

A Dataset Curation

A.1 CFDQuery

This Question and Answer dataset spans a broad spectrum of PDEs, numerical methods and error-analysis topics. It delves into classical finite difference and finite volume schemes applied to 1D advection, 1D diffusion, 1D Burgers equation, etc (e.g. modified-equation analyses of central-difference+RK3, Lax–Friedrichs dissipation, upwind bias) and proceeding through high-order stencils (fourth-order central, compact schemes, WENO) and their dispersion/dissipation properties. It also involves questions based on non-uniform and curvilinear grids—deriving coefficient formulas for second derivatives on unequal spacings, analyzing truncation errors on non-orthogonal meshes, and enforcing the geometric-conservation law. Further the questions probe multi-dimensional flows (Poisson, Navier–Stokes channel and cavity, Rayleigh–Bénard, KdV–Burgers) with questions on stability criteria, and leading-order error terms. Finally, the set includes advanced topics in high-order discontinuous-Galerkin.

These questions are curated, reviewed and solved by human CFD experts before adding to the dataset. The sources of the dataset include textbooks and online sources. Each question is self-sufficient and provides four options to the LLM to select the right answer from. In addition we also provide a system prompt to the LLM to assign the role it will be playing when answering the questions. The system prompt is given below.

Prompt

You are an expert computational fluid dynamics researcher. For each multiple-choice question, read the question and its four options, then respond with only the number (1, 2, 3, or 4) corresponding to the correct answer.

A.2 CFDCodeBench

To construct *CFDCodeBench*, we curated a dataset of 24 computational fluid dynamics (CFD) problems from publicly available GitHub repositories and established numerical solver packages. Foundational problems were selected from the widely used *CFD Python: the 12 Steps to Navier-Stokes Equations* repository [4] and other educational sources such as *ENGR 491 - Computational Fluid Dynamics* [26]. These 17 problems, which include well-documented tutorials and reference code. To introduce more challenging scenarios, we incorporated 7 advanced problems from the Dedalus Project [11], which offers flexible PDE solvers based on spectral methods. These problems lacked detailed tutorials, so CFD experts reviewed the source code and authored accompanying descriptions. All problem descriptions and corresponding solutions were manually validated to ensure correctness and consistency.

- **1D Burgers Equation:** Simulates 1D viscous Burgers equation with Dirichlet boundaries.
- **1D Diffusion Equation:** Models scalar diffusion over time with piecewise constant initial conditions in a 1D domain.
- **Euler’s equation for compressible flow in a shock tube:** Simulates shock propagation in a 1D shock tube using the Euler equations with reflective boundaries. The solution to this equation is highly susceptible to numerical instabilities.
- **1D linear convection Equation:** Solves undamped linear convection of a Gaussian wave with periodic boundaries.
- **1D non-linear convection Equation:** Captures nonlinear wave propagation with sinusoidal initial conditions and periodic boundaries.
- **2D Burgers Equation:** Simulates viscous flow in both x and y directions using the 2D Burgers’ equation with Dirichlet boundaries.
- **2D Convection Equation:** Models 2D inviscid convection of a velocity disturbance with constant boundary conditions.
- **2D Diffusion Equation:** Solves a 2D scalar diffusion problem with fixed values on all boundaries and an initial high-temperature patch.

- 896 • **2D inviscid Burgers Equation:** Captures shock formation in a 2D inviscid Burgers' flow
897 using a square domain and periodic boundaries.
- 898 • **2D Laplace Equation:** Solves a steady-state potential problem with mixed Dirichlet and
899 Neumann conditions.
- 900 • **2D Linear Convection Equation:** Simulates scalar convection in two directions from a
901 localized initial disturbance.
- 902 • **2D Navier-Stokes equation in a cavity:** Computes incompressible viscous flow in a
903 lid-driven cavity setup using the Navier-Stokes equations.
- 904 • **Channel Flow with Navier-Stokes:** Solves channel flow with periodic inlet/outlet, no-slip
905 top/bottom, and constant body force.
- 906 • **2D Poisson Equation:** Solves the 2D Poisson equation with localized sources and Dirichlet
907 boundaries.
- 908 • **2D Steady Heat Equation:** Models steady-state heat conduction on a rectangular plate with
909 fixed temperatures on all edges.
- 910 • **2D Unsteady Heat Equation:** Simulates time-dependent heating with a Gaussian source
911 term and fixed boundaries.
- 912 • **Fully-developed turbulent flow in a channel:** Uses a Cess turbulence model to compute
913 velocity profiles in a turbulent channel with effective viscosity.
- 914 • **1D Korteweg-de Vries / Burgers Equation:** Models the combined effects of diffusion and
915 dispersion in wave dynamics using the KdV-Burgers equation.
- 916 • **2D horizontally-periodic Rayleigh-Benard convection Equation:** Simulates buoyancy-
917 driven convection with temperature gradients and periodic lateral boundaries.
- 918 • **2D periodic incompressible shear flow with a passive tracer field:** Models shear flow
919 evolution and passive tracer transport with periodic boundaries.
- 920 • **Flow past circular cylinder:** Simulates vortex shedding behind a cylinder using
921 streamfunction-vorticity formulation in polar coordinates.
- 922 • **Lane-Emden Equation:** Solves a spherically symmetric nonlinear Poisson equation used
923 in astrophysics.
- 924 • **Incompressible Navier Stokes equations in a lid-driven cavity:** Captures recirculating
925 flow in a square cavity driven by a moving top wall.
- 926 • **Linear stability eigenvalue problem for pipe flow:** Solves the linear stability eigenvalue
927 problem of pipe flow using the linearized Navier-Stokes equations.

928 A.2.1 Prompt Design and Methodology for CFDCoDeBench

929 **Structured Prompt Generation** We adopt a structured, JSON-to-natural language pipeline for
930 prompt generation. Each PDE problem is described in a JSON object with fields such as:

- 931 • **equation:** The governing PDE, formatted using \LaTeX , e.g., $\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}$.
- 932 • **boundary conditions:** A description of boundary behavior, written in either \LaTeX or
933 plain text, e.g., periodic boundary conditions such as $u(0) = u(2\pi)$.
- 934 • **initial conditions:** The initial state of the solution field, typically in compact \LaTeX
935 format, e.g., $u(x, 0) = \begin{cases} 2, & \text{if } 0.5 \leq x \leq 1 \\ 1, & \text{otherwise} \end{cases}$.
- 936 • **domain:** The spatial and temporal domain of the problem, for instance, $x \in [0, 2\pi]$, $t \in$
937 $[0, 0.14\pi]$.
- 938 • **save values:** A list of solution variables (e.g., u , v , p) that should be saved at the final
939 time step.
- 940 • **numerical method** (optional): Specifies the numerical scheme to be used, e.g., finite
941 difference method (FDM), finite volume method (FVM), or finite element method (FEM).
942 This field may be omitted when using FDM as the default method.

943 Example Problem Description in JSON Format

Problem Description (JSON Format)

```
{
  "1D_Burgers_Equation": {
    "equation": "\\[\n \\frac{\\partial u}{\\partial t} + u \\frac{\\partial u}{\\partial x} = \\nu \\frac{\\partial^2 u}{\\partial x^2}\n\\]\n\nwhere:\n- \\( u(x,t) \\) is the velocity field\n- \\( \\nu = 0.07 \\) is the viscosity coefficient\n- \\( x \\) is the spatial coordinate\n- \\( t \\) is time",
    "boundary conditions": "Periodic boundary conditions:\n\\[\n u(0) = u(2\\pi)\n\\]",
    "initial conditions": "\\[\n u = -\\frac{2\\nu}{\\phi} \\frac{\\partial \\phi}{\\partial x} + 4\n\\]\n\nwhere:\n\\[\n \\phi = \\exp\\left(\\frac{-x^2}{4\\nu}\\right) + \\exp\\left(\\frac{-(x - 2\\pi)^2}{4\\nu}\\right)\n\\]",
    "domain": "- Spatial domain: \\( x \\in [0, 2\\pi] \\), - Temporal domain: \\( t \\in [0, 0.14\\pi] \\)",
    "save values": "u",
    "numerical method": "finite difference method"
  },
}
```

944

945 Prompt Generation Function

946 Following the construction of the problem description in JSON format, we systematically generate
947 structured user prompts for the LLM based on the provided information. The conversion from JSON
948 to natural language is automated through a prompt generation function, which formats the fields (e.g.,
949 equation, boundary conditions, initial conditions, domain, save values, and numerical method) into a
950 coherent and standardized instruction. This structured prompt explicitly communicates the problem
951 setup and expected outputs, ensuring consistency across different tasks and minimizing ambiguity
952 during code generation. The generated prompts follow a fixed template to guarantee reproducibility
953 and comparability throughout the benchmark.

954 The prompt is designed to achieve the following goals:

955 **Deterministic and parseable:** The generated prompts follow a consistent structure, enabling easy
956 parsing and reproducibility.

957 **Clear separation of problem components:** Each prompt explicitly isolates the problem definition,
958 including the partial differential equation (PDE), boundary conditions (BC), initial conditions (IC),
959 and domain specifications.

960 **Specification of code generation requirements:** Prompts clearly define additional requirements
961 such as the numerical method, output format, and variables to be saved.

962 **Solver-agnostic design:** While prompts recommend a numerical method (e.g., finite difference
963 method (FDM)), they remain flexible and do not enforce dependence on a particular solver framework.

Prompt Generation Function

```
def generate_prompt(data):
    parts = [
        "You are given the following partial differential equation (PDE) problem:\n",
        "**Equation:**\n" + data.get("equation", "") + "\n",
        "**Boundary Conditions:**\n" + data.get("boundary conditions", "") + "\n",
        "**Initial Conditions:**\n" + data.get("initial conditions", "") + "\n",
        "**Domain:**\n" + data.get("domain", "") + "\n",
        "**Numerical Method:**\n" + data.get("numerical method", "") + "\n"
    ]

    # Check for 'save_values' and add to task description
    save_values = data.get("save_values", [])
    save_values_str = ", ".join(save_values) if save_values else "the relevant variables specified for the problem"

    # Always end with task specification for the code
    parts.append(
        "### Task:\n"
        "- Write Python code to numerically solve the given CFD problem. Choose an appropriate numerical method based "
        "on the problem characteristics.\n"
        "- If the problem is **unsteady**, only compute and save the **solution at the final time step**.\n"
        "- For each specified variable, save the final solution as a separate '.numpy' file using NumPy:\n"
        "  - For **1D problems**, save each variable as a 1D NumPy array.\n"
        "  - For **2D problems**, save each variable as a 2D NumPy array.\n"
        "- The '.numpy' files should contain only the final solution field (not intermediate steps) for each of the "
        "specified variables.\n"
        "- **Save the following variables** at the final time step:\n"
        + save_values_str + "\n"
        "(Each variable should be saved separately in its own '.numpy' file, using the same name as "
        "provided in 'save_values').\n"
        "- Ensure the generated code properly handles the solution for each specified variable "
        "and saves it correctly in '.numpy' format.\n"
        "- **Return only the complete, runnable Python code** that implements the above tasks, "
        "ensuring no extra explanations or information is included."
    )

    return "\n".join(parts)
```

964

965 **Example Generated User Prompt**

Generated User Prompt

```
{
  "1D_Burgers_Equation": "You are given the following partial
    differential equation (PDE) problem:\n\n**Equation:**\n\\[
    \\frac{\\partial u}{\\partial t} + u \\frac{\\partial u}{\\partial x} = \\nu \\frac{\\partial^2 u}{\\partial x^2}
  \\]\n\nwhere:\n- \\( u(x,t) \\) is the velocity field\n- \\( \\nu = 0.07 \\) is the viscosity coefficient\n- \\( x \\) is the spatial coordinate\n- \\( t \\) is time\n\n**Boundary Conditions:**\nPeriodic boundary conditions:\n\\[ u(0) = u(2\\pi) \\]\n\n**Initial Conditions:**\n\\[ u = -\\frac{2\\nu}{\\phi} \\frac{\\partial \\phi}{\\partial x} + 4 \\]\n\nwhere:\n\\[ \\phi = \\exp\\left(\\frac{-x^2}{4\\nu}\\right) + \\exp\\left(\\frac{-(x - 2\\pi)^2}{4\\nu}\\right) \\]\n\n**Domain:**\n- Spatial domain: \\( x \\in [0, 2\\pi] \\), - Temporal domain: \\( t \\in [0, 0.14\\pi] \\)\n\n**Numerical Method:**\nfinite difference method\n\n### Task:\n- Write Python code to numerically solve the given CFD problem. Choose an appropriate numerical method based on the problem characteristics.\n- If the problem is **unsteady**, only compute and save the **solution at the final time step**.\n- For each specified variable, save the final solution as a separate '.numpy' file using NumPy:\n  - For **1D problems**, save each variable as a 1D NumPy array.\n  - For **2D problems**, save each variable as a 2D NumPy array.\n- The '.numpy' files should contain only the final solution field (not intermediate steps) for each of the specified variables.\n- **Save the following variables** at the final time step:\nthe relevant variables specified for the problem\n(Each variable should be saved separately in its own '.numpy' file, using the same name as provided in 'save_values').\n- Ensure the generated code properly handles the solution for each specified variable and saves it correctly in '.numpy' format.\n- **Return only the complete, runnable Python code** that implements the above tasks, ensuring no extra explanations or information is included.",
}
```

966

967 **System Prompt** In addition to the user prompt, we employ a fixed system prompt to explicitly
 968 define the role of the LLM. For models that do not support system prompts natively, the system
 969 prompt is appended to the beginning of the user prompt to ensure consistent behavior across different
 970 models.

System Prompt

```
"You are a highly skilled assistant capable of generating Python code
to solve CFD problems "
    "using appropriate numerical methods."
    "Given the problem description, you should reason
    through the problem and determine the best "
    "approach for discretizing and solving it,"
    "while respecting the specified boundary conditions
    , initial conditions, and domain.\n"
    "For unsteady problems, save only the solution at
    the final time step. For 1D problems, "
    "save a 1D array; for 2D problems, save a 2D array
    .\n"
    "Ensure the code follows the user's specifications
    and saves the requested variables exactly "
    "as named in 'save_values'.\n"
    "Your task is to generate the correct, fully
    runnable Python code for solving the problem "
    "without additional explanations."
```

971

972 **Task Execution Protocol** We evaluate code generation performance across a range of large lan-
973 guage models (LLMs). When configurable, we set the decoding temperature to 0 to minimize
974 randomness and encourage deterministic outputs. For models where temperature is fixed by the
975 provider, we use the default setting. We do not explicitly constrain the maximum number of generated
976 tokens; however, we note that some models impose an internal context window limit of approximately
977 8000 tokens, which encompasses both prompt and output. No additional stop sequences or length
978 truncation strategies are applied.

979 **Execution and Output Validation** For each generated response, we extract the Python code
980 and execute it in a controlled environment. The resulting numerical solution is saved as a NumPy
981 array and compared against the expert-provided reference solution. To quantify the accuracy of the
982 generated results, we compute the *Normalized Mean Squared Error* (NMSE) between the predicted
983 and true solution arrays. In cases where the shapes of the predicted and reference arrays do not match,
984 we apply interpolation to align the dimensions before comparison. Additionally, we visualize both the
985 predicted and reference solutions as images to qualitatively assess agreement and identify structural
986 discrepancies. All code execution is sandboxed with timeouts (default to be 60 seconds) to prevent
987 infinite loops or excessive resource usage.

988 A.3 FoamBench

989 This class of benchmark study focuses on OpenFOAM dataset. Being an open source framework,
990 there are multitudes of cases available. However, scraping through all such available cases to generate
991 the dataset can create additional challenges in evaluating the effectiveness of such frameworks. Also,
992 OpenFOAM is capable of simulating complex geometries that we see in real life scenarios (e.g. flow
993 over an airplane or automobile), which requires creation of a CAD geometry outside of OpenFOAM
994 using specialized tools and further creating a computational mesh, which is then imported into
995 OpenFOAM for further CFD analysis. In the current benchmark, we want to evaluate an end to
996 end usage of OpenFOAM, where it can generate its own geometry and mesh and further do the
997 required numerical analysis. Hence we stick with geometries that can be easily described using
998 natural language and/or part of the tutorial. With these guidelines in mind we curate *FoamBench*
999 *Basic* and *FoamBench Advanced* dataset.

1000 A.3.1 FoamBench Basic

1001 The basic dataset consists of tutorial problems with regards to the physics, geometry and models. We
1002 picked out 11 different tutorial problems namely:

1003 **BernardCells** Simulates Rayleigh-Bénard convection within a rectangular cavity, driven by a
1004 temperature gradient between the hot bottom wall and the cold top wall. It models buoyancy-driven
1005 flow and thermal instabilities using Boussinesq approximation.

1006 **Cavity** Classic lid-driven cavity problem with a square geometry and a moving top wall. It is used
1007 to validate laminar incompressible solvers and study vortex formation.

1008 **counterFlowFlame2D** Models a 2D counter-flow diffusion flame with detailed combustion and
1009 chemistry. The domain consists of opposing inlets where fuel and oxidizer meet, ideal for flame
1010 structure studies.

1011 **Cylinder** Simulates flow past a stationary cylinder in a 2D channel. Demonstrates vortex shedding
1012 and drag, and is widely used for benchmarking turbulence models.

1013 **damBreakWithObstacle** A multiphase VOF (Volume of Fluid) simulation of a dam break in the
1014 presence of a central obstacle. Tests free-surface dynamics and wave-obstacle interactions.

1015 **forwardStep** Compressible flow over a sudden forward-facing step in a duct. Used to observe
1016 shock reflections, expansion fans, and flow separation at high Mach numbers.

1017 **obliqueShock** This case simulates compressible, inviscid supersonic flow over a flat domain,
1018 leading to the formation of an oblique shock wave. Unlike classic textbook setups that use a physical
1019 wedge to induce the shock, this case creates an oblique shock by imposing different velocity and
1020 temperature conditions at the inlet boundary of a flat channel.

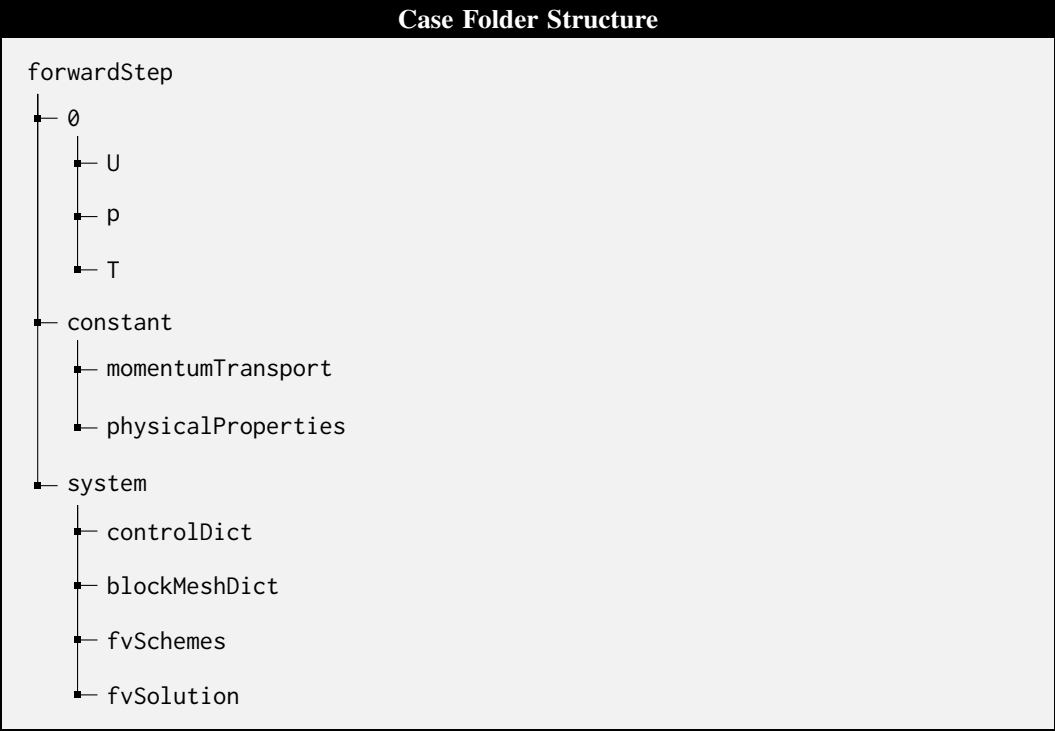
1021 **pitzDaily** Simulates turbulent flow in a channel with a backward-facing step. It is a standard test
1022 case for turbulence model validation due to its separation and reattachment zones.

1023 **shallowWaterWithSquareBump** Uses the shallow water equations to model surface flow over a
1024 square bump. Tests numerical schemes for water surface deformation and hydraulic jumps.

1025 **squareBend** Involves internal flow through a 90-degree square bend. Demonstrates secondary flow
1026 effects caused by pressure gradients in curved channels.

1027 **wedge** An axisymmetric setup often used for supersonic flow around wedges. Useful for studying
1028 inviscid compressible flows with symmetry assumptions.

1029 We create 10 variations for each of these 11 datasets by varying parameters such as inlet velocity,
1030 viscosity, boundary temperature values etc. This gives us 110 distinct OpenFOAM case files which
1031 can be used as reference. The benchmark dataset consists of reference human made OpenFOAM case
1032 files and a prompt for the agentic framework describing the problem to be solved. We provide an
1033 example from our basic benchmark dataset for the forwardStep case. The reference folder structure
1034 in which the files are to be organized is shown below.



We show the details of these files which is used as reference in Figure 6, Figure 7 and Figure 8

```
foamFile
{
    format      ascii;
    class       volScalarField;
    object      p;
}
// ..... //

dimensions      [1 -1 -2 0 0 0];
internalField   uniform 1;
boundaryField
{
    inlet
    {
        type      fixedValue;
        value      uniform 1;
    }
    outlet
    {
        type      zeroGradient;
    }
    bottom
    {
        type      symmetryPlane;
    }
    top
    {
        type      symmetryPlane;
    }
    obstacle
    {
        type      zeroGradient;
    }
    defaultFaces
    {
        type      empty;
    }
}
// ..... //

foamFile
{
    format      ascii;
    class       volScalarField;
    object      T;
}
// ..... //

dimensions      [0 0 0 1 0 0];
internalField   uniform 1;
boundaryField
{
    inlet
    {
        type      fixedValue;
        value      uniform 1;
    }
    outlet
    {
        type      inletOutlet;
        inletValue uniform 1;
        value      uniform 1;
    }
    bottom
    {
        type      symmetryPlane;
    }
    top
    {
        type      symmetryPlane;
    }
    obstacle
    {
        type      zeroGradient;
    }
    defaultFaces
    {
        type      empty;
    }
}
// ..... //

foamFile
{
    format      ascii;
    class       volVectorField;
    object      up;
}
// ..... //

dimensions      [0 1 -1 0 0 0];
internalField   uniform (4 0 0);
boundaryField
{
    inlet
    {
        type      fixedValue;
        value      uniform (4 0 0);
    }
    outlet
    {
        type      inletOutlet;
        inletValue uniform (4 0 0);
        value      uniform (4 0 0);
    }
    bottom
    {
        type      symmetryPlane;
    }
    top
    {
        type      symmetryPlane;
    }
    obstacle
    {
        type      slip;
    }
    defaultFaces
    {
        type      empty;
    }
}
// ..... //
```

Figure 6: OpenFoam reference case files defining the initial and boundary conditions.

Finally the prompt that is input to the frameworks is shown below. Since they are tutorial problems, we do not describe the geometry in great lengths and assume the RAG should be able to pick it out based on the description.

Prompt

Do a laminar, compressible flow over a forward-facing step using the rhoCentralFoam solver. Boundary conditions include a fixed velocity of 3 m/s and temperature of 1K at the inlet and slip conditions on the obstacle. Use a timestep of 0.002 and output every 0.1. Final time is 4.

A.3.2 FoamBench Advanced

The advanced dataset consists of cases that are not part of the tutorial. These cases are used to evaluate the LLMs capabilities in piecing together the information from available tutorials and extrapolating to major changes that the user requests in the following categories:

- **Turbulence Model Changes:** Changes in turbulence models requires the LLM to understand the required file changes and parameter changes. Shifting from one turbulence model to another not only requires simple option changes in files but may also involve additional file to be created in the initial and boundary condition folders specifying the parameters for the given turbulence model.
- **Geometric Modifications:** We ask the LLM to make changes to the geometry in tutorial problems in changing the size of the domain. This requires the LLM to understand spatial configuration of a given problem and make the required changes to the domain configuration in the geometry definition.
- **Unseen Geometry:** In these tasks we ask LLM to create new obstacle shapes within the flow domain. Such tasks can be quite complex in nature, requiring the LLM to understand the new geometric requirements of the user and piece together information from its own knowledge and tutorial cases and perform spatial reasoning to generate these geometries.

We have curated a total of 16 cases, covering the above mentioned aspects of extrapolation. A sample prompt given as the input to the framework is shown below

Prompt

Perform an incompressible turbulent flow simulation over a 2D diamond obstacle using the k-epsilon RANS turbulence model and pimpleFoam solver. The computational domain spans 0 to 15 in x direction and 0 to 5 in y direction and -0.5 to 0.5 in z direction. The diamond obstacle is a square rotated by 45 degrees with diagonal length of 1 unit centered at 2.5 x 2.5 x 0.0. Use one cell in z direction making the geometry effectively 2D. Refine the mesh near to diamond. Use sufficient grid points to discretize the domain and dont use more than 10000 cells in your mesh. The left boundary is the inlet which uses a uniform velocity of (1,0,0) m/s. The right boundary is the outlet using zero gradient pressure condition. Top and bottom boundaries are fixed walls with no-slip condition. The front and back faces are empty. The diamond obstacle also has no-slip boundary condition on its surface. The kinematic viscosity is $2e-6 \text{ m}^2/\text{s}$. Use a deltaT of 0.5 s and run till a final time of 5 s. Write the results at every 0.5 s. Use a maximum Courant number of 1.0.

Unlike *FoamBench Basic* these are unseen geometries. Hence the prompt is made descriptive enough for the LLM to understand the user need and generate the relevant blockMeshDict file containing the geometric details and mesh information. Since it is difficult to perform mesh control over natural language we only specify the maximum number of cells to be used and required refinement near the obstacle.

B Agentic Framework

B.1 Detailed Performance Comparison

Table 3: Performance of **Zero Shot Pure LLM** on *FoamBench Basic* and *FoamBench Advanced*.

Variation	Model	<i>FoamBench Basic</i>					<i>FoamBench Advanced</i>				
		M_{exec}	M_{struct}	M_{file}	M_{NMSE}	Success Rate	M_{exec}	M_{struct}	M_{file}	M_{NMSE}	Success Rate
Zero Shot Pure LLM	Sonnet 3.5	0.064	0.670	0.506	0.050	0.045	0.017	0.773	0.573	0.009	0.007
	o3-mini	0.009	0.788	0.529	0.009	0.009	0.000	0.707	0.408	0.000	0.000
	Gemini 2.5 Flash	0.000	0.828	0.573	0.000	0.000	0.000	0.666	0.406	0.000	0.000
	Haiku 3.5	0.000	0.905	0.629	0.000	0.000	0.000	0.801	0.492	0.000	0.000
	GPT-4o	0.000	0.819	0.589	0.000	0.000	0.000	0.735	0.466	0.000	0.000
	Gemma-2-9B-IT	0.000	0.735	0.460	0.000	0.000	0.000	0.670	0.390	0.000	0.000

Table 4: Component-wise mean scores and true-Success Rates for each model and framework on *FoamBench* Basic (top) and Advanced (bottom).

Variation	Model	MetaOpenFOAM					Foam-Agent				
		M_{exec}	M_{struct}	M_{file}	M_{NMSE}	Success Ratio	M_{exec}	M_{struct}	M_{file}	M_{NMSE}	Success Ratio
RAG + Reviewer	Sonnet 3.5	0.555	0.883	0.763	0.173	0.136	0.836	0.879	0.778	0.427	0.336
	o3-mini	0.491	0.872	0.664	0.236	0.227	0.573	0.883	0.772	0.291	0.264
	Gemini 2.5 Flash	0.245	0.841	0.695	0.091	0.082	0.182	0.568	0.496	0.141	0.136
	Haiku 3.5	0.218	0.845	0.701	0.095	0.091	0.518	0.921	0.797	0.205	0.191
	GPT-4o	0.173	0.801	0.715	0.105	0.091	0.591	0.878	0.765	0.309	0.282
	Gemma-2-9B-IT	0.000	0.690	0.540	0.000	0.000	0.000	0.710	0.590	0.000	0.000
RAG + No Reviewer	Sonnet 3.5	0.064	0.810	0.728	0.023	0.009	0.373	0.668	0.599	0.232	0.200
	o3-mini	0.055	0.823	0.651	0.027	0.027	0.436	0.837	0.744	0.273	0.255
	Gemini 2.5 Flash	0.009	0.793	0.682	0.009	0.009	0.191	0.811	0.685	0.145	0.136
	Haiku 3.5	0.055	0.806	0.708	0.027	0.027	0.182	0.915	0.799	0.100	0.091
	GPT-4o	0.045	0.796	0.710	0.018	0.018	0.455	0.843	0.738	0.286	0.255
	Gemma-2-9B-IT	0.000	0.680	0.540	0.000	0.000	0.000	0.720	0.590	0.000	0.000
No RAG + Reviewer	Sonnet 3.5	0.400	0.747	0.522	0.195	0.145	0.473	0.862	0.647	0.291	0.245
	o3-mini	0.045	0.623	0.347	0.000	0.000	0.009	0.811	0.549	0.009	0.009
	Gemini 2.5 Flash	0.009	0.609	0.364	0.009	0.009	0.000	0.829	0.571	0.000	0.009
	Haiku 3.5	0.000	0.587	0.346	0.000	0.000	0.009	0.910	0.633	0.009	0.009
	GPT-4o	0.000	0.557	0.341	0.000	0.000	0.000	0.017	0.012	0.000	0.000
	Gemma-2-9B-IT	0.000	0.420	0.220	0.000	0.000	0.000	0.600	0.480	0.000	0.000
Variation	Model	MetaOpenFOAM					Foam-Agent				
		M_{exec}	M_{struct}	M_{file}	M_{NMSE}	Success Ratio	M_{exec}	M_{struct}	M_{file}	M_{NMSE}	Success Ratio
RAG + Reviewer	Sonnet 3.5	0.125	0.775	0.599	0.125	0.125	0.625	0.792	0.621	0.406	0.250
	o3-mini	0.125	0.665	0.484	0.125	0.125	0.312	0.734	0.597	0.219	0.187
	Gemini 2.5 Flash	0.000	0.796	0.586	0.000	0.000	0.062	0.692	0.511	0.000	0.000
	Haiku 3.5	0.062	0.646	0.498	0.031	0.000	0.188	0.840	0.609	0.188	0.187
	GPT-4o	0.000	0.514	0.430	0.000	0.000	0.375	0.805	0.634	0.312	0.250
	Gemma-2-9B-IT	0.000	0.540	0.410	0.000	0.000	0.000	0.630	0.450	0.000	0.000
RAG + No Reviewer	Sonnet 3.5	0.000	0.743	0.594	0.000	0.000	0.188	0.771	0.609	0.156	0.125
	o3-mini	0.000	0.746	0.535	0.000	0.000	0.000	0.702	0.566	0.000	0.000
	Gemini 2.5 Flash	0.000	0.688	0.518	0.000	0.000	0.000	0.666	0.496	0.000	0.000
	Haiku 3.5	0.000	0.654	0.518	0.000	0.000	0.000	0.801	0.583	0.000	0.000
	GPT-4o	0.000	0.733	0.603	0.000	0.000	0.000	0.744	0.594	0.000	0.000
	Gemma-2-9B-IT	0.000	0.530	0.400	0.000	0.000	0.000	0.610	0.440	0.000	0.000
No RAG + Reviewer	Sonnet 3.5	0.375	0.655	0.451	0.344	0.187	0.250	0.806	0.592	0.188	0.125
	o3-mini	0.250	0.649	0.372	0.062	0.000	0.000	0.710	0.420	0.000	0.000
	Gemini 2.5 Flash	0.000	0.685	0.407	0.000	0.000	0.000	0.702	0.410	0.000	0.000
	Haiku 3.5	0.000	0.718	0.456	0.000	0.000	0.000	0.825	0.511	0.000	0.000
	GPT-4o	0.188	0.658	0.415	0.000	0.000	0.000	0.710	0.443	0.000	0.000
	Gemma-2-9B-IT	0.000	0.500	0.360	0.000	0.000	0.000	0.590	0.400	0.000	0.000

B.2 Reasons For Failure

We examine the common reasons for failure of execution for the cases in *FoamBench Basic* dataset for the two frameworks with the best performing model (Sonnet 3.5) in Figure 9. The mentioned reasons can be further elaborated as:

- **Inconsistent Patch or Patch Field:** This error means that certain boundaries that was defined in the boundary conditions file does not exist in the mesh file.
- **File Not Found:** This happens when a certain file, example: blockMeshDict, controlDict etc, required for the OpenFoam run is not found among the case files.
- **Undefined keyword:** This happens when certain keywords like the flux schemes or parameter values are not defined appropriately. The LLM will good knowledge about the OpenFoam to decided, which flux schemes are to be defined based on the solver that is being used and the variables that are being solved for.
- **Numerical Instability:** This occurs when the chosen numerical schemes, time step size, or boundary/initial conditions lead to unstable simulations, often causing divergence or NaN values in the solution. It typically arises from violating stability criteria (e.g., CFL condition) or poor discretization choices that amplify numerical errors during time integration.

- **Geometry/Mesh Error:** These errors stem from issues in mesh generation or geometry definition, such as non-orthogonal cells, skewed elements, or overlapping/missing faces. They can cause solver initialization to fail or lead to inaccurate or unphysical results during the simulation.

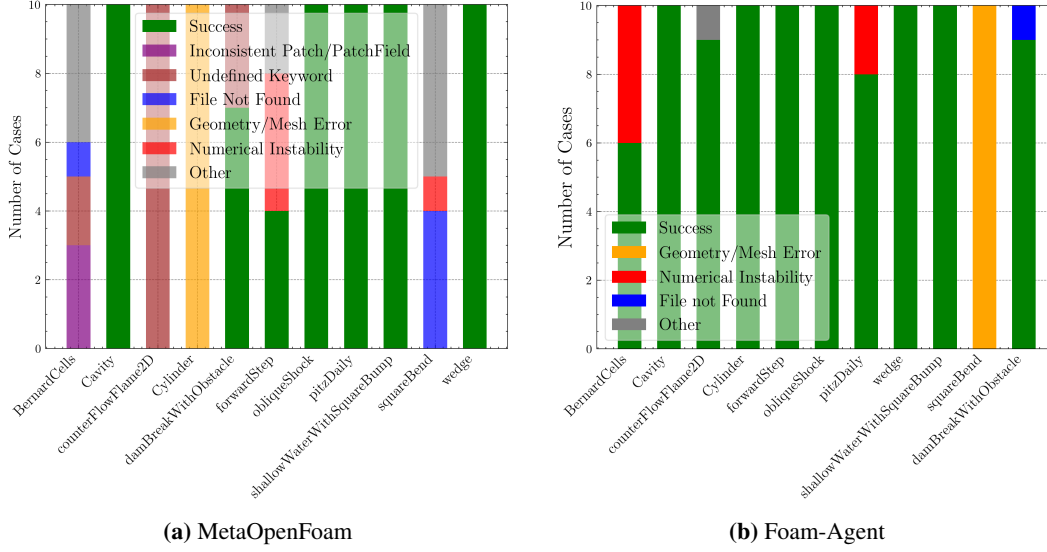


Figure 9: Common Reasons for execution failure found in MetaOpenFoam and Foam-Agent with RAG and Reviewer and using Sonnet 3.5 as the prompt model.

B.3 Token Usage

The token usage statistics of the two frameworks in combination with the different models is shown in Table 5.

Table 5: Average total token usage, API cost (\$) as of May 2025, and loop counts per model variant for each agentic framework. Average is over all cases in *FoamBench*.

Variation	Model	MetaOpenFOAM				Foam-Agent			
		Prompt	Completion	Cost (\$)	Loop	Prompt	Completion	Cost (\$)	Loop
RAG + Reviewer	Sonnet 3.5	63061.55	8337.20	1.19	7	378848.37	9346.10	6.56	2
	o3-mini	60273.86	45325.69	0.29	8	197334.74	6549.15	0.27	4
	Gemini 2.5 Flash	50603.35	8454.29	0.01	9	47929.00	4127.96	0.01	2
	Haiku 3.5	29021.51	6499.74	0.06	9	426189.16	18235.88	0.43	5
	GPT-4o	73603.88	9109.8	0.28	9	147011.94	5702.80	0.42	9
	Gemma-2-9B-IT	35487.00	7322.00	-	10	331582.00	10322.00	-	10
RAG + No Reviewer	Sonnet 3.5	10922.04	5526.07	0.13	1	278880.44	7358.98	5.0	1
	o3-mini	10128.16	24162.77	0.12	1	121070.15	3154.96	0.16	1
	Gemini 2.5 Flash	129137.86	5935.76	0.02	1	129137.86	5935.76	0.02	1
	Haiku 3.5	11103.52	5563.83	0.06	1	110582.05	7818.34	0.50	1
	GPT-4o	9251.12	4662.87	0.07	1	80967.33	4389.14	0.24	1
	Gemma-2-9B-IT	8722.00	6670.00	-	1	98431.00	8888.00	-	1
No RAG + Reviewer	Sonnet 3.5	39030.58	13369.42	0.54	9	92618.95	13369.42	1.28	4
	o3-mini	47062.19	47868.35	0.33	10	89131.14	5319.50	0.15	4
	Gemini 2.5 Flash	50253.88	10123.70	0.01	10	40675.74	6415.07	0.01	2
	Haiku 3.5	24907.50	6663.95	0.04	10	111263.76	14704.62	0.61	4
	GPT-4o	52901.2	7958.66	0.21	10	202418	10972	0.55	4
	Gemma-2-9B-IT	29888.00	7799.00	-	10	95112.00	16444.00	-	10

1091 C Solution Comparison

1092 C.1 CFDQuery

Question:

Which of the following is closest to the correct modified equation for the discretized 1D advection equation using second-order central difference in space and third-order Runge-Kutta (RK3) in time?

Options:

1. $\frac{\partial u}{\partial t} + a \frac{\partial u}{\partial x} = \frac{a\Delta x^2}{6} \frac{\partial^3 u}{\partial x^3} + \mathcal{O}(\Delta x^3)$
2. $\frac{\partial u}{\partial t} + a \frac{\partial u}{\partial x} = \frac{a\Delta x^2}{2} \frac{\partial^2 u}{\partial x^2} + \mathcal{O}(\Delta x^3)$
3. $\frac{\partial u}{\partial t} + a \frac{\partial u}{\partial x} = -\frac{a\Delta x^2}{6} \frac{\partial^3 u}{\partial x^3} + \frac{a\Delta t^2}{6} \frac{\partial^3 u}{\partial t^3} + \mathcal{O}(\Delta x^3)$
4. $\frac{\partial u}{\partial t} + a \frac{\partial u}{\partial x} = \frac{a\Delta x^2}{6} \frac{\partial^3 u}{\partial x^3} - \frac{a^3\Delta t^2}{6} \frac{\partial^3 u}{\partial x^3} + \mathcal{O}(\Delta x^3)$

Correct Answer: Option 4

Model Responses:

- **Sonnet 3.5:** Option 4 ✓
- **o3-mini:** Option 3 ✗
- **Gemini 2.5 Flash:** Option 4 ✓
- **Haiku 3.5:** Option 1 ✗
- **GPT-4o:** Option 3 ✗
- **Gemma-2-9B-IT:** Option 1 ✗

1093

1094 C.2 CFDCoDeBench

1095 The visual comparison of the model produced results and the ground truth solution at the final
1096 timestep for the 1D Burgers equation is shown in Figure 10 and for the 2D convection equation is
1097 given in Figure 11. Models such as o3-mini, Haiku 3.5 and Gemini 2.5 Flash is able to closely match
1098 the ground truth solution for the 1D Burgers equation. Sonnet 3.5 is able to match the solution near
1099 the shock, but does not get the boundary conditions right.

1100 In the solution for 2D convection, Sonnet 3.5 seems to have some numerical instability, while the
1101 other models seems to have a decent prediction of the x directional velocity.

1102 C.3 FoamBench

1103 Figure 12 and Figure 13 shows the comparison between the results from the two frameworks
1104 (*MetaOpenFOAM* and *Foam-Agent*) for the Cavity and forwardStep case respectively. The results
1105 from **Foam-Agent** is more similar to the ground truth in the Cavity case, while both the frameworks
1106 does well in the case of forwardStep.

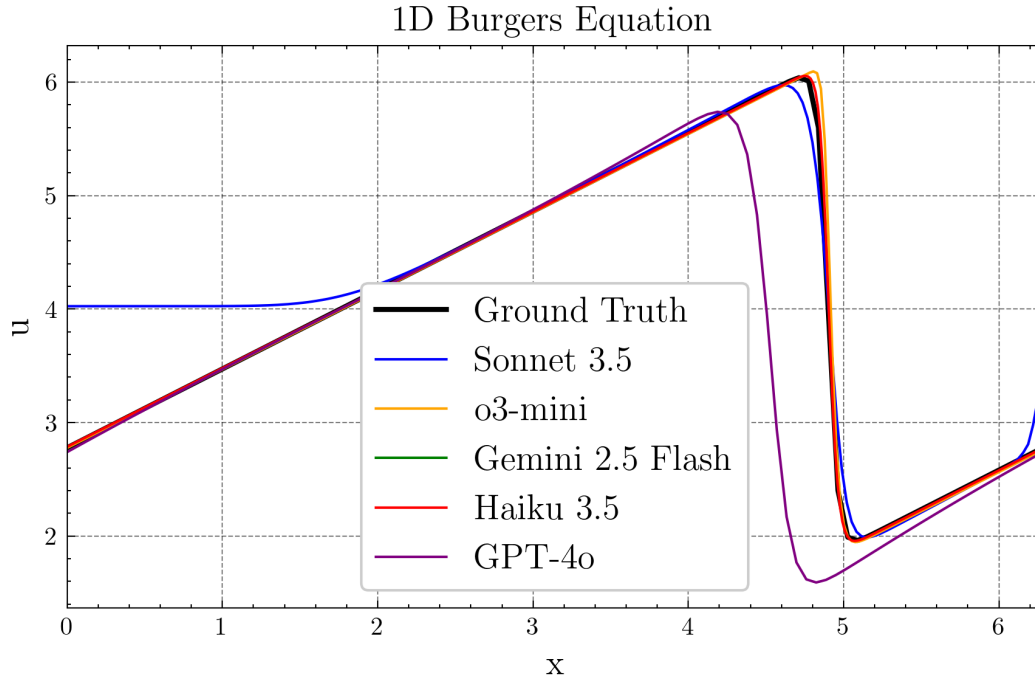


Figure 10: Solution comparison at the final time step for 1D Burgers equation

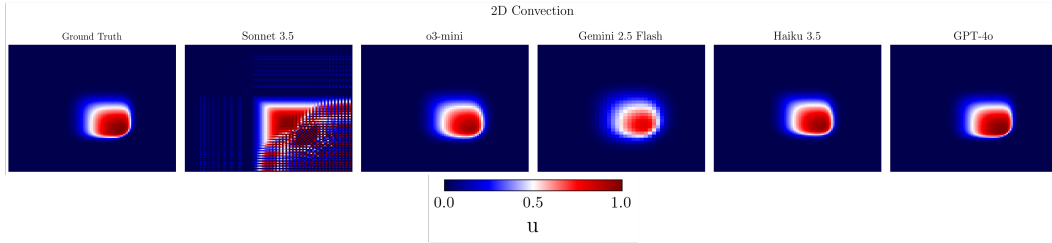


Figure 11: X direction velocity (u) comparison at the final time step for 2D Convection equation

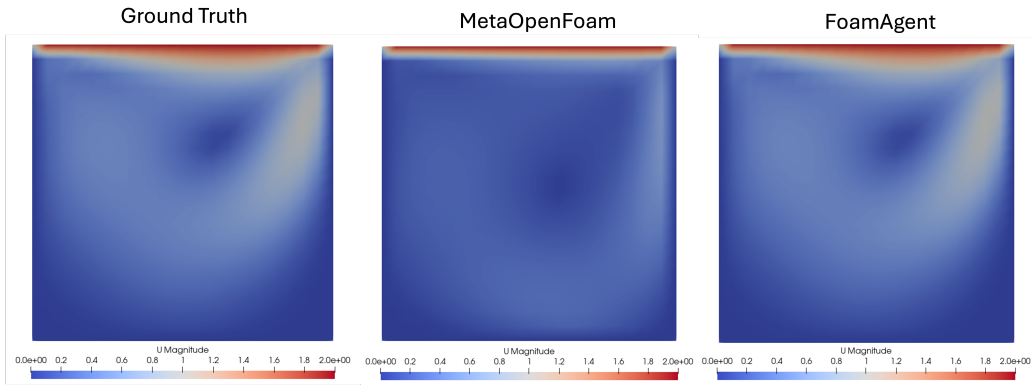


Figure 12: Comparison of velocity magnitude at the final timestep for 2D Cavity case.

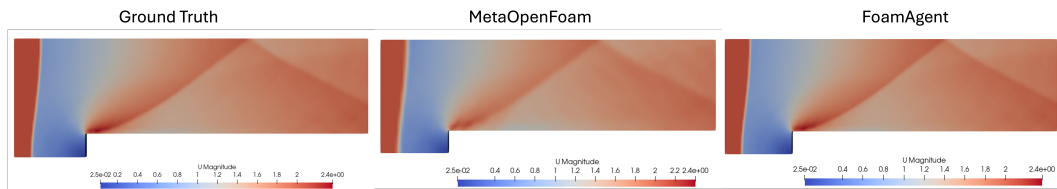


Figure 13: Comparison of velocity magnitude at the final timestep for 2D forwardStep case.