-file scientific simulation workflow whose quality depends jointly on software execution, preservation of the physical specification, and validation-ready evidence.

We investigate three research questions:

- **RQ1:** Can an LLM agent generate integrated simulation workflow packages across heterogeneous scientific computing frameworks?
- **RQ2:** How do input representations affect workflow executability, physical faithfulness, and LLM-side efficiency?
- **RQ3:** How does missing physical information affect generated workflows and their failure modes?

We make four contributions. First, we formulate LLM-mediated simulation workflow generation as a task that maps scientific specifications to executable, verifiable, and auditable workflow packages. Second, we construct a cross-framework benchmark covering CFD, solid mechanics, and phase-field simulation using OpenFOAM, SU2, CalculiX, and OpenPhase. Third, we evaluate three input representations—math-oriented Markdown, natural-language text, and tabular CSV—using executability, physical faithfulness, and efficiency metrics. Fourth, we conduct an OpenFOAM physical-information ablation study and show that missing specification components induce distinct failure modes, including boundary-condition drift, mesh-scale errors, implicit material reconstruction, and insufficient time integration.

2 LLM-Mediated Simulation Workflow Generation

2.1 Task Definition

We define LLM-mediated simulation workflow generation as the transformation of a scientific specification S into a generated workflow package W. Unlike ordinary code generation, the target is not a single function or script, but a collection of software artifacts that must jointly instantiate a physical problem, support execution, and provide evidence for inspection. The input specification contains the information needed to define the intended simulation:

$$S = \{P, G, E, B, I, M, N, V\}, \tag{1}$$

where P is the physical problem, G is the geometry or computational domain, E is the governing model, B and I are boundary and initial conditions, M denotes material or physical parameters, N denotes numerical settings, and V denotes validation or output targets.

The generated workflow package contains the artifacts needed to realize and inspect the simulation:

$$W = \{F, C, R, D, O, L\}, \tag{2}$$

where F is the directory and file structure, C is solver configuration, R is execution logic, D is documentation or stated assumptions, O is post-processing and output, and L is execution or editing evidence such as logs. Figure 1 summarizes this pipeline. The agent reads a scientific specification, generates solver-specific files and scripts, attempts execution or demonstration runs, and leaves artifacts that are then evaluated for executability, physical faithfulness, and generation efficiency.

2.2 Workflow Artifacts

Table 1 summarizes the artifact types considered in this work. This artifact-level framing is important because a scientific workflow can fail at different stages. A package may contain valid solver inputs but lack a runnable script, may execute only a smoke test, or may produce fields without validation-ready quantities. We therefore treat the generated package as an auditable workflow rather than as isolated source code.

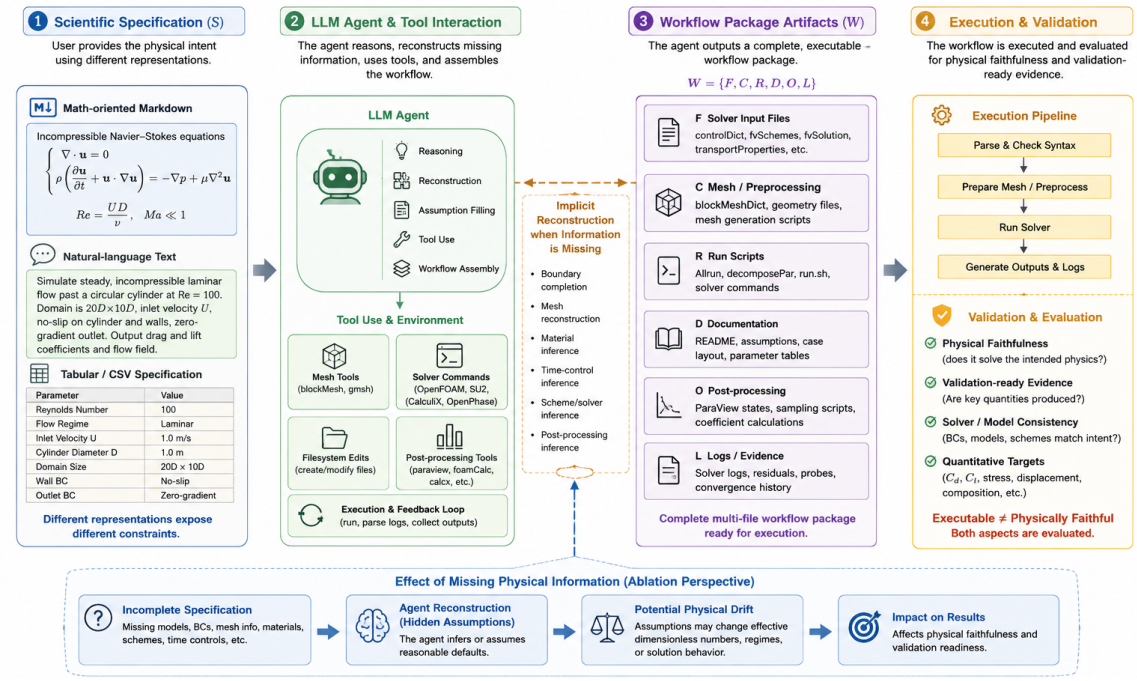


Figure 1: Pipeline for LLM-mediated scientific simulation workflow generation. A scientific specification is converted into a multi-file simulation workflow package, including solver inputs, preprocessing and execution scripts, documentation, post-processing outputs, and execution evidence. The generated package is then evaluated from complementary views: workflow executability, physical faithfulness, and LLM-side efficiency.

Artifact type	Examples
Solver inputs	OpenFOAM dictionaries, SU2 configuration, CalculiX input deck, OpenPhase input files
Preprocessing	Mesh generation or conversion scripts
Execution	Run scripts, solver commands, environment checks
Documentation	README files, stated assumptions, workflow notes
Post-processing	Force extraction, displacement extraction, field visualization
Evidence	Solver logs, partial outputs, editing traces

Table 1: Generated workflow artifacts evaluated in this work.

2.3 Evaluation View

We evaluate each generated workflow from three complementary perspectives. *Executability* measures whether the package is structurally complete, syntactically usable, mesh-ready, runnable in the recorded setting, and able to produce expected artifacts. *Physical faithfulness* measures whether the workflow preserves the intended model, geometry, boundary and initial conditions, material parameters, numerical choices, and validation targets. *Efficiency* measures the LLM-side cost of producing the workflow, including billable tokens, output tokens, command count, generated files, and tokens per file. Together, these views separate file-level usability from scientific correctness and from the cost of reconstructing missing or implicit workflow information.

3 Experimental Setup

Agent and execution environment. The workflow-generation agent used in this study is Codex, connected to GPT-5.5 as the underlying large language model. Local workflow inspection, script execution, and recorded simulation runs were performed under WSL Ubuntu 20.04 on a workstation with an AMD Ryzen 9 9950X3D CPU and an NVIDIA GeForce RTX 5080 GPU. The scientific solvers used in this benchmark were executed as local CPU-side workflows; the GPU was not used as an accelerator for OpenFOAM, SU2, CalculiX, or OpenPhase runs. This environment note is included for reproducibility, while solver wall-clock time is not treated as a hardware-performance claim in this study.

3.1 Simulation Backends and Cases

Table 2 lists four representative scientific-computing backends. OpenFOAM and SU2 cover FVM-based CFD with the same Re=100 cylinder-

flow problem but different software conventions. CalculiX covers FEM-based solid mechanics, and OpenPhase covers an FDM-style grid-based phase-field workflow. Together, the cases span fluid flow, structural mechanics, and microstructure evolution, testing whether LLM-generated workflows generalize beyond a single solver or physics domain.

The benchmark has two experimental axes. First, the 12 base settings cross four backends with three input representations, testing whether math-oriented Markdown, natural-language text, and tabular CSV lead to different workflow quality and cost. Second, an OpenFOAM ablation varies physical-information completeness, because missing boundary conditions, mesh scale, material values, time controls, or numerical schemes can change the simulated problem even when generated files are runnable.

3.2 Input Representations and Outcome Evaluation

For each base case, we provide the same task as math-oriented Markdown, natural-language text, and tabular CSV. These formats emphasize different information channels: equations and parameters, workflow intent, and structured engineering fields. Table 3 gives short excerpts; longer examples of the three input forms are provided in Appendix E.

We evaluate each representation by workflow executability, physical faithfulness, and LLM-side efficiency. Executability captures file completeness, syntax, mesh availability, recorded execution, and output artifacts. Physical faithfulness checks whether the generated setup preserves the intended model, conditions, scale, parameters, numerics, and validation evidence. Efficiency reports token cost, command count, generated files, and tokens per file.

3.3 Completeness of Simulation Physics Information and Outcome Evaluation

Beyond representation format, we test whether missing physical information changes the generated workflow. Using OpenFOAM cylinder flow, we construct a full baseline and six ablations removing governing equations, boundary conditions, numerical schemes, mesh information, governing equations plus material values, or time-step/end-time controls. This creates 21 ablation settings across the three input representations.

For each ablation, we record execution success, validation-file availability, boundary consistency, solver and numerical choices, mesh and material reconstruction, and LLM-side cost. ValidationOK means that the recorded run produced outputs supporting the intended scientific check, such as drag/lift histories or reference comparisons. ValidationOK=0 does not necessarily mean the workflow is wrong or impossible to complete: several cases use dry runs or short demonstrations while retaining scripts for full-length execution.

One natural-language ablation setting has missing efficiency-log data; it is excluded only from token-based efficiency summaries, not from execution and validation scoring. Billable tokens are computed as fresh input tokens plus output tokens plus reasoning tokens. The full scoring rubric is included in the supplementary material.

4 Agent-Generated Simulation Workflows

This section summarizes the direct outputs of the agentic simulation runs. The goal is to show what the user receives from the agent before analyzing how input representation and physical-information completeness affect reliability.

4.1 Generated Workflow Packages

The agent produces usable workflow packages rather than isolated code fragments. The generated packages typically include solver inputs, mesh conversion or generation steps, run scripts, README-style instructions, logs, and post-processing scripts. These artifacts reduce the user's manual burden: instead of writing case dictionaries, input decks, or execution commands from scratch, the user receives an executable directory that can be inspected, run, or extended. Representative generated file trees are shown in Appendix A.

The generated packages also differ in how directly they support subsequent analysis. Some workflows provide post-processing scripts or output summaries that can be inspected immediately. Others provide compact execution scripts or smoke-test logs, leaving full-length execution to the user because the agent judged the complete simulation to be costly in the interactive setting. This distinction is important: a demo run can still be paired with a reusable full-run script.

4.2 Simulation Outputs

The workflows also produce simulation artifacts that can be directly inspected or used as starting

Software	Numerical Method	Case	Inputs	Main generated artifacts
OpenFOAM	FVM	Re=100 cylinder flow	Math/NL/Table	Case dictionaries, mesh conversion, run scripts, force post-processing
SU2	FVM	Re=100 cylinder flow	Math/NL/Table	Configuration file, mesh, scripts, dry/smoke logs
CalculiX	FEM	Cantilever beam	Math/NL/Table	Input deck, run script, FRD output, displacement/stress post-processing
OpenPhase	FDM	Spinodal decomposition	Math/NL/Table	Input files, execution scripts, composition snapshots

Table 2: Benchmark cases and input representations.

Representation	Excerpt from OpenFOAM cylinder-flow input
Math Markdown	"Consider the incompressible laminar flow of a Newtonian fluid past a circular cylinder... $Re = U_{in}D/\nu$, with $D = 1$ m, $U_{in} = 1$ m/s, and $\nu = 0.01$ m^2/s."
Natural language	"Please create a complete and runnable two-dimensional incompressible laminar flow simulation case... The Reynolds number is 100, based on a cylinder diameter of 1 m and an inlet velocity of 1 m/s."
CSV/Table	"Software Framework, OpenFOAM; Problem, cylinder flow; Reynolds Number, 100; Cylinder Diameter, 1 m; Inlet Velocity, 1 m/s; Kinematic Viscosity, 0.01 m^2/s."

Table 3: Short prompt excerpts illustrating the three input representations.

points for further analysis. Figure 2 shows output fields generated from the workflow runs, including OpenFOAM and SU2 cylinder-flow fields, CalculiX displacement and stress fields, and an OpenPhase spinodal-decomposition field. For the cylinder-flow cases, the generated workflows produce velocity and pressure fields around the cylinder. For the cantilever-beam case, the workflow produces displacement and von Mises stress fields. For the spinodal-decomposition case, the workflow produces a composition field showing phase separation. These outputs show that the agent can connect specification, software setup, execution, and post-processing into an end-to-end workflow. They are not used alone as correctness evidence; the reliability analysis is reported in Section 5.

Metric	Result
FileComplete	12/12
SyntaxOK	12/12
MeshOK	12/12
RunOK	10/12
OutputOK	9/12
PhysicsModelOK	12/12
BoundaryOK	10/12
InitialOK	12/12
GeometryOK	10/12
MaterialOK	12/12
NumericsOK	11/12
ValidationOK	6/12

Table 4: Core evaluation summary across 12 base settings. ValidationOK indicates whether the recorded run produced validation-ready outputs; a failure may reflect a deliberate demo or smoke run rather than an intrinsically incorrect workflow.

5 Evaluation Analysis

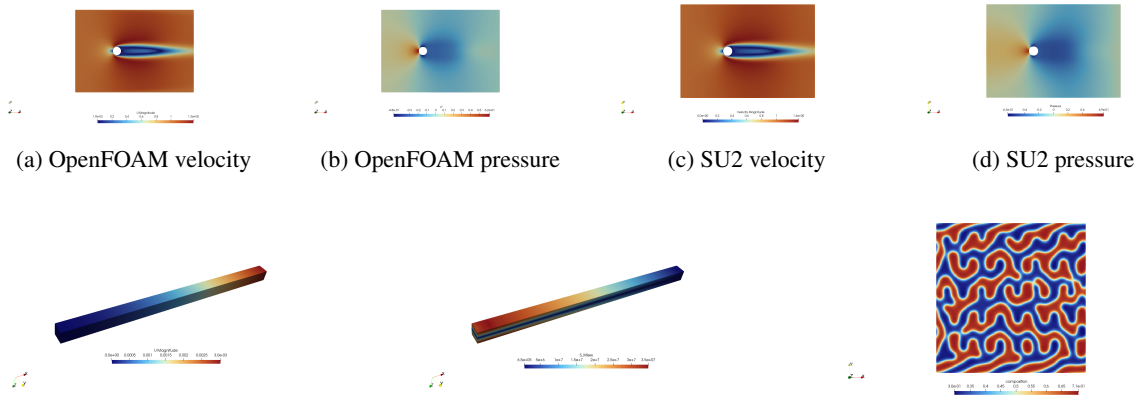
This section analyzes four aspects of the evaluation: workflow executability, physical faithfulness, sensitivity to missing physical information, and generation efficiency.

5.1 Input Effects on Workflow Executability

All three input forms support file-level workflow generation. Across the 12 base settings, FileComplete, SyntaxOK, and MeshOK are each 12/12 (Table 4), showing that the agent is strong at assembling workflow structure. The main executability differences appear at the run and output stages. Math-oriented Markdown obtains the highest aggregate executability score, 19/20, because all four math settings are recorded as RunOK and only OpenFOAM lacks expected outputs. Natural-language and tabular inputs each score 18/20, mainly because their SU2 settings only record dry/smoke runs and do not produce full flow-field or history outputs. The full workflow-executability matrix is provided in Appendix B.

5.2 Input Effects on Physical Faithfulness

Input representation has a stronger effect on physical faithfulness than on file-level construction. Table 5 summarizes the aggregate scores: natural language obtains 27/28, table input obtains 25/28, and math-oriented Markdown obtains 22/28. The difference is not because one representation is in-



(a) OpenFOAM velocity (b) OpenFOAM pressure (c) SU2 velocity (d) SU2 pressure

(e) CalculiX displacement (f) CalculiX stress (g) OpenPhase composition

Figure 2: Examples of generated workflow outputs across simulation backends, including cylinder-flow velocity and pressure fields, cantilever displacement and stress fields, and a spinodal-decomposition composition field. The visualizations are paired with log-based and metric-based evaluation rather than used as standalone correctness evidence.

Input	Exec.	Phys.	Validation
Natural language	18/20	27/28	3/4
Math Markdown	19/20	22/28	1/4
CSV/Table	18/20	25/28	2/4

Table 5: Aggregate scores by input representation. Executability aggregates five binary criteria across four software backends; physical faithfulness aggregates seven criteria. Validation is shown separately because it is the most stringent physical criterion.

trinsically superior. Rather, each representation foregrounds different information and leaves different gaps for the agent to fill. Math-heavy prompts make equations and parameters explicit, but they can under-specify software-facing details such as mesh scaling, marker conventions, solver controls, or validation targets. Tabular prompts make fields easy to parse, but they may fragment the context needed to understand how those fields interact. Natural language often preserves workflow intent and validation goals, but can be more verbose and less structured.

The general pattern is that physical faithfulness depends on whether the representation makes both *scientific constraints* and *software-realization constraints* explicit. Failures occur when a workflow is operational but a hidden assumption changes scale, discretization, resolution, or validation evidence. This is visible across the benchmark: mesh-scale drift changes effective dimensionless quantities, discretization choices can deviate from the requested numerical method, and a runnable structural case can still fail reference validation. The full physical-faithfulness matrix is provided in Appendix B.

5.3 Effect of Missing Physical Information

The completeness ablation tests how missing physical information affects accuracy and validation evidence. Across the seven ablation settings, natural-language specifications complete execution in 5/7 cases, while math and table specifications complete execution in 2/7 cases each. Validation-file availability is 5/7 for natural language, 2/7 for math, and 5/7 for table. The detailed ablation matrix is provided in Appendix C.

The main finding is that missing information changes the *failure mode*, not only the success rate. Missing governing equations or material values triggers physical reconstruction: the agent may infer a plausible model or parameter from task context, but the workflow becomes harder to audit because the recovered assumption is no longer explicitly grounded in the input. Missing boundary conditions triggers plausible completion: the generated workflow can remain executable and even pass a coarse boundary rubric, while still risking a silent change in the physical problem definition. Missing mesh information triggers numerical reconstruction: the agent may create or modify a mesh, but scale and resolution choices directly affect dimensionless quantities, near-wall behavior, and derived forces. Missing time controls primarily affects validation readiness: scripts may exist for longer execution, but the recorded run may not

reach a physical time horizon suitable for scientific checks.

These patterns generalize beyond the specific cylinder-flow case. They show that scientific simulation specifications contain several classes of information with different roles: model-defining information, constraint-defining information, discretization information, material/scale information, and validation-time information. LLM agents can often fill gaps in each class, but the resulting workflow should expose those assumptions rather than silently treating them as part of the original specification.

5.4 Input Effects on LLM-Side Efficiency

Input format also changes the cost of generating the workflow. The broad pattern is that cost reflects the amount of *workflow reconstruction* the agent performs, not simply the length or structure of the prompt. Structured input can be efficient when table fields map cleanly to software parameters, as shown by lower table-input costs in some backend summaries. However, structured input can also be costly when the agent must translate fragmented fields into many software-specific files, execution scripts, and checks. Natural language can preserve intent but may require more tokens to turn that intent into precise artifacts. Math-heavy input can be compact, but missing operational details may require additional tool use or repair.

The ablation efficiency results reinforce this interpretation. When governing equations are removed, cost increases for natural language by 47.5% and for math by 64.7%, indicating additional effort to reconstruct the physical model. When mesh information is removed, math cost increases by 53.6%, reflecting the cost of reconstructing numerical infrastructure rather than merely generating text. Conversely, some cheaper ablations correspond to incomplete execution or missing validation files, showing that lower cost may reflect less work rather than better automation. Efficiency is therefore best interpreted as a deployment-cost signal and a proxy for reconstruction burden, not as a correctness metric. The detailed efficiency summaries are provided in Appendix D.

6 Discussion

From code generation to workflow generation. The results in Section 4 show that the agent's output is best understood as a scientific workflow package rather than a code snippet. It creates solver inputs, mesh and preprocessing steps, execution scripts, logs, documentation, and post-processing artifacts. This shifts the evaluation problem: success is not only whether a file is syntactically correct, but whether the generated artifact chain preserves the scientific specification through execution and analysis. This is why file completeness, recorded execution, physical faithfulness, validation evidence, and efficiency must be evaluated separately.

Representation and completeness are control surfaces. The results in Section 5 suggest that reliability is shaped by two control surfaces: how the specification is represented and how complete its physical information is. Input format changes which constraints are explicit and which are left for the agent to infer. Completeness changes the type of inference the agent must perform: physical reconstruction, plausible boundary completion, numerical reconstruction, material reconstruction, or validation-time reconstruction. These are not merely prompt-engineering details. They determine whether the workflow remains tied to the intended scientific problem or drifts into a plausible but different simulation.

Toward assumption-aware scientific agents. The main design implication is that future LLM simulation agents should not silently fill gaps. When information is missing, the agent should expose assumptions, request clarification when possible, and attach checks to the generated workflow. Useful checks include unit and scale consistency, boundary-condition consistency, mesh and resolution checks, solver-log analysis, and validation-target extraction. Efficiency should also be reported as part of the workflow evidence, because high cost can indicate heavy reconstruction while unusually low cost can indicate incomplete execution or missing validation work. In short, trustworthy scientific agents should produce not only runnable workflows, but auditable workflows with explicit assumptions and validation hooks.

Workflow evidence as a first-class output. The benchmark also suggests that scientific agents should package evidence in a way that lets users distinguish generated intent from executed fact. For example, a run script, a solver log, a visualization, and a force-history file carry different evidential value. A generated `Allrun` script shows that the

agent knows how the workflow should be launched, but it does not prove that the full simulation was completed. A smoke-test log shows that the case can enter the solver, but it does not establish that the transient window is long enough for validation. A field image confirms that post-processing can access solver output, but it may not contain the quantitative observable required by the original task. For simulation agents, these distinctions should be explicit in the output package. The agent should label which artifacts were generated, which were executed, which were inspected, and which support validation. Such provenance is especially important when the interaction budget encourages short demonstrations while the generated workflow still includes scripts for full-length execution.

Implications for benchmark design. A benchmark for LLM-based scientific computing should therefore evaluate both construction and verification. Construction metrics ask whether the workflow contains the necessary directories, input decks, solver controls, scripts, and documentation. Verification metrics ask whether the workflow preserves physical scale, boundary meaning, material parameters, numerical intent, and validation targets. These two metric families should not be collapsed into a single success score. In our experiments, file completeness and mesh readiness are nearly saturated, while validation readiness remains much harder. This separation is useful because it identifies where an LLM agent is already helpful to users and where scientific risk remains. An engineer may benefit from a generated case skeleton even if a full validation run still requires manual inspection. At the same time, a paper or automated platform should not treat that skeleton as scientific evidence without solver logs and validation outputs. This layered view is the basis for the executability, physical-faithfulness, and efficiency analysis used in this work.

Practical use in simulation workflows. The results also clarify where LLM agents can be useful today. Their strongest role is not to replace expert judgment at the final validation stage, but to reduce the friction of moving from a specification to an inspectable first workflow. In practice, this includes creating solver-specific directory structures, translating physical parameters into configuration files, writing launch scripts, and preparing post-processing code that a user can then audit. This is valuable because many scientific-computing errors arise before the solver is ever run: missing files, inconsistent paths, incompatible mesh formats, or absent extraction scripts. At the same time, the user should remain responsible for accepting or rejecting inferred assumptions, choosing production run lengths, and interpreting validation quantities. The appropriate near-term framing is therefore human-in-the-loop workflow acceleration rather than autonomous scientific certification.

7 Limitations and Conclusion

Limitations. This study evaluates one LLM-agent setting and four representative open-source simulation environments. The benchmark focuses on classical cases rather than large-scale industrial simulations, multiphysics coupling, or closed commercial software. Some physical-faithfulness judgments require expert inspection, especially when deciding whether a simulation is validation-ready. The ablation study focuses on OpenFOAM cylinder flow and may not cover all physical domains. The local execution environment is also a single workstation, so wall-clock runtime should not be read as a general performance claim for the solvers themselves. Finally, several workflows record only smoke tests or demonstrations because the agent judged full execution to be costly in the interactive setting; we explicitly separate such cases from full scientific validation, and do not treat them as evidence that the generated workflow could not run to completion.

Conclusion. We studied LLM-mediated scientific simulation workflow generation across CFD, solid mechanics, and phase-field settings. Rather than treating the task as isolated code generation, we formulated it as the transformation of scientific specifications into executable, inspectable, and auditable workflow packages. Across OpenFOAM, SU2, CalculiX, and OpenPhase, the agent can often assemble complete workflows containing solver inputs, preprocessing steps, execution scripts, documentation, logs, post-processing routines, and visual outputs. At the same time, our results show that file completeness and short recorded execution do not by themselves guarantee physical faithfulness or validation-ready evidence. Input representation affects which scientific and software-realization constraints are explicit, while physical-information completeness determines how much the agent must infer or reconstruct. These factors jointly shape correctness, validation readiness,

and generation cost. The broader implication is that scientific LLM agents should be evaluated as assumption-bearing workflow systems: they should expose inferred assumptions, preserve links to the original specification, and attach validation checks to generated artifacts. Such specification-aware validation is essential before LLM-generated simulation workflows can be trusted as scientific computing tools. More broadly, the value of such agents is not limited to replacing expert users. They can lower the cost of assembling cross-software workflow skeletons, make simulation assumptions more visible, and provide reusable starting points for execution and post-processing. The remaining challenge is to bind that convenience to evidence: future systems should turn every inferred assumption, shortened run, generated script, and validation target into an inspectable part of the workflow record.

References

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

Iz Beltagy, Kyle Lo, and Arman Cohan. 2019. SciBERT: A pretrained language model for scientific text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pages 3615–3620.

Daniil A. Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. 2023. Autonomous chemical research with large language models. *Nature*, 624:570–578.

Andres M. Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D. White, and Philippe Schwaller. 2023. ChemCrow: Augmenting large-language models with chemistry tools. *arXiv preprint arXiv:2304.05376*.

Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q. Feldman, Arjun Guha, Michael Greenberg, and Abhinav Jangda. 2023. MultiPL-E: A scalable and extensible approach to benchmarking neural code generation. *IEEE Transactions on Software Engineering*, 49(7):3675–3691.

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, and 39 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021a. Measuring coding challenge competence with APPS. *Advances in Neural Information Processing Systems*, 34.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021b. Measuring mathematical problem solving with the MATH dataset. *arXiv preprint arXiv:2103.03874*.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. 2024. MLAgentBench: Evaluating language agents on machine learning experimentation. In *International Conference on Machine Learning*.

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

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2022. DS-1000: A natural and reliable benchmark for data science code generation. *arXiv preprint arXiv:2211.11501*.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. 2022. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, and 7 others. 2022. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, and 3 others. 2023. AgentBench: Evaluating LLMs as agents. *arXiv preprint arXiv:2308.03688*.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, and Nan Duan. 2021. CodeXGLUE: A machine learning benchmark

dataset for code understanding and generation. *arXiv preprint arXiv:2102.04664*.

Microsoft Research AI4Science and Microsoft Azure Quantum. 2023. The impact of large language models on scientific discovery: A preliminary study using GPT-4. *arXiv preprint arXiv:2311.07361*.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. CodeGen: An open large language model for code with multi-turn program synthesis. In *International Conference on Learning Representations*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. 2020. CodeBLEU: A method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297*.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. 2024. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, and 7 others. 2023. Code Llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. HuggingGPT: Solving AI tasks with ChatGPT and its friends in Hugging Face. *arXiv preprint arXiv:2303.17580*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Ross Taylor, Marcin Kardas, Guillem Cucurull, Thomas Scialom, Anthony Hartshorn, Elvis Saravia, Andrew Poulton, Viktor Kerkez, and Robert Stojnic. 2022. Galactica: A large language model for science. *arXiv preprint arXiv:2211.09085*.

Minyang Tian, Luyu Gao, Shizhuo Dylan Zhang, Xinan Chen, Cunwei Fan, Xuefei Guo, Roland Haas, Pan Ji, Kittithat Krongchon, Yao Li, Shengyan Liu, Di Luo, Yutao Ma, Hao Tong, Kha Trinh, Chenyu Tian, Zihan Wang, Bohao Wu, Yanyu Xiong, and 11 others. 2024. SciCode: A research coding benchmark curated by scientists. *Advances in Neural Information Processing Systems*, 37.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023a. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Xiaoxuan Wang, Ziniu Hu, Pan Lu, Yanqiao Zhu, Jieyu Zhang, Satyen Subramaniam, Arjun R. Loomba, Shichang Zhang, Yizhou Sun, and Wei Wang. 2023b. SciBench: Evaluating college-level scientific problem-solving abilities of large language models. *arXiv preprint arXiv:2307.10635*.

Frank F. Xu, Uri Alon, Graham Neubig, and Vincent J. Hellendoorn. 2022. A systematic evaluation of large language models of code. *arXiv preprint arXiv:2202.13169*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

A Representative Generated File Trees

The generated workflows are multi-file packages rather than single scripts. The examples below show representative file structures; complete generated artifacts are provided in the supplementary material.

```
OpenFOAM cylinder-flow workflow
  0/
    U
    p
  constant/
    transportProperties
    turbulenceProperties
    polyMesh/
  system/
    controlDict
    fvSchemes
    fvSolution
  Allrun or run.sh
```

```
  README.md
  postProcessing/

SU2 cylinder-flow workflow
  cylinder.cfg
  mesh.su2
  run.sh
  dry_run.log or solver logs
  history.csv / flow output when full run is
    recorded
  README.md

CalculiX cantilever-beam workflow
  cantilever_beam.inp
  run.sh or run_ccx.sh
  solver.log
  cantilever_beam.frd
  cantilever_beam.vtu
  postprocess.py
  summary.txt
  README.md

OpenPhase spinodal-decomposition workflow
  ProjectInput.opi
  Makefile
  run.sh
  run.log
  RawData/ or output/
  VTK/ or results/
  README.md
```

B Detailed Evaluation Tables

Tables A1 and A2 reproduce the two base-case evaluation matrices used for the aggregate results in Section 5. They follow the structure of `Workflow-Executability.xlsx` and `Physical-Faithfulness.xlsx`, respectively. Entries marked as 0 include the diagnostic reason used during scoring.

C OpenFOAM Physics-Information Ablation

Table A3 follows `Physics-Ablation.xlsx`. It reports the main execution, physics, mesh, solver, and validation fields used in Section 5.

D LLM-Side Efficiency Tables

Tables A4 and A5 correspond to the `Input_vs_Software_Summary` and `OpenFOAM_Ablation_Efficiency` sheets in `LLM-Efficiency.xlsx`.

E Input Representation Examples

This appendix provides representative excerpts from the OpenFOAM cylinder-flow prompts used in the base benchmark. The three examples specify the same intended physical task but differ in representation format. Full prompts for all benchmark settings will be released with the supplementary material.

E.1 Math-Oriented Markdown

```
# Two-Dimensional Incompressible Laminar
    Cylinder Flow

Consider the incompressible laminar flow of a
    Newtonian fluid
past a circular cylinder in two dimensions.

The governing equations are

\nabla \cdot \mathbf{u} = 0

\frac{\partial \mathbf{u}}{\partial t}
+ \mathbf{u}\cdot\nabla \mathbf{u}
= -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf
    {u}.

The Reynolds number is

Re = \frac{U_{in} D}{\nu},

with D = 1 m, U_in = 1 m/s, and
\nu = 0.01 m^2/s.

A mesh file is already available at:
rsc/mesh/cylinder2D.msh

Use Crank-Nicolson time discretization, second-
    order upwind
treatment for convective terms, central
    differencing for
diffusion terms, and a pressure-velocity
    coupling method such
as PISO or PIMPLE.

All generated contents should be written under:
experiments/FVM/Cylinder2D/Math/Case0/OpenFOAM
```

E.2 Natural-Language Text

```
Please create a complete and runnable two-
    dimensional
incompressible laminar flow simulation case
    within the current
software framework. The problem concerns uniform
     flow past a
circular cylinder in two dimensions. The fluid
    is Newtonian and
incompressible with constant density, and no
    energy equation is
required. The Reynolds number is 100, based on a
     cylinder
diameter of 1 m and an inlet velocity of 1 m/s,
    which
corresponds to a kinematic viscosity of 0.01 m
    ^2/s.

A mesh file has already been prepared in .msh
    format at
rsc/mesh/cylinder2D.msh, and the generated
    workflow should make
use of this mesh. The boundary conditions should
     represent a
uniform incoming flow from the inlet boundary
    with velocity
```

Input	Backend	FileComplete	SyntaxOK	MeshOK	RunOK	OutputOK	Diagnostic note
NL	OpenFOAM	1	1	1	1	1	Complete recorded run with expected output artifacts.
NL	SU2	1	1	1	0	0	Only dry-run/smoke logs were recorded; no full solver log, flow-field output, or history file.
NL	CalculiX	1	1	1	1	1	Solver output and post-processing artifacts were produced.
NL	OpenPhase	1	1	1	1	1	Phase-field output artifacts were produced.
Math	OpenFOAM	1	1	1	1	0	Only a short smoke run was recorded; no full post-processing directory or time-step output.
Math	SU2	1	1	1	1	1	Recorded run produced expected workflow outputs.
Math	CalculiX	1	1	1	1	1	Solver output and post-processing artifacts were produced.
Math	OpenPhase	1	1	1	1	1	Phase-field output artifacts were produced.
Table	OpenFOAM	1	1	1	1	1	Recorded run and post-processing artifacts were produced.
Table	SU2	1	1	1	0	0	Only dry-run logs were recorded; no full solver log, flow-field output, or history file.
Table	CalculiX	1	1	1	1	1	Solver output and post-processing artifacts were produced.
Table	OpenPhase	1	1	1	1	1	Phase-field output artifacts were produced.

Table A1: Workflow executability matrix following `Workflow-Executability.xlsx`. FileComplete checks required files, SyntaxOK checks syntactic usability, MeshOK checks mesh availability, RunOK checks recorded execution, and OutputOK checks expected output artifacts.

```
(1 0 0) m/s and zero-gradient pressure, while
    the outlet should
use zero-gradient velocity and fixed pressure p
    = 0.

The simulation should be run long enough, with
    the physical time
reaching at least t = 60 s. The time step should
     satisfy a
Courant number below 1. Please generate all
    required solver
settings, case files, execution scripts,
    commands, and
representative run logs.
```

E.3 Tabular CSV

```
Category,Item,Value
Software Framework,Framework,rsc/software_link/
    openfoam2412
Problem,Type,Two-dimensional incompressible
    laminar flow past a circular cylinder
Physics,Fluid,Newtonian incompressible fluid
Physics,Energy Equation,Disabled
Physics,Reynolds Number,100
Geometry,Cylinder Diameter,1 m
Geometry,Cylinder Location,Origin
Geometry,Domain Type,2D rectangular domain
Flow,Inlet Velocity,1 m/s
Material,Kinematic Viscosity,0.01 m^2/s
Mesh,Mesh File,rsc/mesh/cylinder2D.msh
Boundary,inlet,U=(1 0 0), p zeroGradient
Boundary,outlet,U zeroGradient, p=0
Boundary,top,U=(1 0 0), p zeroGradient
Boundary,bottom,U=(1 0 0), p zeroGradient
Boundary,cylinder,U=(0 0 0), p zeroGradient
Time,End Time,t >= 60 s
Time,Courant Number,Co < 1
Numerics,Time Discretization,Crank-Nicolson
Numerics,Convective Scheme,Second-order upwind
Numerics,Pressure-Velocity Coupling,PISO or
    PIMPLE
Output,Output Path,experiments/FVM/Cylinder2D/
    Table/Case0/OpenFOAM
```

F Artifact Licenses and Terms of Use

The artifacts associated with this study include input prompts, benchmark specifications, evaluation tables, generated workflow file trees, generated solver input files, execution scripts, logs, and post-processing outputs. Artifacts authored by the authors for this benchmark, including prompts, scoring tables, and generated workflow records, will be released with the supplementary material under a permissive research-use license. The released artifacts are intended to support inspection, reproduction, and non-commercial research use.

This artifact release does not relicense third-party software or services. OpenFOAM, SU2, CalculiX, OpenPhase, WSL Ubuntu, and other external tools remain governed by their respective licenses and terms of use. The LLM-agent runs were produced using Codex connected to GPT-5.5, and use of that service is subject to the provider's applicable terms. Users of the released artifacts are responsible for installing and using any required external solver, platform, or model service in accordance with its own license.

Backend	Model	BC	IC	Geometry	Material	Numerics	Validation	Note
OpenFOAM	1	1	1	1	1	1	1	validation-ready outputs
SU2	1	1	1	1	1	1	0	dry/smoke only; no Cd/Cl
CalculiX	1	1	1	1	1	1	1	displacement/stress outputs
OpenPhase	1	1	1	1	1	1	1	composition snapshots

(a) Natural-language input

Backend	Model	BC	IC	Geometry	Material	Numerics	Validation	Note
OpenFOAM	1	0	1	0	1	1	0	mesh scale/Re drift; smoke only
SU2	1	1	1	1	1	0	0	JST scheme; smoke only
CalculiX	1	1	1	1	1	1	0	tip displacement error $\sim$16%
OpenPhase	1	1	1	1	1	1	1	composition snapshots

(b) Math-oriented Markdown input

Backend	Model	BC	IC	Geometry	Material	Numerics	Validation	Note
OpenFOAM	1	0	1	0	1	1	0	mesh scale/Re drift; force not valid
SU2	1	1	1	1	1	1	0	dry/smoke only; no Cd/Cl
CalculiX	1	1	1	1	1	1	1	displacement/stress outputs
OpenPhase	1	1	1	1	1	1	1	composition snapshots

(c) Tabular CSV input

Table A2: Physical faithfulness matrix following `Physical-Faithfulness.xlsx`, split by input representation. Model, BC, IC, Geometry, Material, Numerics, and Validation correspond to PhysicsModelOK, BoundaryOK, InitialOK, GeometryOK, MaterialOK, NumericsOK, and ValidationOK. ValidationOK means that the recorded run produced validation-ready outputs; a 0 does not necessarily imply that the workflow could not run to completion.

Case	Ablated info	Exec.	ValFiles	Solver	Turb.	BC	IC	Cells	Mesh/source	Numerics	Val.	Note
A0	Full spec	1	1	pimpleFoam	laminar	1	U0=(1 0 0)	34764	supplied .msh; O-grid near wall	CN 0.9; linearUpwind	1	outputs available
A1	No equations	1	1	pimpleFoam	laminar	1	U0=(1 0 0)	34764	supplied .msh; O-grid near wall	CN 0.9; linearUpwind	1	inferred model
A2	No BCs	1	1	pimpleFoam	laminar	1	U0=(1 0 0)	34764	supplied .msh; O-grid near wall	CN 0.9; linearUpwind	1	BCs auto-filled
A3	No numerics	1	1	pimpleFoam	laminar	1	U0=(1 0 0)	34764	supplied .msh; O-grid near wall	CN 0.9; linearUpwind	1	controls selected
A4	No mesh	1	1	pimpleFoam	laminar	1	U0=(1 0 0)	12480	LLM blockMesh, coarse	CN 0.9; linearUpwind	1	mesh changed
A5	No equations/-material	0	0	pisoFoam	laminar	1	U0=(1 0 0)	34764	supplied .msh; O-grid near wall	CN 0.9; linearUpwind	0	no full log; material inferred
A6	No time controls	0	0	pimpleFoam	laminar	1	U0=(1 0 0)	34764	supplied .msh; O-grid near wall	CN 0.9; linearUpwind	0	no full log; time inferred

(a) Natural-language input

Case	Ablated info	Exec.	ValFiles	Solver	Turb.	BC	IC	Cells	Mesh/source	Numerics	Val.	Note
A0	Full spec	0	0	pimpleFoam	laminar	1	U0=(1 0 0)	34764	supplied .msh; O-grid near wall	CN 0.9; linearUpwind	0	no full log/files
A1	No equations	0	1	pimpleFoam	laminar	1	U0=(1 0 0)	34764	supplied .msh; O-grid near wall	CN 0.9; linearUpwind	1	files only; no full log
A2	No BCs	0	1	pimpleFoam	laminar	1	U0=(1 0 0)	34764	supplied .msh; O-grid near wall	CN 0.9; linearUpwind	1	BCs auto-filled
A3	No numerics	0	0	pimpleFoam	laminar	1	U0=(1 0 0)	34764	supplied .msh; O-grid near wall	Euler; linearUpwind	0	no full log/files
A4	No mesh	0	0	pimpleFoam	laminar	1	U0=(1 0 0)	95670	LLM blockMesh, refined	CN 0.9; linearUpwind	0	mesh changed; no files
A5	No equations/-material	1	0	pimpleFoam	laminar	1	U0=(1 0 0)	34764	supplied .msh; O-grid near wall	CN 1; linearUpwind	0	material inferred
A6	No time controls	1	0	pimpleFoam	laminar	1	U0=(1 0 0)	34764	supplied .msh; O-grid near wall	CN 0.9; linearUpwind	0	time inferred

(b) Math-oriented Markdown input

Case	Ablated info	Exec.	ValFiles	Solver	Turb.	BC	IC	Cells	Mesh/source	Numerics	Val.	Note
A0	Full spec	1	1	pimpleFoam	laminar	1	U0=(1 0 0)	34764	supplied .msh; O-grid near wall	CN 0.9; linearUpwindV	1	outputs available
A1	No equations	0	1	pimpleFoam	laminar	1	U0=(1 0 0)	34764	supplied .msh; O-grid near wall	CN 0.9; linearUpwind	1	files only; no full log
A2	No BCs	0	1	pimpleFoam	laminar	1	U0=(1 0 0)	34764	supplied .msh; O-grid near wall	CN 0.9; linearUpwind	1	BCs auto-filled
A3	No numerics	0	1	icoFoam	missing	1	U0=(1 0 0)	34764	supplied .msh; O-grid near wall	Euler; upwind	1	simpler solver/schemes
A4	No mesh	0	0	pimpleFoam	laminar	1	U0=(1 0 0)	34764	supplied .msh; O-grid near wall	CN 0.9; bounded linearUpwind	0	partial dirs; no files
A5	No equations/-material	0	0	pimpleFoam	laminar	1	U0=(1 0 0)	34764	supplied .msh; O-grid near wall	CN 0.9; linearUpwind	0	material inferred
A6	No time controls	1	1	pimpleFoam	laminar	1	U0=(1 0 0)	34764	supplied .msh; O-grid near wall	CN 0.9; linearUpwind	1	time inferred

(c) Tabular CSV input

Table A3: Physics-information ablation matrix for OpenFOAM cylinder flow, split by input representation. Exec. denotes complete recorded execution, and ValFiles/Val. denote whether validation files were produced in the recorded run. CN denotes Crank–Nicolson.

Input	Software	Cases	Billable	OutputTok	ReasonTok	Turns	Commands	Cache%	Files	Tok/File
Math	CalculiX	1	43207	6372	1057	1	12	76.8	13	3324
Math	OpenFOAM	7	107916	12059.7	3075.7	1	40.6	92.2	21.4	7960.4
Math	OpenPhase	1	84059	10620	1718	1	21	80.3	6	14010
Math	SU2	1	166802	14152	3071	1	53	93.1	7	23829
NL	OpenFOAM	6	142865.5	15447.2	3935.3	1	50.5	93.2	11.7	12331
NL	OpenPhase	1	82723	10740	1225	1	24	86.6	5	16545
NL	SU2	1	283358	21472	4947	1	71	93.6	5	56672
Table	CalculiX	1	52436	5326	448	1	17	78.3	12	4370
Table	OpenFOAM	7	150503.3	13116.4	2933.7	1	57.7	93.7	11.4	13175.4
Table	OpenPhase	1	64821	7809	808	1	25	87.3	7	9260
Table	SU2	1	80579	5378	1260	1	27	86.1	5	16116

Table A4: LLM-side efficiency aggregated by input representation and software backend. Billable tokens are fresh input tokens plus output tokens plus reasoning tokens.

Input	Case	Ablated info	Billable	Δ%	OutputTok	ReasonTok	Commands	Files	Tok/File	Cache%
NL	Case0	baseline	143772	0	22453	7043	66	12	11981	97.4
NL	Case1	governing equations	212072	47.5	17833	5783	63	11	19279	94.3
NL	Case2	boundary conditions	missing							
NL	Case3	numerical schemes	85614	-40.5	8240	1204	48	12	7134	91.4
NL	Case4	mesh information	133903	-6.9	18506	4728	42	12	11159	92.8
NL	Case5	equations + material	162467	13.0	15261	3404	42	13	12497	91.6
NL	Case6	time controls	119365	-17.0	10390	1450	42	10	11936	92.0
Math	Case0	baseline	94938	0	13235	2780	46	11	8631	95.0
Math	Case1	governing equations	156346	64.7	16049	3556	68	10	15635	94.9
Math	Case2	boundary conditions	63616	-33.0	10262	2868	32	10	6362	94.3
Math	Case3	numerical schemes	87447	-7.9	8972	2760	23	10	8745	86.0
Math	Case4	mesh information	145790	53.6	14068	3816	32	16	9112	92.9
Math	Case5	equations + material	141362	48.9	13262	4082	50	81	1745	94.1
Math	Case6	time controls	65913	-30.6	8570	1668	33	12	5493	88.5
Table	Case0	baseline	202767	0	19771	4676	99	13	15597	97.0
Table	Case1	governing equations	186358	-8.1	12298	2511	68	11	16942	94.1
Table	Case2	boundary conditions	89992	-55.6	12336	2954	40	11	8181	94.9
Table	Case3	numerical schemes	145153	-28.4	11938	3089	40	10	14515	90.9
Table	Case4	mesh information	161319	-20.4	15805	4180	67	11	14665	96.2
Table	Case5	equations + material	104031	-48.7	9289	1749	39	12	8669	92.1
Table	Case6	time controls	163903	-19.2	10378	1377	51	12	13659	90.7

Table A5: LLM-side efficiency for OpenFOAM physics-information ablations. The missing row corresponds to the unavailable natural-language Case2 efficiency log.