Foam-Agent: A Multi-Agent Framework for Automating OpenFOAM-based CFD Simulation

Ling Yue[*]
Rensselaer Polytechnic Institute
`yuel2@rpi.edu`

Nithin Somasekharan[*]
Rensselaer Polytechnic Institute
`somasn@rpi.edu`

Tingwen Zhang
Rensselaer Polytechnic Institute
`zhangt20@rpi.edu`

Yadi Cao
University of California San Diego
`yadicao95@gmail.com`

Shaowu Pan[†]
Rensselaer Polytechnic Institute
`pans2@rpi.edu`

Abstract

Agentic systems powered by large language models (LLMs) are increasingly reshaping scientific workflows by automating tasks that traditionally required expert intervention. In Computational Fluid Dynamics (CFD), the steep learning curve of simulation software and the complexity of multistage workflows present major barriers to accessibility and productivity. Hence, we introduce Foam-Agent, a multi-agent framework that automates the end-to-end OpenFOAM workflow from a single natural language prompt. Foam-Agent offers key innovations targeting realistic deployment scenarios through End-to-end workflow automation that spans 1) external mesh file import and text-to-mesh generation via the Gmsh library; 2) automatic HPC job submission and execution; and 3) post-simulation visualization, all implemented within a decomposable service architecture built on the Model Context Protocol (MCP) and high-fidelity generation through hierarchical retrieval across case metadata and dependency-aware file ordering, ensuring core files are produced first and supporting files follow in the correct sequence. On a benchmark of 110 CFD cases across 11 physics scenarios, Foam-Agent achieves an 88.2% execution success rate, surpassing existing frameworks by a large margin. By substantially lowering the expertise barrier for CFD, Foam-Agent demonstrates how specialized multi-agent systems can democratize complex scientific computing. The code is open-sourced at `https://github.com/csml-rpi/Foam-Agent`.

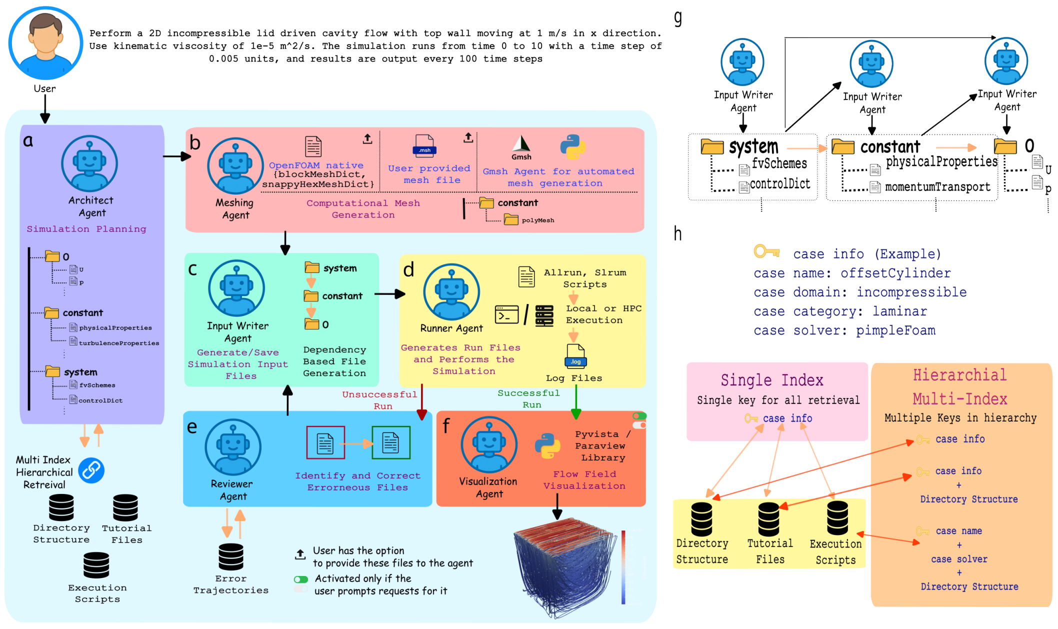
1 Introduction

Agents based on large language models (LLM) are rapidly transforming scientific workflows by automating complex, multistep tasks that traditionally required significant human expertise [15, 14]. In fields from biology [9, 18] to chemistry [4], agents are now indispensable collaborators, capable of planning experiments, invoking tools, and analyzing results [17]. This agentic paradigm is particularly promising for applied engineering, where sophisticated simulation software is critical but often challenging for users due to steep learning curves. In Computational Fluid Dynamics (CFD),

[*]Equal contribution.
[†]Corresponding author.

practitioners must manually define geometry, generate a high-quality mesh, specify complex physical models, initial and boundary conditions, and finetune dozens of solver parameters, a tedious and error-prone process where even minor mistakes can lead to simulation failure.

Recent work has begun to automate CFD with LLM-based systems [5, 11]. However, these approaches have been demonstrated on toy cases and face significant limitations in realistic deployments, e.g., a lack of robust geometry editing and meshing capabilities, manually designed rigid workflows, and a lack of demonstration coupling with high-performance computing (HPC) environments. To overcome these challenges, we introduce Foam-Agent, a multi-agent framework that automates the entire OpenFOAM [8] workflow from a single natural language prompt. Compared to existing work, our key contributions are rooted in the challenges of realistic deployments: (1) **Comprehensive Workflow Automation**: Foam-Agent is the first system to manage the full simulation pipeline, including advanced pre-processing with a versatile Meshing Agent capable of handling *external mesh files* and *providing text-to-mesh capability via the Gmsh library* to generate novel geometries and the associated mesh, automatic generation of HPC submission scripts, and post-simulation visualization. (2) **Decomposable Service Architecture**: Going beyond a monolithic agent, the framework uses Model Context Protocol (MCP) to expose its core functions as discrete, callable tools. This allows for flexible integration and use by other agentic systems, such as Claude-code, for more exploratory workflows. (3) **High-Fidelity Generation**: We employ a RAG based on a hierarchical index of case metadata (e.g., case name, domain, category, solver) to retrieve precise context. Generation is performed in a dependency-aware order, creating basic files first and then producing the dependent ones to ensure consistency.



Perform a 2D incompressible lid driven cavity flow with top wall moving at 1 m/s in x direction. Use kinematic viscosity of 1e-5 m^2/s. The simulation runs from time 0 to 10 with a time step of 0.005 units, and results are output every 100 time steps
User
a
Architect Agent
Simulation Planning
0
U
p
constant
physicalProperties
turbulenceProperties
system
fvSchemes
controlDict
Multi Index Hierarchical Retreival
Directory Structure
Tutorial Files
Execution Scripts
b
Meshing Agent
OpenFOAM native (blockMeshDict, snappyHexMeshDict)
User provided mesh file
Gmsh Agent for automated mesh generation
Computational Mesh Generation
constant
polyMesh
c
Input Writer Agent
Generate/Save Simulation Input Files
system
constant
0
Dependency Based File Generation
d
Runner Agent
Allrun, Slrum Scripts
Local or HPC Execution
Generates Run Files and Performs the Simulation
Log Files
Unsuccessful Run
Successful Run
e
Reviewer Agent
Identify and Correct Erroneous Files
Error Trajectories
User has the option to provide these files to the agent
Activated only if the user prompts requests for it
f
Visualization Agent
Pyvista / Paraview Library
Flow Field Visualization
g
Input Writer Agent
system
fvSchemes
controlDict
constant
physicalProperties
momentumTransport
0
U
p
h
case info (Example)
case name: offsetCylinder
case domain: incompressible
case category: laminar
case solver: pimpleFoam
Single Index
Single key for all retrieval
case info
Hierarchial Multi-Index
Multiple Keys in hierarchy
case info
case info + Directory Structure
case name + case solver + Directory Structure
Directory Structure
Tutorial Files
Execution Scripts

Figure 1: Foam-Agent system architecture illustrating the complete end-to-end workflow from natural language input to post-processing visualization of simulation.

2 Foam-Agent Framework

As displayed in Figure 1, Foam-Agent is a multi-agent system built on LangGraph [1], where each agent performs a specialized task within the CFD pipeline. The workflow allows for iterative refinement and error correction until a successful simulation is achieved. The core components are described below, with further details on the underlying architecture available in Appendix A.1.

Multi-Index Hierarchical Retrieval The Foam-Agent workflow begins by identifying suitable reference cases, even for novel scenarios, through a *hierarchical retrieval strategy*. A high-level case description (e.g., `name`, `domain`, `solver`) retrieves a semantically similar directory structure from a FAISS database, which then guides more detailed queries to other indices specifying file configurations and scripts. By prioritizing structural over literal similarity, this approach enables Foam-Agent to construct robust plans for unseen cases using analogous case architectures.

Architect Agent This agent translates natural language descriptions into structured simulation plans by classifying requirements with domain-specific taxonomies, retrieving semantically similar reference cases from hierarchical indices, and decomposing tasks into files and directories with defined dependencies and priorities. The resulting plan $P = \{F_1, F_2, ..., F_n\}$ specifies each required file along with its dependencies (e.g., boundary condition files in folder `0` depend on the turbulence model prescribed in files within `constant` folder), content, and generation order.

Meshing Agent This agent handles meshing in three modes: (1) *OpenFOAM native* via `blockMesh`/`snappyHexMesh`; (2) *Gmsh-based mesh generation*, where natural language descriptions generate Python scripts that utilize the Gmsh library to create geometries and associated meshes. This versatility overcomes the limitations of native meshing in OpenFOAM. (3) *Other meshes*, where user-provided pre-generated mesh files in third party software are converted to OpenFOAM format.

Input Writer Agent with Dependency-Aware Contextual Generation This agent generates OpenFOAM input files by adhering to a dependency-aware sequence: `system` (e.g., `controlDict`, `fvSolution`) $\rightarrow$ `constant` (e.g., `physicalProperties`) $\rightarrow$ `0` (e.g., `U`, `p`) $\rightarrow$ auxiliary files. This generation order mirrors the workflow of expert users, reducing setup errors such as inconsistent parameters. Dependencies are modeled as a directed acyclic graph, and cross-file consistency is enforced by providing previously generated files as context. `Pydantic` validation [2] further ensures syntactic and semantic correctness, minimizing setup errors.

Runner Agent This agent executes simulations locally or on High Performance Computing (HPC) systems. For HPC clusters, it generates and submits `Slurm` [16] scripts with inferred or user-provided parameters. It monitors jobs, cleans artifacts, and extracts structured error records from logs $E : L \rightarrow e_i$, where L is the execution error log and e_i refers to the ith structured error record, enabling robust error detection.

Reviewer Agent This agent iteratively analyzes the simulation log and corrects potential errors. It maintains a review trajectory $H = (F_i^j, E^j)$, where F_i^j denotes the state of the ith file at review iteration j and E^j is the corresponding structured error record (as defined in the Runner Agent) from that iteration. The agent then generates fixes ΔF consistent with user requirements C. This iterative refinement continues until success or a user-specified limit is reached.

Visualization Agent This agent generates requested visualizations using the `PyVista` [13] or `ParaView` [3] Python libraries. It interprets prompt-specified quantities, executes scripts, and iteratively corrects errors until the required plots (e.g., velocity fields) are produced.

Moving beyond a monolithic tool, Foam-Agent is built on the *Model Context Protocol* (MCP) [7], which decouples workflows into atomic functions orchestrated by higher-level agents. The design of these atomic functions is detailed in Appendix A.1. We realize this through a stateful LangGraph, with `Pydantic` ensuring structured I/O and LangSmith providing end-to-end traceability. These choices make Foam-Agent flexible, reliable, and fully observable.

3 Experiments and Results

We evaluated Foam-Agent on a comprehensive benchmark of 110 OpenFOAM cases covering 11 different physics scenarios [12]. We compare against MetaOpenFOAM [5], using both Claude 3.5 Sonnet. The comparison metric is the execution success rate, defined as the percentage of simulation cases that ran to completion given the natural language prompt describing the case.

Overall Performance As shown in Table 1, Foam-Agent significantly outperforms all baselines. With Claude 3.5 Sonnet, it achieves an 88.2% execution success rate, a substantial improvement over MetaOpenFOAM, which has 55.5%, with a visual comparison provided in Figure 2. The performance gap highlights the effectiveness of our advanced framework. Ablation studies, detailed in the Appendix A.2, confirm that the Reviewer Agent is the most critical component, contributing to

Table 1: Performance comparison and ablation study

Framework / Configuration	**Execution Success Rate (%)**
MetaOpenFOAM	55.5
Foam-Agent	**88.2**
- *without Reviewer Node*	57.3
- *without Reviewer & Hierarchical RAG*	44.6
- *without Reviewer & File Dependency*	45.4

an increase in performance of 30.9% alone. Further analysis reveals that even after its removal, the Hierarchical RAG and File Dependency modules still provide significant performance gains of 12.7% and 11.9%, respectively.

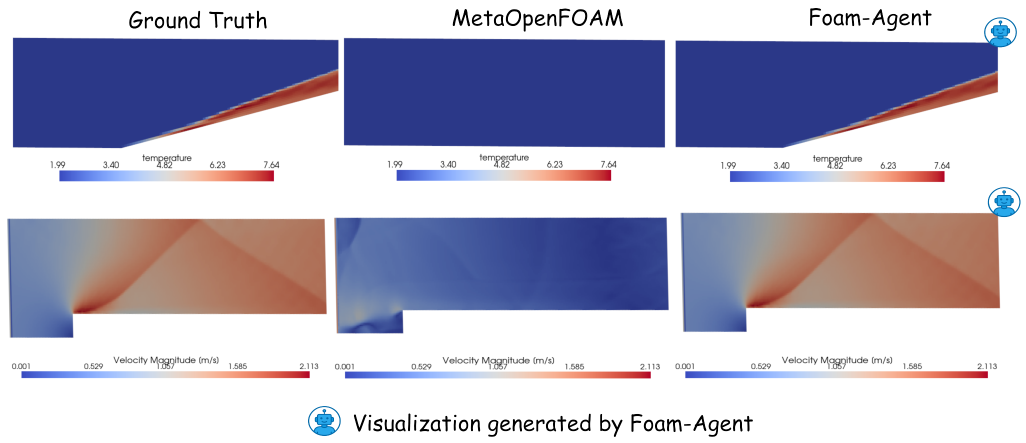


Figure 2: Comparison of simulation results produced by MetaOpenFOAM and Foam-Agent. Top: Temperature contour at a timestep of t=0.2s for `wedge` case. Bottom: Velocity magnitude contour at a timestep of t=4s for `forwardStep` case. Ground truth is produced by human OpenFOAM expert. The visuals for Foam-Agent are auto generated by the agent. Foam-Agent generates a Python script, which upon running, loads the case and produces the visual as a `.png` file. The same Python script is used to generate the visuals for ground truth and MetaOpenFOAM results.

Key Capabilities Showcase We demonstrate Foam-Agent's unique capabilities on two tasks that are beyond the scope of prior frameworks. Figure 3 shows the successful simulation of flow over a multi-element airfoil using a complex, externally provided mesh file. Figure 4 illustrates the agent's ability to generate a mesh for flow around a cylinder from a natural language description using the Gmsh library. These cases highlight the practical advancements delivered by Foam-Agent that are absent in prior frameworks, which primarily address tutorial or toy problems. Further, Figure 5 shows the atomization of each agentic function using the Model Context Protocol and exposes these functions to an orchestrator, which in this study is the debugging environment Cursor. The orchestrator performs a complete simulation of the flow over an NACA 0012 airfoil. It first generates the mesh using Gmsh by invoking `generate_mesh()`. Next, it creates the input files via `generate_file_content()` (the Input Write agent). Once the files are ready, it targets an HPC node by calling `generate_hpc_script()` to produce `Allrun` and Slurm scripts, and then submits the job with `run_simulation()`. Finally, it renders the velocity magnitude by invoking `generate_visualization()`, which triggers the Visualization node.

4 Conclusion

We presented Foam-Agent, a multi-agent framework that automates the full OpenFOAM workflow, from meshing and HPC execution to visualization, driven by a single natural language prompt. Foam-Agent achieves an 88.2% execution success rate across 110 benchmark cases, substantially

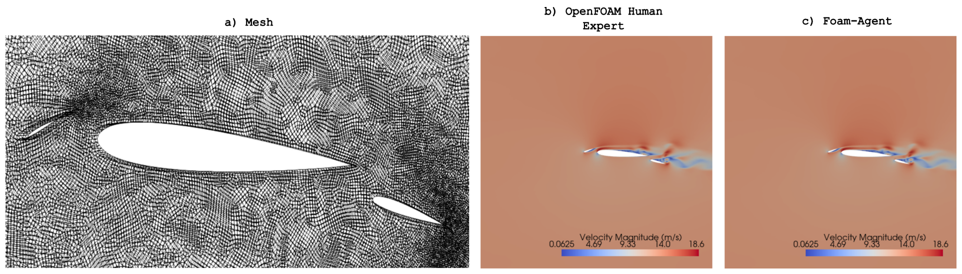


Figure 3: Flow over a multi-element airfoil by Foam-Agent. The mesh file is provided by the user and Foam-Agent creates the required files to run the same. The agentic results are indistinguishable in comparison to the simulation results by human OpenFOAM expert using the same mesh file.

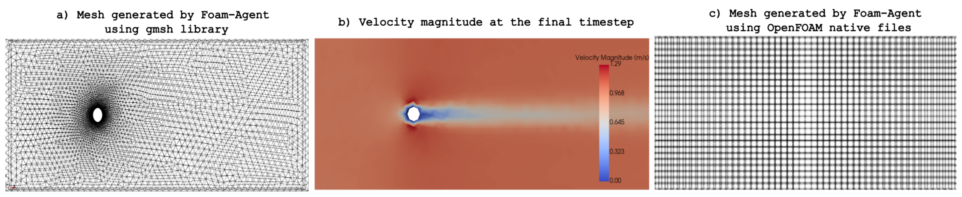
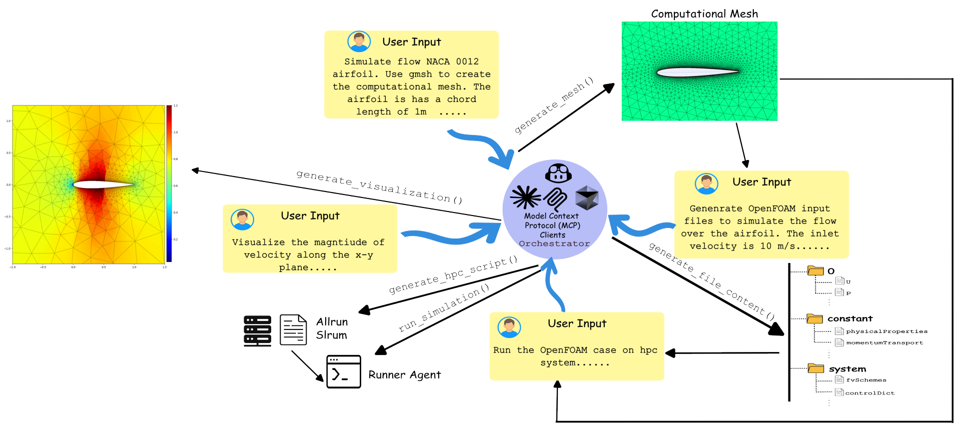


Figure 4: Mesh generated with the text to mesh functionality of Foam-Agent which utilizes the Gmsh meshing library for 2D flow over a cylinder. This capability is particularly useful since OpenFOAM's native meshing utilities cannot adequately capture the obstacle geometry in this case.



Simulate flow NACA 0012 airfoil. Use gmsh to create the computational mesh. The airfoil is has a chord length of 1m .....

Visualize the magntiude of velocity along the x-y plane.....

Genenrate OpenFOAM input files to simulate the flow over the airfoil. The inlet velocity is 10 m/s......

Run the OpenFOAM case on hpc system......

Figure 5: Simulation of flow over NACA 0012 airfoil using the MCP version of Foam-Agent. The user only provides the natural language request and it is the orchestrator that decides which function is to be called given the request.

outperforming prior systems. Unlike earlier approaches, Foam-Agent provides a modular architecture through the Model Context Protocol [10], enabling flexible integration with other agentic systems. It also supports external mesh import and text-to-mesh generation via the Gmsh library, capabilities that were not addressed in prior frameworks. By significantly lowering the expertise barrier for high-fidelity simulation, Foam-Agent democratizes access to powerful scientific computing tools and paves the way for broader adoption of agentic AI in engineering and applied sciences.

Acknowledgements

This work received funding support by the U.S. Department of Energy with ID DE-SC0025425. Shaowu Pan is supported by the Google Research Scholar Program. Computing resources are supported by the Lambda research grant program, NSF-ACCESS-PHY240112 and by the National Energy Research Scientific Computing Center under award NERSC DDR-ERCAP0030714.

References

[1] Langgraph documentation. https://langchain-ai.github.io/langgraph/. Accessed: August 2025.

[2] Pydantic documentation. https://docs.pydantic.dev/. Accessed: August 2025.

[3] Utkarsh Ayachit. The ParaView Guide: A Parallel Visualization Application. Kitware, Inc., Clifton Park, NY, USA, 2015.

[4] A. M. Bran et al. Chemcrow: augmenting large-language models with chemistry tools. Nature Machine Intelligence, 6(4):525–535, 2024.

[5] Yuxuan Chen, Xu Zhu, Hua Zhou, and Zhuyin Ren. Metaopenfoam: an llm-based multi-agent framework for cfd, 2024.

[6] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. 2024.

[7] Abul Ehtesham, Aditi Singh, Gaurav Kumar Gupta, and Saket Kumar. A survey of agent interoperability protocols: Model context protocol (mcp), agent communication protocol (acp), agent-to-agent protocol (a2a), and agent network protocol (anp). arXiv preprint arXiv:2505.02279, 2025.

[8] Hrvoje Jasak. Openfoam: Open source cfd in research and industry. International journal of naval architecture and ocean engineering, 1(2):89–94, 2009.

[9] John Jumper et al. Highly accurate protein structure prediction with alphafold. Nature, 596(7873):583–589, 2021.

[10] Chaoqian Ouyang, Ling Yue, Shimin Di, Libin Zheng, Shaowu Pan, and Min-Ling Zhang. Code2mcp: A multi-agent framework for automated transformation of code repositories into model context protocol services. arXiv preprint arXiv:2509.05941, 2025.

[11] S. Pandey, R. Xu, W. Wang, et al. Openfoamgpt: A retrieval-augmented large language model agent for openfoam-based computational fluid dynamics. Phys. Fluids, 37(3):037124, 2025.

[12] Nithin Somasekharan, Ling Yue, Yadi Cao, Weichao Li, Patrick Emami, Pochinapeddi Sai Bhargav, Anurag Acharya, Xingyu Xie, and Shaowu Pan. Cfd-llmbench: A benchmark suite for evaluating large language models in computational fluid dynamics, 2025.

[13] C. Bane Sullivan and Alexander A. Kaszynski. Pyvista: 3d plotting and mesh analysis through a streamlined interface for the visualization toolkit (vtk). Journal of Open Source Software, 4(37):1450, 2019.

[14] Tianyi Wu, Sherman Zhang, Atri Zheng, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. arXiv preprint arXiv:2308.08155, 2308(08155), 2023.

[15] Hui Yang, Sifu Yue, and Yunzhong He. Auto-GPT for online decision making: Benchmarks and additional opinions. arXiv preprint arXiv:2306.02224, 2023.

[16] Andy B. Yoo, Morris A. Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, Job Scheduling Strategies for Parallel Processing, pages 44–60, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[17] Ling Yue, Shimin Di, and Shaowu Pan. Autonomous scientific discovery through hierarchical ai scientist systems. Preprints, July, 2025.

[18] Ling Yue, Sixue Xing, Jintai Chen, and Tianfan Fu. Clinicalagent: Clinical trial multi-agent system with large language model-based reasoning. In Proceedings of the 15th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics, pages 1–10, 2024.

A Appendix

A.1 Detailed Methodology

Hierarchical Multi-Index RAG A key innovation in Foam-Agent is its hierarchical multi-index retrieval system that segments domain knowledge into specialized indices optimized for specific phases of the simulation workflow. This approach significantly improves retrieval precision compared to conventional single-index RAG systems. We construct the knowledge base by parsing OpenFOAM's tutorial cases and implement four distinct FAISS [6] indices: a `Tutorial Structure Index` for high-level case organization, a `Tutorial Details Index` for specific configuration parameters, an `Execution Scripts Index` for command sequences, and a `Command Documentation Index` for utility usage. This specialized architecture ensures that the agent receives highly relevant, context-specific information at each stage of the workflow.

Decoupling Capabilities via a Model Context Protocol (MCP) To transition Foam-Agent from a monolithic tool into a flexible scientific service, we design its core around the Model Context Protocol (MCP). This decouples the CFD workflow into atomic, callable functions exposed via a standardized protocol, making Foam-Agent a composable component that higher-level agents or workflow engines can orchestrate. The MCP design follows three principles: **atomicity** (each function does one task), **statefulness** (tracking multi-stage simulations via identifiers), and **workflow decoupling** (separating meshing, solving, and post-processing). Key functions are summarized in Table 2.

Table 2: The core functions of the Foam-Agent Model Context Protocol (MCP). Each function represents a decoupled capability within the CFD workflow, featuring strongly-typed inputs and outputs to ensure reliable interaction with orchestrating agents.

Function Name	**Description**	**Input Schema**	**Output Schema**
`create_case`	Initializes a new CFD simulation case and its workspace.	`{user_prompt: str}`	`{case_id: str}`
`plan_simulation_structure`	(Architect Agent) Plans the required file and directory structure based on the user prompt.	`{case_id: str}`	`{plan: List[{file, folder}]}`
`generate_file_content`	(Input Writer Agent) Generates the content for a single specified configuration file.	`{case_id, file, folder}`	`{content: str}`
`generate_mesh`	(Meshing Agent) Asynchronously generates the computational mesh using a specified method.	`{case_id, mesh_config: Dict}`	`{job_id: str}`
`generate_hpc_script`	(HPC Agent) Generates a job submission script (e.g., Slurm) for a high-performance computing cluster.	`{case_id, hpc_config: Dict}`	`{script_content: str}`
`run_simulation`	(Runner Agent) Asynchronously executes the simulation either locally or by submitting to an HPC cluster.	`{case_id, environment: str}`	`{job_id: str}`
`check_job_status`	Checks the status of any asynchronous job (meshing, simulation, visualization).	`{job_id: str}`	`{status: Dict}`
`get_simulation_logs`	Retrieves detailed logs for a failed job to enable error diagnosis.	`{case_id, job_id}`	`{logs: Dict}`
`review_and_suggest_fix`	(Reviewer Agent) Analyzes error logs and proposes corrective actions.	`{case_id, logs}`	`{suggestions: Dict}`
`apply_fix`	Applies suggested modifications to the relevant case files.	`{case_id, modifications: List}`	`{status: str}`
`generate_visualization`	(Visualization Agent) Asynchronously generates a visualization of the simulation results.	`{case_id, quantity, ...}`	`{job_id: str}`

System Orchestration with LangGraph and LangSmith The sequence of MCP function calls is dynamically determined by an intelligent orchestrator implemented as a stateful graph using LangGraph. Nodes in the graph correspond to MCP function calls, while conditional edges direct the workflow based on intermediate outcomes. This is particularly effective for the iterative refinement loop: a simulation failure routes the workflow to a debugging subgraph before re-attempting the run. To ensure reliability, all data exchanges use strict Pydantic schemas for validation. For observability, we integrate LangSmith to log every function call, LLM invocation, and state transition, creating an immutable record of the agent's actions for debugging and analysis.

A.2 Ablation Studies

We conducted a series of ablation studies to quantify the contribution of each key component in our framework. We analyzed the impact of the **Reviewer Node**, the **File Dependency** analysis module, the generation **Temperature**, and the **Hierarchical RAG**. All experiments were performed using the Claude-Sonnet-3.5 model. The results are summarized in Tables 3 and 4.

The inclusion of the **Reviewer Node** is the most significant factor, dramatically improving the execution success rate from roughly 50% to over 80%. This highlights the critical role of iterative self-correction. The **File Dependency** analysis improves execution success rate in the absence of the reviewer and reduces the number of review loops required for convergence when the reviewer is present, thus improving efficiency. The **Hierarchical Multi-Index RAG** also provides a significant boost in performance (57.3% vs. 44.6% without the reviewer), demonstrating the value of specialized, context-aware retrieval.

Table 3: Ablation study results evaluating the impact of the reviewer node and file dependency analysis on execution success rate, token usage, and reviewer efficiency. We varied the temperature to show that Foam-Agent's performance is generalizable. The highest execution success rate for each configuration pair is highlighted in bold.

Reviewer Node	**File Dependency**	**Temp-erature**	**Execution Success Rate (%)**	**Token Usage**	**Avg. Reviewer Loops**
			Configuration without Reviewer Node		
✗	✗	0.0	48.2	282,056	N/A
✗	✓	0.0	**56.4**	314,670	N/A
✗	✗	0.6	45.4	282,034	N/A
✗	✓	0.6	**57.3**	314,922	N/A
			Configuration with Reviewer Node (max_loops=10)		
✓	✗	0.0	86.4	309,873	0.90
✓	✓	0.0	**88.2**	332,324	0.79
✓	✗	0.6	86.4	356,272	1.87
✓	✓	0.6	**88.2**	334,303	0.96

Table 4: Comparison of Foam-Agent's performance using hierarchical multi-index retrieval vs. a single-index (baseline) retrieval. All experiments are performed using Claude-Sonnet-3.5 at T=0.6. The highest execution success rates are highlighted in bold.

Retrieval Method	**Reviewer Node**	**Execution Success Rate (%)**	**Token Usage**	**Avg. Reviewer Loops**
		Configuration without Reviewer Node		
baseline	✗	44.6	271,136	N/A
hierarchy	✗	**57.3**	306,844	N/A
		Configuration with Reviewer Node (max_loops=10)		
baseline	✓	84.6	323,011	0.73
hierarchy	✓	**88.2**	334,303	0.96

A.3 Additional Case Studies

To further demonstrate Foam-Agent's accuracy, we present comparative visualizations for two additional physics scenarios. Figure 6 shows that Foam-Agent accurately reproduces the sharp concentration gradient of a flame front in a counterflow flame simulation, whereas MetaOpenFOAM produces a diffuse, inaccurate result. Similarly, in Figure 7, Foam-Agent correctly captures the complex circular wave dynamics in a shallow water simulation, a task where MetaOpenFOAM fails completely.

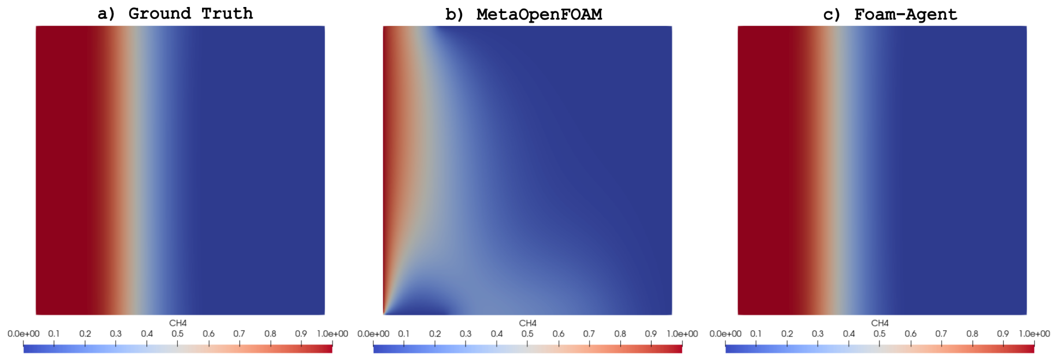


Figure 6: CH_4 mass fraction distribution comparison in counterflow flame simulations at t=0.5s: Ground Truth (left), MetaOpenFOAM (center), and Foam-Agent (right), demonstrating Foam-Agent's superior reproduction of concentration gradients.

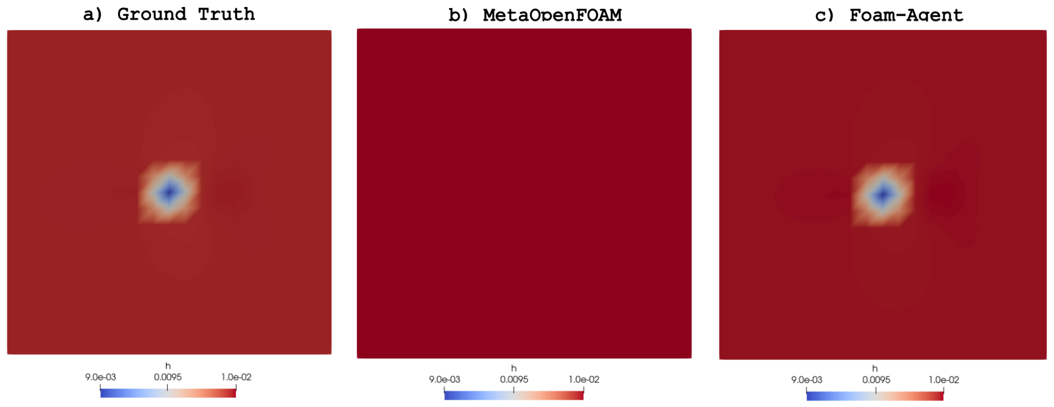


Figure 7: Comparison of free surface height distribution in shallow water equation simulations: Ground Truth (left), MetaOpenFOAM (center), and Foam-Agent (right), highlighting Foam-Agent's ability to accurately reproduce complex wave dynamics.

A.4 System and User Prompts

This section presents a selection of the system and user prompts used to guide the agents within the Foam-Agent framework.

A.4.1 Architect Agent: Task Decomposition

Task Decomposition System Prompt

You are an experienced Planner specializing in OpenFOAM projects. Your task is to break down the following user requirement into a series of smaller, manageable subtasks. For each subtask, identify the file name of the OpenFOAM input file (foamfile) and the corresponding folder name where it should be stored. Your final output must strictly follow the JSON schema below and include no additional keys or information:
{ "subtasks": [{ "file_name": "<string>", "folder_name": "<string>" }] }
Make sure that your output is valid JSON and strictly adheres to the provided schema.

A.4.2 Input Writer Agent: File Generation

File Generation System Prompt

You are an expert in OpenFOAM simulation and numerical modeling. Your task is to generate a complete and functional file named: <file_name>{file_name}</file_name> within the <folder_name>{folder_name}</folder_name> directory. Ensure all required values are present and match with the files content already generated. Before finalizing the output, ensure: - All necessary fields exist (e.g., if ‘nu‘ is defined in ‘constant/transportProperties‘, it must be used correctly in ‘0/U‘). - Cross-check field names between different files to avoid mismatches. - Ensure units and dimensions are correct for all physical variables. Provide only the code—no explanations, comments, or additional text.

A.4.3 Reviewer Agent: Error Analysis

Error Analysis System Prompt

You are an expert in OpenFOAM simulation and numerical modeling. Your task is to review the provided error logs and diagnose the underlying issues. You will be provided with a similar case reference, which is a list of similar cases that are ordered by similarity. You can use this reference to help you understand the user requirement and the error. When an error indicates that a specific keyword is undefined (for example, ’div(phi,(p|rho)) is undefined’), your response must propose a solution that simply defines that exact keyword as shown in the error log. Do not reinterpret or modify the keyword (e.g., do not treat ’|’ as ’or’); instead, assume it is meant to be taken literally. Propose ideas on how to resolve the errors, but do not modify any files directly. Please do not propose solutions that require modifying any parameters declared in the user requirement, try other approaches instead. Do not ask the user any questions. The user will supply all relevant foam files along with the error logs, and within the logs, you will find both the error content and the corresponding error command indicated by the log file name.

A.4.4 Example User Requirement Prompts

Multi-Element Airfoil (External Mesh)

do an incompressible 2D incompressible flow over a multi element airfoil setup. The mesh is provided as a .msh file. The msh file contains 4 boundaries named "inlet", "outlet", "walls", "airfoil" and "frontAndBack". The "inlet" and "outlet" are of type freestream with the freestream velocity being 9 m/s. The "walls" and "airfoil" have a no-slip boundary condition. The "frontAndBack" faces are designated as ’empty’. The simulation runs from time 0 to 10 with a time step of 1.0 units. The viscosity (nu) is set as constant with a value of 1.5×10^{-5} m^2/s. Use simpleFoam solver. Use SpalartAllmaras turbulence model. Further visualize the magnitude of velocity along the Z plane.

Flow Over Cylinder (Gmsh Generation)

Simulate incompressible flow over a circular cylinder. Use gmsh to create the computational mesh. The computational domain extends from -2.5 to 2.5 in the x-direction, -1 to 1 in the y-direction. The cylinder is positioned at (-1, 0) with a radius of 0.1 units. The inlet boundary named "inlet" has a uniform velocity of 1 m/s. The right boundary is the outlet named "outlet". The top and bottom walls named "topWall" and "bottomWall" use slip boundary conditions. The cylinder surface named "cylinder" uses a no-slip boundary condition. Use base mesh size of 0.5 on cylinder and size of 1.0 elsewhere. The simulation runs from time 0 to 2 seconds with a time step of 0.001 units. The kinematic viscosity (nu) is set as constant with a value of $1 \times 10^{-5}\ m^2/s$. Use pisoFoam solver. Visualize the magnitude of velocity ('U') along the x-y plane.

3D Cavity (HPC Case)

Do an incompressible 3D lid driven cavity flow using icoFoam solver. The cavity is a cube of dimension [0, 0.1]x [0, 0.1]x [0,0.1]. Use simple grading with 100X100x100 in x, y and z direction. The top wall ('movingWall') moves in the x-direction with a uniform velocity of 1 m/s. The 'fixedWalls' have a no-slip boundary condition. The simulation runs from time 0 to 0.015 with a time step of 0.001 units. The viscosity (nu) is set as constant with a value of 0.01 m^2/s. Perform an hpc run for this case in perlmutter cluster. My account is xxxx. Do a parallel run for this case by splitting it into 32 subdomains.