Evaluations Framework for Scientific Simulation Agents

Raunak Bardia, Shuvayan Bhramachary, Aniruddha Panda, Ankit Tyagi, Subodh Madhav Joshi, Kaushik Koneripalli, Kaushik Kalyanaraman

Shell India Markets Private Limited

Raunak.Bardia@shell.com

Abstract

Running multi-physics simulations typically demands significant expertise and manual effort. Agent-based workflows simplify this process by using natural user queries to automate key steps—such as parsing domain and property specifications, configuring mathematical properties, running auto-generated scripts, and reviewing outputs. Tools like OpenFOAM support these workflows owing to extensive documentation and open-source resources. However, agentic pipelines often fail due to error-prone intermediate steps, making systematic evaluation essential. This paper introduces a dual evaluation framework for simulation agents: : (a) rules-based approach that scores structural, syntactic, and keyword validity, and (b) an LLM-as-a-judge method that applies domain-aware rubrics to access physics coherence and cross-file consistency. The framework was applied to thirty-six OpenFoam test cases, each run five times to capture stochastic variability, resulting in 180 experiments. Results show that rules-based checks consistently assign higher scores by rewarding structural completeness, while LLM-based evaluation applies stricter criteria and reveals subtle physical inconsistencies. Prompt clarity strongly influences both evaluators, with expert prompts achieving the highest combined scores. These findings highlight that syntactic validity alone is insufficient for reliable simulation setup and advocate for multi-dimensional evaluation strategies to ensure trustworthy agentic workflows in scientific computing.

1. Introduction

Over the past four decades, simulations have been used to understand increasingly complex multi-physics problems, led by improvements in the numerical methods employed, and the increase in computational capacity. But it has also led to increasing complexity of these tools across all spatial-temporal scales. At present, applied computational scientists work with an abstracted layer of software, which is either a GUI or text files that interface with the simulation engine built in FORTRAN, C++ or other high-level languages. These interfaces require the user to be syntactically adherent and be familiar with nuances of each software, which make this interaction limiting. Only select engineers with expertise in simulation software are able to leverage these simulation tools [1] and that too with limited bandwidth for experimentation. The advent of generative AI solutions can be seen as a potential solution to the problem of making scientific software more accessible [2] by adding a layer of natural language interaction. This requires an LLM agent to convert the user query into a bash, text, or python script that in turn interfaces with the simulation engine.

One such use has been demonstrated for molecular simulations using LAMMPS, which uses a keyword-driven text file. GPT-4 was used to generate input files for simple test cases without additional database augmentation [2]. The authors tested the performance of this task agent by checking whether the generated input files got executed, by quantifying the stochasticity of the LLM output, and reporting an overall text-based similarity metric. Such simple pass/fail evaluation have also been reported for the MOOSE software agent [3] and the work of Wang, Zhang [4], where the authors use an LLM to directly generate a simulation engine from scratch in Python script.

More active development of simulator agents has been reported for OpenFOAM over just the past year [5-9]. Most agents, like MetaOpenFOAM [5], OpenFOAMGPT [8] and Foam-Agent [9], use a Retrieval Augmented Generation method for the multi-agent workflow, which references a pre-tokenized repository of cases to determine the most similar case to the user query, which is used as a basis for generation of simulation input files. The most comprehensive evaluation reported for OpenFOAM agents is reported in [10], which uses a ROUGE score to quantify the quality of text output compared to a ground truth, where the ROUGE score is only a lexical similarity metric.

The authors observe a gap in the agentic AI literature for scientific simulations in the lack of a comprehensive framework to quantify the outputs generated by an agent to execute a simulation. Often a simple lack of executability can be reported as a failed case but in practical use, the generated files may have minor errors that can be readily rectified by a trained user like the adoption of Copilot in higher-level programming. Due to the extent of progress in OpenFOAM agents, this work uses one of the published agents, Foam-agent [11], for analysis. To address the gap in evaluations, a comprehensive framework is defined in this work that is based on two pillars. First is a rules-based deterministic method, which uses structured knowledge of OpenFOAM keywords, and file structure to score the quality of agent outputs. Second is an LLM-as-a-judge method, which generates an LLM personality of OpenFOAM expertise and provide it the necessary guardrails to score different aspects of the generated agentic solution.

In section 2, we first report the architecture of the multi-agent workflow that is reported in the work of Yue, Somasekharan [11], which is evaluated for its performance using the quantification framework. Section 3 details the specifications of both the evaluation pillars, and section 0 presents the results from the evaluation framework when agent outputs are compared with ground truth solutions. Finally, section 5 summarizes the findings and identifies potential areas of investigation.

2. Multi-agent architecture of foam-agent

Foam-Agent is one of the OpenFOAM agents released in the past year and available through their GitHub repository: https://github.com/csml-rpi/Foam-Agent.git. This section briefly explains the architecture used by the agentic workflow that this paper evaluates, details of which are in [11].

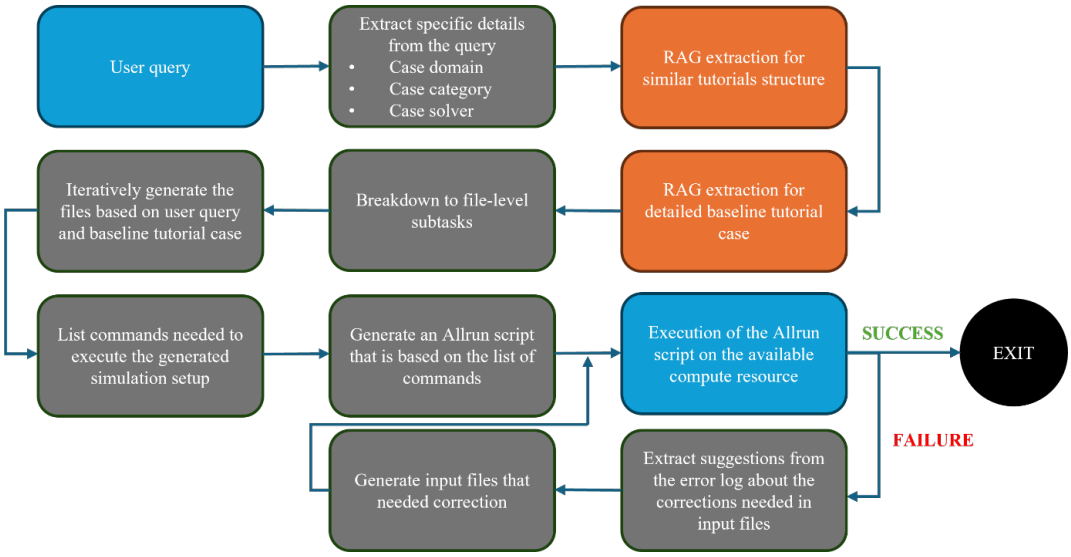
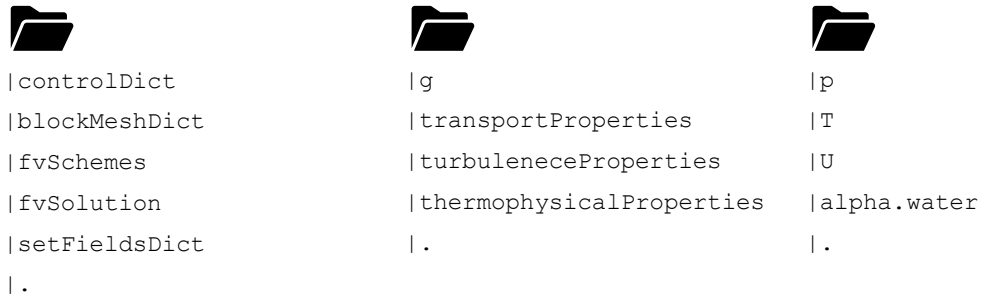


Figure 1: Workflow followed by Foam-Agent [11] for generation of OpenFOAM simulation cases

The workflow detailed in Figure *1* is constituted of four agents, namely Architect, Input writer, Runner, and Reviewer. Primary task of the architect agent is to ingest user query and determine the most similar case (using cosine similarity) from the OpenFOAM tutorials tokenized using LangChain. In [11], the authors use a hierarchical retrieval strategy where case information of the RAG database is level 1 of similarity check, and the case details of the RAG database are matched from the subset of level 1 retrieval. Hereon, we refer to the similar case as the baseline case. Following that determination, architect determines a breakdown of tasks into sub-tasks for each of the files that need to be generated. Input writer works on each sub-task to generate input files based on the specifications in user query and filling missing information using the baseline case. The final subtask is the generation of a script (Allrun) that creates a sequence of OpenFOAM commands to execute different utilities needed to execute a successful simulation. The Runner agent executes the Allrun script, which either ends up in a success or fails due to runtime error. In case of an error, a reviewer agent is invoked to interpret the error, suggest input file fixes, generate modified input files, and invoke the runner agent again.

3. Evaluations framework

This section provides a detailed overview of the two frameworks that are used to analyse the output of LLM-agents meant to generate inputs files for OpenFOAM simulation. OpenFOAM follows a rigid directory structure for the definition of a simulation setup as shown in Figure 2. In addition, each of the files in the directory must follow a specific naming convention, follow a dictionary structure, and use the keywords from a specific list of available options.




```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2406                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
version     2.0;
format      ascii;
class       dictionary;
object      turbulenceProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
simulationType      RAS;

RAS
{
RASModel        kEpsilon;

kEpsilonCoeffs
{
Prt 0.85;
}

turbulence      on;

printCoeffs     on;
}
```

Figure 2: (Left) Directory structure of a typical OpenFOAM simulation case (Right) Structure of a typical case file (turbulenceProperties file shown as an example)

3.1 Rules-based evaluation framework

For this deterministic approach to quantify agent's performance, at first, the dictionary format of OpenFOAM files is converted in the form of a JSON file for comparison. The keywords and type choices logged in JSON files were then compared using the following metrics:

1. **Lexical similarity –** It describes the use of relevant string comparison metrics often used in general LLM literature. The intent is to determine if the agent's output has minor deviations. In our implementation, we chose two specific scores for comparison:
    a. Levenshtein distance [12] – This determines the number of edits that agent output keyword would have to make to get the ground truth keyword

b. Rouge-1 score [13] – This tests the unigram overlaps and captures basic phrase level similarity. For instance, it tests if the choice of numerical scheme "Gauss linear corrected" matches with ground truth.

2. **Structural similarity –** Because OpenFOAM uses cascaded dictionary to specify keyword choices, and values for individual methods, it must follow a standardized tree structure. The generated file tree is node-wise compared with ground truth by evaluating a tree-edit distance [14].

3. **Semantic similarity –** Due to a strict naming convention that must be followed in OpenFOAM and the dependence between keywords within and across different files, a tool-specific scoring was built that could differentiate between a complete failure of the agent at understanding the nuances of OpenFOAM versus a mere difference in choice of keyword.

   This is achieved by curating a set of dictionaries that limits the type of method selection, which can be made once a particular keyword is chosen. These dictionaries are built for: boundaryFields, controlDict, fvSchemes, fvSolution, thermophysicalProperties, transportProperties, and turbulenceProperties. While more detailed specification of OpenFOAM keywords is possible, a first pass assessment with these dictionaries provides a baseline assessment of a correct OpenFOAM setup. Few dictionaries are shown in Figure .

Figure 3: (Left) Directory of boundaryFields comparing setup in blockMeshDict with type choices for each flow variable (Middle) Directory of finite volume numerical schemes that can be used for different mathematical operators in OpenFOAM (Right) Directory of keywords needed to define the turbulence properties

3.2 LLM-as-a-judge evaluation framework

While a programmatic check generated in the previous section is important, it can be limiting due to the complexity of OpenFOAM and other simulators. It is difficult to comprehensively check all possible settings and their internal associations. This problem is not exclusive to simulator agents discussed in this paper but in general as large language models have growing autonomy for complex tasks. Hence, many papers have proposed approaches to use reasoning ability of LLMs to assess the quality of outputs generated from other models. If its outputs are acceptable, this mode of evaluation has much larger scope of generalizability [15-17]. This is termed as "LLM-as-a-judge".

While methods have been proposed for using multiple such judges to either debate [18, 19] or vote [20] to get a final evaluation on an agent's performance, the method adopted in this work is a single LLM judge approach. This is done to avoid many unknowns introduced by either multiple judge-personalities or use of different LLMs for judges, each of which produces a stochastic output.

In this work, two "single judges" are deployed to assess the performance of the agent from two different perspectives:

1. Physics judge: This judge focuses on the physics that the user query intends to capture from the query. It specifically focuses on the similarity between the two setups and the correctness of the agentic solution instead of absolute quality.
2. Syntax judge: This judge is focused on the syntax, structure, and formatting correctness as it is defined as a code syntax expert for OpenFOAM

Both judges follow a scoring system as visualized in Figure 4. The reason for the skewed scoring is because in practice it was found that for the simple tasks that have been assessed in literature and this work, the agents do not make critical errors. Hence, a skewed scoring allows for the system to have a better breakdown when the agent performs at the basic level of being able to generate the files. The breakdown helps bring focus on the right and wrong file.

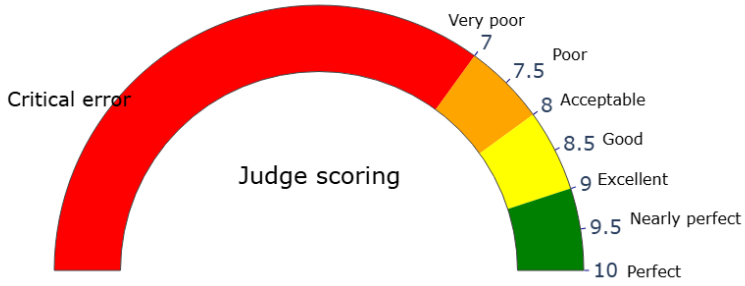


Figure 4: Scoring system used by LLM judges

4. Design of Experiment

In this study, eight representative OpenFOAM solvers were selected, from which 12 different test cases were chosen. These cases were carefully curated to capture variability across multiple dimensions: geometry and mesh (e.g., Cartesian grids, O-grid block meshes), flow physics (single-phase and multiphase), temporal characteristics (steady and transient), and flow regimes (incompressible and compressible). The solvers and their corresponding assessment cases are as follows:

- **icoFoam**: Transient solver for incompressible, laminar flow; assessed using the lid-driven cavity and elbow cases.
- **simpleFoam**: Steady-state solver for incompressible, turbulent flow; assessed using a backward facing step and Pitz Daily cases.
- **pisoFoam**: Transient solver for incompressible flow (laminar and turbulent); assessed using the cavity and porousBlockage cases.
- **pimpleFoam**: Transient solver combining PISO and SIMPLE iterations; assessed using contactAngleCavity, simulating a partially filled cavity with wall-interface contact angle effects.
- **interFoam**: Multiphase solver; assessed using cavity and elbow cases.
- **potentialFoam**: Potential flow solver; assessed using a cylinder case.
- **rhoCentralFoam**: Compressible flow solver; assessed using the forward step case.
- **sonicFoam**: Compressible, transonic solver; assessed using the shock tube case.

For each of the 12 OpenFoam test cases, three different user queries were generated to represent different tiers of complexity based on the user experience, i.e., average user prompt, intermediate user prompt and advanced user prompt. The collection of (12 × 3) test cases served as the ground truth data (i.e., simulations case files generated by an expert OpenFoam practitioner).

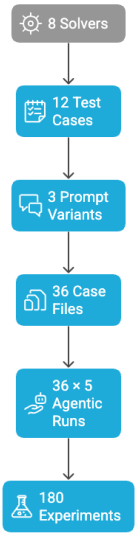


Figure 5. Schematic explaining the scheme behind the design of experiments

These 36 user prompts were now fed as the input prompt for the agentic runs. While these prompts were initially generated by a LLM (GPT-4o) with access to the ground truth case files, it required manual intervention from CFD experts in the case files generated by the LLM to ensure error free user prompts. This step was necessary to enable an error-free input prompt which reflect actual simulation settings in the ground truth case files. This stage ensures all ingredients to run agentic OpenFoam simulations. To address the stochasticity of the generated agentic output case files, each of the 36 test cases was repeated five times.

Results

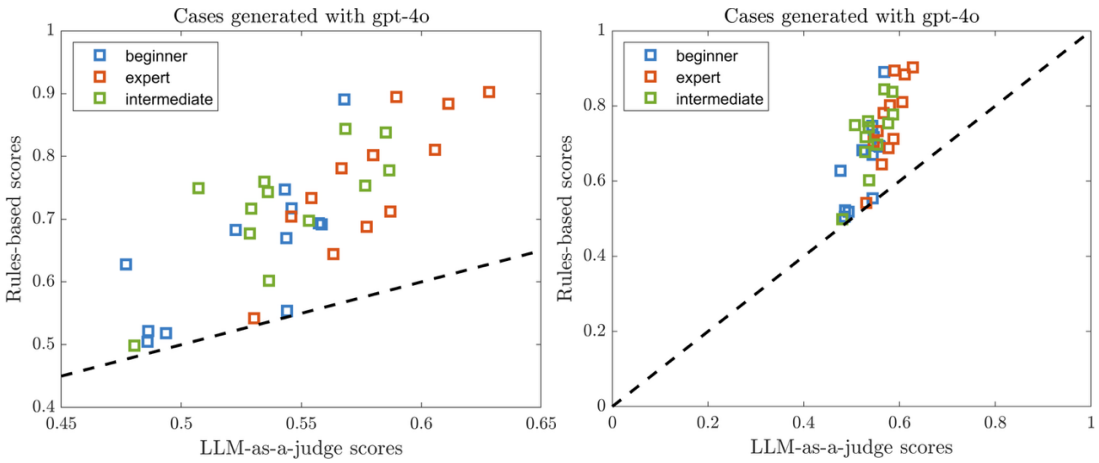


Figure 6: Mean scores obtained via LLM-as-a-judge method and rule-based evaluation, across all test cases (a) zoomed in view (b) axis normalised

In Fig. 6, each point represents a test case generated end-to-end using GPT-4o. The **x-axis** is the score assigned by the LLM-as-a-judge evaluator, and the **y-axis** is the score from the deterministic rules-based evaluator. The dashed line is the identity line (y = x), so points above it indicate that the rules-based score is higher than the LaaJ score for the same case. Several key observations can be made from this plot as follows:

(a) Rules-based evaluation consistently result in higher score across all test cases, compared to the scores obtained from LLM-as-a-judge. It rewards structural correctness—valid directory hierarchy, dictionary syntax, and keyword admissibility—even when the physics or cross-file consistency is imperfect.

(b) The figure reflects the conservative nature of the LLM-as-a-judge approach (≈0.45-0.63), which penalizes subtle inconsistencies such as mismatched solver settings, turbulence models, or boundary

conditions that violate physical plausibility. In contrast, rules-based scores span a wider range (≈0.50–0.90), indicating greater sensitivity to syntactic completeness.

(c) **Expert prompts** cluster toward the upper-right, achieving both higher LaaJ and rules-based scores, suggesting better alignment with physics and syntax. **Intermediate prompts** occupy the mid-region, while **beginner prompts** remain near the lower envelope, confirming that prompt clarity strongly influences case files quality and hence both evaluators.

(d) Cases with LaaJ ≈0.50 but rules ≈0.75–0.85 likely passed structural checks yet failed physics checks (e.g., inconsistent turbulence properties or solver choice for the stated regime).

The figure highlights that syntactic validity alone is insufficient for reliable simulation setup. While rules-based evaluation ensures executability, it cannot guarantee physical correctness. The LLM-as-a-judge adds this layer of semantic and domain-aware scrutiny, albeit offers limited variability across a range of test cases. Therefore, both evaluators are complementary: rules-based checks for structure, LaaJ for physics coherence.

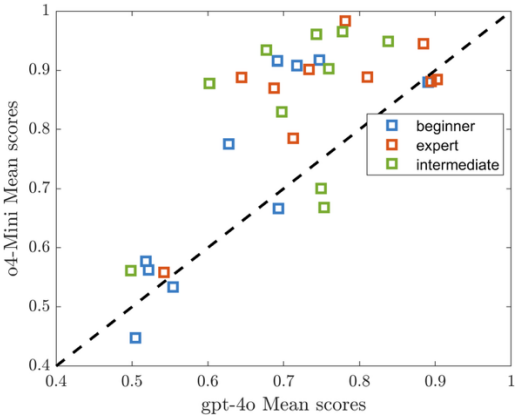


Figure 7: Rules-based evaluation scores using GPT4o and O4-mini

Figure 7 compares rule-based evaluation scores for two sets of case files: one generated by GPT-4o and the other by O4-Mini. Each point represents the same case–prompt pair evaluated under identical deterministic criteria. Most points lie above the diagonal, indicating that for many cases, the files produced by O4-Mini achieved higher structural and syntactic compliance than those from GPT-4o. However, the spread around the diagonal shows that this difference is not uniform—some points fall close to or below the line, suggesting comparable or slightly better performance by GPT-4o in a few instances.

In addition, prompt tier effects are visible: beginner prompts cluster toward the lower-left region, reflecting lower rule-based scores regardless of model, while intermediate and expert prompts dominate the upper region, where both models produce more compliant outputs. The overall distribution highlights that deterministic checks capture improvements in structural correctness when generation quality increases, but they do not fully explain physics validity or cross-file consistency, which require complementary evaluation.

5. Summary

This work addresses the challenge of evaluating agentic workflows for scientific simulations, focusing on OpenFOAM as a representative multi-physics solver. While recent LLM-driven agents simplify simulation setup through natural language queries, their outputs often exhibit structural and semantic inconsistencies that impact reliability. To quantify these issues, the paper introduces a dual evaluation framework: (i) a deterministic rules-based method that checks directory hierarchy, syntax, and keyword validity; and (ii) an LLM-as-a-judge approach that assesses physics coherence and cross-file consistency using domain-aware scoring rubrics.

The framework was applied to 12 curated OpenFOAM test cases spanning multiple solvers, flow regimes, and prompt tiers (beginner, intermediate, expert), with each case executed five times to capture stochastic variability. Results show that rules-based evaluation consistently assigns higher scores by rewarding syntactic completeness, while LLM-as-a-judge applies stricter criteria, penalizing subtle physical inconsistencies. Prompt clarity strongly influences both evaluators, with expert prompts achieving the highest combined scores. Additional analysis

comparing two generation models (GPT-4o and O4-Mini) under identical conditions reveals variability in structural compliance, reinforcing the need for repeated trials and multi-dimensional scoring.

Overall, the study demonstrates that syntactic validity alone is insufficient for reliable simulation setup. Combining deterministic checks with LLM-based semantic evaluation provides a more robust measure of agent performance. These findings highlight the importance of prompt engineering, dual evaluation strategies, and future work on physics-aware generation to advance trustworthy agentic workflows in scientific computing.

6. References

1. Doug, K., et al., *Accelerating Science and Engineering Discoveries Through Integrated Research Infrastructure for Experiment, Big Data, Modeling and Simulation: 22nd Smoky Mountains Computational Sciences and Engineering Conference, SMC 2022, Virtual Event, August 23–25, 2022, Revised Selected Papers*. 2023: Springer Nature.
2. Verduzco, J.C., E. Holbrook, and A. Strachan, *GPT-4 as an interface between researchers and computational software: improving usability and reproducibility.* arXiv preprint arXiv:2310.11458, 2023.
3. Zhang, T., et al., *MooseAgent: A LLM Based Multi-agent Framework for Automating Moose Simulation.* arXiv preprint arXiv:2504.08621, 2025.
4. Wang, L., L. Zhang, and G. He, *Evaluations of Large Language Models in Computational Fluid Dynamics: Leveraging, Learning and Creating Knowledge.* Theoretical and Applied Mechanics Letters, 2025: p. 100597.
5. Chen, Y., et al., *MetaOpenFOAM 2.0: Large Language Model Driven Chain of Thought for Automating CFD Simulation and Post-Processing.* arXiv preprint arXiv:2502.00498, 2025.
6. Dong, Z., Z. Lu, and Y. Yang, *Fine-tuning a large language model for automating computational fluid dynamics simulations.* Theoretical and Applied Mechanics Letters, 2025. **15**(3).
7. Fan, E., W. Wang, and T. Zhang, *ChatCFD: an End-to-End CFD Agent with Domain-specific Structured Thinking.* arXiv preprint arXiv:2506.02019, 2025.
8. Feng, J., R. Xu, and X. Chu, *OpenFOAMGPT 2.0: end-to-end, trustworthy automation for computational fluid dynamics.* arXiv preprint arXiv:2504.19338, 2025.
9. Yue, L., et al., *Foam-Agent 2.0: An End-to-End Composable Multi-Agent Framework for Automating CFD Simulation in OpenFOAM.* arXiv preprint arXiv:2509.18178, 2025.
10. Somasekharan, N., et al., *CFD-LLMBench: A Benchmark Suite for Evaluating Large Language Models in Computational Fluid Dynamics.* arXiv preprint arXiv:2509.20374, 2025.
11. Yue, L., et al., *Foam-agent: Towards automated intelligent cfd workflows.* arXiv preprint arXiv:2505.04997, 2025.
12. Lcvenshtcin, V. *Binary coors capable or 'correcting deletions, insertions, and reversals*. in *Soviet physics-doklady*. 1966.
13. Lin, C.-Y. *Rouge: A package for automatic evaluation of summaries*. in *Text summarization branches out*. 2004.
14. Zhang, K. and D. Shasha, *Simple fast algorithms for the editing distance between trees and related problems.* SIAM journal on computing, 1989. **18**(6): p. 1245-1262.
15. Zhu, L., X. Wang, and X. Wang, *Judgelm: Fine-tuned large language models are scalable judges.* arXiv preprint arXiv:2310.17631, 2023.
16. Li, D., et al., *From generation to judgment: Opportunities and challenges of llm-as-a-judge, 2025.* URL https://arxiv. org/abs/2411.16594, 2025.
17. Yu, F., *When AIs Judge AIs: The Rise of Agent-as-a-Judge Evaluation for LLMs.* arXiv preprint arXiv:2508.02994, 2025.
18. Bandi, C. and A. Harrasse, *Adversarial multi-agent evaluation of large language models through iterative debates.* arXiv preprint arXiv:2410.04663, 2024.
19. Verga, P., et al., *Replacing judges with juries: Evaluating llm generations with a panel of diverse models.* arXiv preprint arXiv:2404.18796, 2024.
20. Qian, Y., et al., *Enhancing LLM-as-a-judge via multi-agent collaboration.* 2025.