FOSSEE CFD-OpenFOAM Semester Long Internship Spring 2026 on

Domain decomposed Agentic RAG-based pipeline for OPENFOAM case synthesis.

Report Submitted by

Gokulpriyadharsan M

B.Sc(Honors).Physics
University of Delhi

Guide

Dr. Chandan Bose
Assistant Professor
University of Birmingham

Mentor

Mr Pranay Pandey
FOSSEE-CFD
Indian Institute of Technology Bombay

Contents

Abstract

This presents an automated **OpenFOAM** case generation pipeline using an **RAG**-based (Retrieval-Augmented Generation) framework. The primary objective of this work is to reduce the manual effort involved in setting up **OpenFOAM CFD** simulations by integrating **LLMs** (Large Language Models) with structured domain data extracted from OpenFOAM tutorial cases. The system consists of a multi-agent pipeline comprising various components for solver detection, case planning, document retrieval, generation and validation. The retrieval system is implemented through **FAISS** vector stores. The database is created using the default OpenFOAM tutorial files and is properly classified into five types based on their relative sizes and functionality within the OpenFOAM case. Since the system uses a stateless LLM for generation, the quality of the files generated would depend on the underlying LLM The results demonstrate that the pipeline is capable of generating minimal, solver-runnable cases for a range of incompressible flow problems. This work highlights the potential of combining retrieval-based techniques with generative models to develop intelligent engineering tools, thereby improving **accessibility** and **automation in CFD workflows**.

Introduction

CFD simulations using OpenFOAM requires manual construction of case directories, including configuration files, boundary conditions, solver settings, and mesh data. This process is often time-consuming and demands a good understanding of OpenFOAM's file structure and syntax, making it challenging for beginners and hard to rapid prototype.

By integrating LLM, most of the redundant part could be automated by have enabling the generation of structured syntax from natural language prompts. However, naive use of LLMs often leads to inconsistent or invalid OpenFOAM cases due to lack of domain understanding. Retrieval-Augmented Generation (RAG) has been explored to address this issue by adding domain aware knowledge during generation. In most existing RAG-based approaches for OpenFOAM, a single monolithic database containing all types of simulation files is used for retrieval. This lack of structural separation could lead to irrelevant context being retrieved, adding noise to the generation, reducing the reliability and accuracy of generated outputs.

In this project, a structured multi-agent RAG pipeline is developed for automated OpenFOAM case generation. The system introduces a classified retrieval framework, where domain knowledge is divided into multiple specialized vector databases corresponding to different components of an OpenFOAM case, based on the functionality and relative sizes of the documents. This enables more targeted and context-aware retrieval during file generation.

1 Background Concepts

1.1 Computational Fluid Dynamics (CFD)

Computational Fluid Dynamics (CFD) is a branch of computational physics that focuses on the numerical simulation of fluid flow and related transport phenomena. CFD techniques

are widely used in engineering and scientific applications to analyze the behavior of fluids under different physical conditions.

Modern CFD workflows typically involve geometry preparation, mesh generation, specification of material properties, definition of boundary conditions, solver configuration, and post-processing of simulation results. While CFD provides significant flexibility and analytical capability, the setup process can become highly complex, especially for large or multi-physics simulations.

1.2 OpenFOAM Framework and Case Structure

OpenFOAM is the free, open source CFD software developed primarily by OpenCFD Ltd since 2004. It has a large user base across most areas of engineering and science, from both commercial and academic organisations. OpenFOAM has an extensive range of features to solve anything from complex fluid flows involving chemical reactions, turbulence and heat transfer, to acoustics, solid mechanics and electromagnetics. A standard OpenFOAM case generally consists of three primary directories:

- **0/** : Contains field initialization and boundary condition files such as velocity (`U`) and pressure (`p`).
- **constant/** : Stores physical properties, turbulence models, transport properties, and mesh-related information.
- **system/** : Contains solver configuration files including `controlDict`, `fvSchemes`, and `fvSolution`.

Each file follows a strict dictionary-based syntax. Even small formatting or configuration errors can cause simulations to fail during execution. Due to the modular structure of OpenFOAM cases, automating case generation requires proper understanding of the role and dependency of each component.

1.3 Large Language Models (LLMs)

Large Language Models (LLMs) are neural network based systems trained on large amounts of textual data to generate structured and context-aware outputs. Recent advancements in LLMs have demonstrated strong capabilities in code generation, reasoning, summarization, and structured text synthesis.

1.4 Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) is a framework that combines information retrieval with generative language models. Instead of relying solely on internal model knowledge, relevant external documents are first retrieved and then provided as context to the language model during generation.

The use of RAG is particularly important in domain-specific applications such as CFD case generation, where accuracy and consistency are essential. Pure LLM-based generation

may produce hallucinated or invalid OpenFOAM configurations due to insufficient grounding in real simulation examples.

1.5 Vector Embeddings and FAISS Retrieval

To enable semantic retrieval of OpenFOAM configurations, textual simulation data is converted into numerical vector representations known as embeddings, which capture semantic relationships between documents. In this work, the `nomic-embed-text` model was used to generate 768-dimensional embeddings for OpenFOAM documentation, solver configurations, and dictionary files. These embeddings were indexed using Facebook AI Similarity Search (FAISS), enabling efficient semantic retrieval of contextually relevant simulation data through vector similarity search.

1.6 Agentic Workflows

Agentic workflows refer to computational systems where multiple specialized agents collaborate to solve a larger task. Instead of relying on a single monolithic generation step, the workflow is divided into smaller modular components, each responsible for a specific operation. In this work, the agents utilize the `deepseek-v3.2:cloud` large language model for reasoning, retrieval planning, and OpenFOAM dictionary generation within the multi-agent pipeline.

2 System Architecture

The developed system follows a modular multi-agent architecture to automate generation and correction of OpenFOAM simulation cases from natural language prompts.

The overall architecture consists of retreival methods and classified database which would be used by multiple agents that perform a dedicated task within the generation pipeline. Instead of relying on a single big generation stage, the workflow decomposes the problem into smaller modular operations such as solver detection, retrieval planning, file synthesis, validation, and error analysis, guiding the LLM generation reducing abrupt LLM dependency.

Table 1 summarizes the major components used in the system and their corresponding functionalities.

This categorized retrieval strategy reduces unnecessary content retreival and improves generation consistency by limiting retrieval operations to the most relevant domain.

The complete workflow is coordinated through the central RAG Pipeline module, which manages retrieval, synthesis, validation, file writing, solver execution, and iterative correction.

Type	Name	Function
Agent	Case Planner Agent	Detects the appropriate OpenFOAM solver and determines the minimal required case structure.
	Retrieval Planner Agentt	Selects the most suitable vector database and generates semantic retrieval queries
	Synthesizer Agent	Generates OpenFOAM dictionary files using retrieved contextual information and LLM-based synthesis
	Error Analyzer Agent	Processes solver logs, identifies root causes of errors, and determines corrective actions
	Validator Agent	Cleans generated outputs by removing markdown artifacts and correcting formatting inconsistencies
Module	FAISS Retriever	Retrieves relevant OpenFOAM tutorial examples using vector similarity search
Database	StructureDB	Stores solver-wise OpenFOAM case structures extracted from tutorial cases.
	boundaryDB	Stores boundary condition related OpenFOAM examples used for field generation
	fvSchemesDB	Stores numerical discretization scheme examples
	fvSolutionDB	Stores solver and algorithm configuration examples
	physicsDB	Stores physical property and transport model related files
	geoDB	Stores a minimal set of geo files for gmsh generation
Scripts	run_case.py	Main execution script responsible for initializing the case generation pipeline, retrieval workflow, dictionary synthesis, validation, and OpenFOAM case construction.
	run_error_-loop.py	Executes the generated OpenFOAM simulation, analyzes solver logs, and performs iterative error correction and regeneration of faulty files.
	Tut_Creation.py	Processes and categorizes OpenFOAM tutorial files into structured datasets for vector database creation.
	VectorDB.py	Generates FAISS vector databases from categorized OpenFOAM tutorial datasets using semantic embeddings.

Table 1: Architecture

The modular architecture improves maintainability, extensibility, and debugging capability while enabling partial automation of OpenFOAM simulation setup using natural language input.

3 Execution Workflow

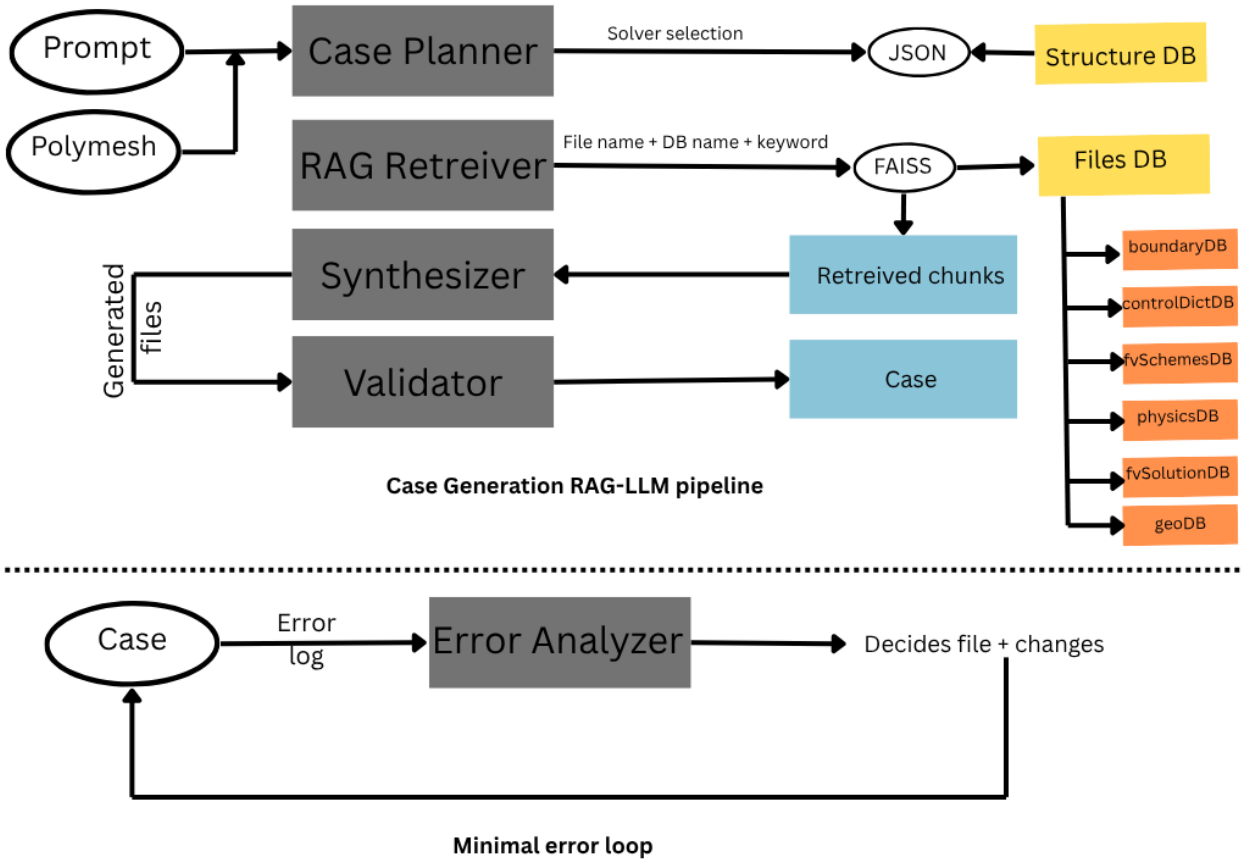

Figure 1: System Workflow

The execution workflow of the developed pipeline is divided into two major operational modules:

- **Case Generation Pipeline**
- **Error Correction and Debugging Loop**

Together, these two modules form a complete workflow generate and correct the generated files.

The execution workflow is explained as follows :

3.1 Case Generation Pipeline

The workflow proceeds through multiple sequential stages, where each stage performs a dedicated operation.

User Prompt Input:

The workflow begins with the user providing a natural language description of the simulation problem through the `input/prompt` file If a mesh is already available, the corresponding `polyMesh` directory is placed inside `input/polyMesh`

The generation pipeline is initiated using:

```
python scripts/run_case.py
```

Pipeline Initialization

Once execution starts, the RAG pipeline initializes all required agents and retrieval modules and the working case directory. This includes the Case Planner Agent, Retrieval Planner Agent, FAISS Retriever, Synthesizer Agent, and Validator Agent.

Solver Detection and Structure Planning

The Case Planner Agent takes in the user prompt and retrurns the most suitable solver. After detecting the solver, the planner retrieves a candidate directory structure from the StructureDB(JSON) database. The retrieved structure is then filtered to retain only the minimal set of files required for the simulation.

Mesh Integration

If the user provides a valid `polyMesh` directory, the mesh is automatically copied into the generated case directory under:

`Case/constant/polyMesh` The boundary file from `Case/polymesh/boundary` is later used for generating consistent boundary conditions inside `Case/0`.

Retrieval Planning and Semantic Retrieval

The Retrieval Planner Agent loops around the files chosen by planner and extracts suitable keywords from the prompt and choses the suitable database to perform retreival.

The generated retrieval query is passed to the FAISS Retriever module, which performs semantic similarity search on the database and returns the most contextually relevant chunks.These chunks serve as contextual grounding for generation.

File Synthesis

The retrieved contextual examples are passed to the Synthesizer Agent along with the user prompt and file generation requirements.

Validation and Formatting

Raw LLM outputs usually contain formatting inconsistencies. The Validator Agent performs formatting cleanup and correction to ensure compatibility between generated outputs and OpenFOAM syntax.

Case Construction

After validation, the generated files are written into the corresponding directories inside the generated OpenFOAM case folder.

Error Correction and Debugging Pipeline

Once the simulation case has been generated, the debugging and correction workflow is initiated using:

```
python scripts/run_error_loop.py
```

The primary objective of this pipeline is to automatically execute the generated simulation case, analyze solver failures, identify the cause of errors, and iteratively regenerate incorrect files until the simulation executes successfully or the correction limit is reached.

3.2 Solver Detection and Error Loop Execution

Provided the Case file, the system detects the solver from the generated `controlDict` file inside the case. After detecting the solver, the pipeline launches the simulation and captures the generated error logs. These logs are monitored to determine whether the simulation completed successfully or failed due to configuration errors.

Error Log Analysis

If the solver execution fails, the generated error logs are passed to the Error Analyzer Agent.

The Error Analyzer Agent analyzes the solver output and attempts to identify the cause of the failure. The analyzer is designed to detect common OpenFOAM issues including:

- missing boundary patches
- invalid dictionary syntax
- inconsistent boundary conditions
- incorrect solver configurations
- scheme related errors

The analyzer also determines which OpenFOAM files are responsible for the detected failure.

Corrective File Regeneration

Once the problematic files are determined, corrective prompts are dynamically generated and passed to the LLM synthesis stage.

The Synthesizer Agent regenerates updated versions of the affected files while incorporating information from the solver error log, previously generated outputs, contextual retrieval information. The regenerated files are then cleaned and formatted using the Validator Agent before being rewritten into the simulation case.

Iterative Correction Loop

After regeneration, the solver is executed again using the rewritten files.

This process continues iteratively until:

- the simulation executes successfully, or
- the maximum correction iteration limit is reached
- the same error presists

4 Results and Limitations

- The modular retrieval architecture allows vector databases to be regenerated or replaced easily, so that the generation is not restricted to a specific OpenFOAM version
- The pipeline successfully identifies the suitable solver and structure of case (required files) from prompt
- The retrieval planning stage effectively extracted semantic retrieval keywords from the prompt to perform similarity search on the complete query.
- The categorized FAISS retrieval system retrieved contextually relevant OpenFOAM tutorial examples for generation.
- Generated outputs generally followed valid OpenFOAM structure and syntax conventions after validation and formatting.
- The automated debugging pipeline identified common execution issues and configuration inconsistencies.
- The responses directly rely on the quality of the underlying LLM used

The system does not consistently generate fully simulation-ready OpenFOAM cases, but produces strong starting templates requiring only minimal manual correction.

5 Future Improvements

- Development of solver-specific agents instead of generic generation agents, so that solver-specific requirements such as turbulence models, buoyancy effects, and multiphysics configurationsc can be handled better
- Integration of physics-aware validation mechanisms to improve physical consistency and reliability of generated OpenFOAM simulations.
- Development of a graphical user interface (GUI) to simplify interaction with the pipeline and improve usability.

- Addition of post-processing agents for automated data extraction, result analysis, and visualization of simulation outputs.
- Expansion of retrieval databases using larger collections of OpenFOAM tutorials and research cases to improve contextual coverage and retrieval quality.

6 Appendix

A Setup and Installation

This section deals with installation and setup of the system.

Step 1: Setup Ollama

The pipeline requires Ollama for running the language model and embedding model.

Install Ollama and login through the terminal using:

```
ollama login
```

Step 2: Download Embedding Model

Download the `nomic-embed-text` embedding model required for semantic retrieval:

```
ollama pull nomic-embed-text
```

Step 3: Install Python Dependencies

The required Python packages are listed inside:

```
ReadMe/requirements.txt
```

Install the dependencies using:

```
pip install -r ReadMe/requirements.txt
```

Step 4: Automatic Setup and Initialization (Recommended)

The project includes automated setup and workflow scripts for easier initialization and execution.

First, make the scripts executable:

```
chmod +x startup.sh
chmod +x allrun
```

Run the startup script:

```
./startup.sh
```

This script automatically handles:

- Python environment setup
- dependency installation
- Ollama setup guidance
- virtual environment creation

If the automatic setup fails or requires manual debugging, continue with the remaining setup steps below.

Step 5: Add OpenFOAM Tutorial Cases
Copy the OpenFOAM tutorial directory into the project folder and place it under:

```
tutorials/
```

The tutorial cases are used for generation of retrieval datasets and vector databases.

Step 6: Generate Flattened Tutorial Dataset
Run the tutorial processing script:

```
python Tut_Creation.py
```

This script processes and categorizes OpenFOAM tutorial files into structured datasets and generates the:

```
Flattened_Tut/
```

directory.

Step 7: Generate Vector Databases
Run the vector database generation script:

```
python VectorDB.py
```

The output should look like :

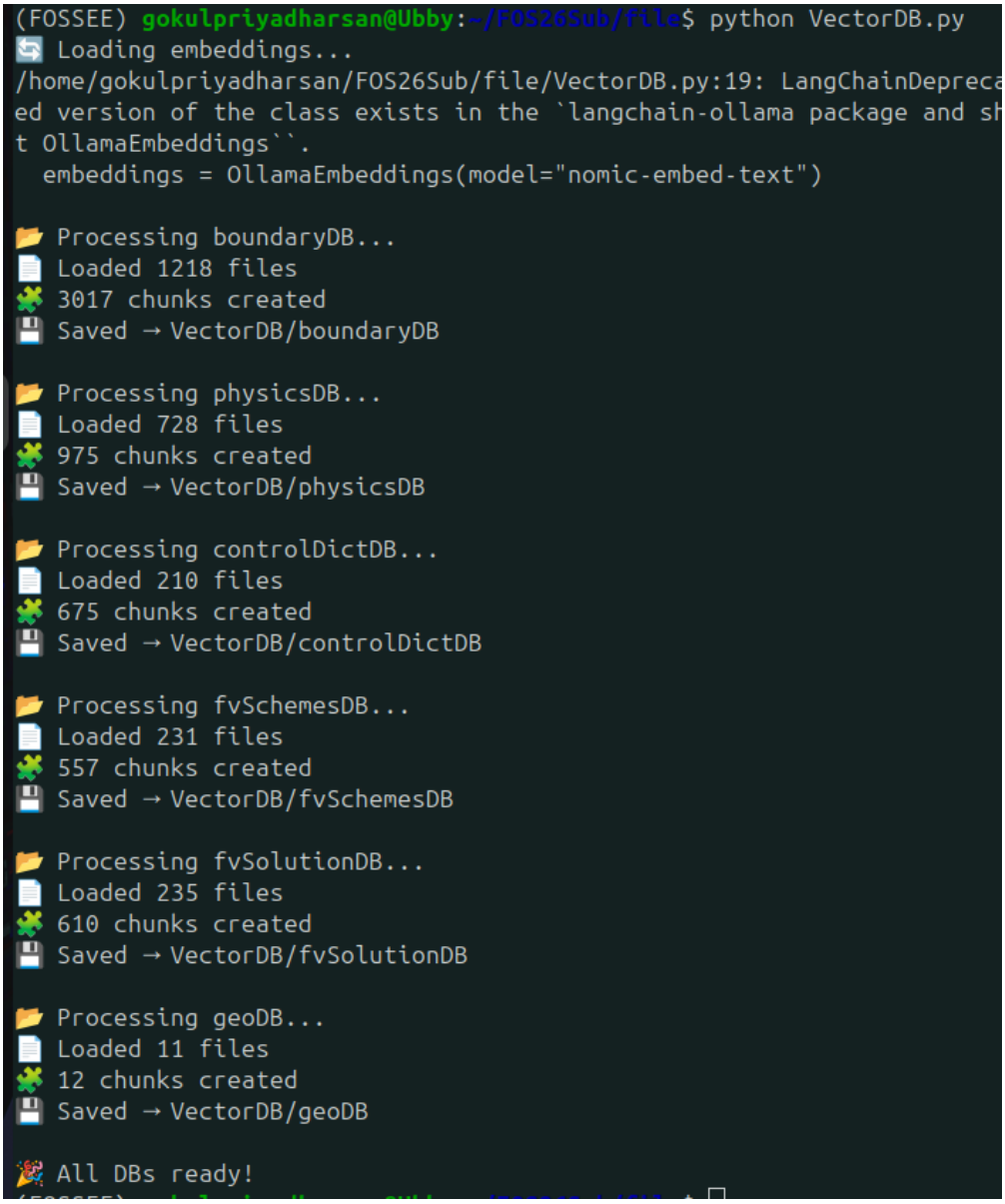

Figure 2: Caption

These databases are later used during semantic retrieval and dictionary generation.

Step 8: Generate Structure Database

Run the structure database generation script:

```
python build_structure_db.py
```

This generates the solver-wise StructureDB used during case planning and structure selection. The output would look like :

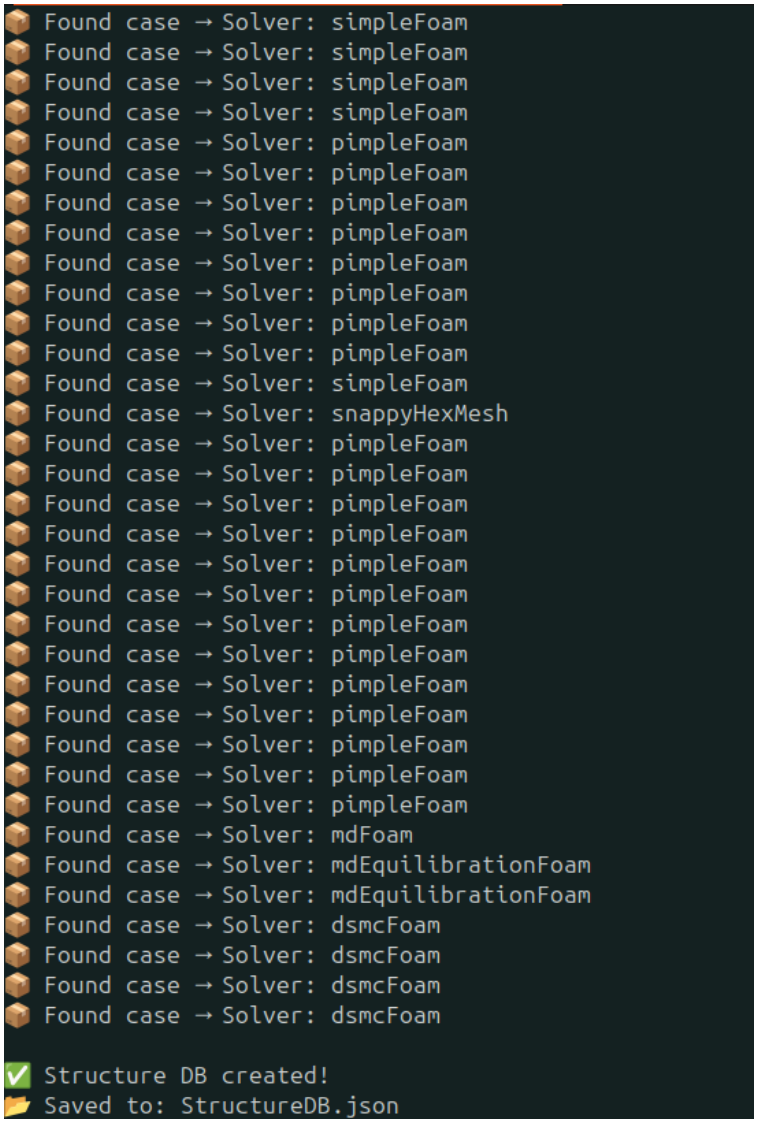

Figure 3: Caption

B Example Case Generation Run

This section demonstrates an example workflow for generating and executing an OpenFOAM simulation case using the developed pipeline.

Step 1: Add Simulation Prompt

Write the simulation description inside:

`input/prompt`

An example prompt is shown below:

Simulate 2D incompressible, steady-state flow over a circular cylinder. Set the

Reynolds number to Re=20. Ensure the mesh is refined near the cylinder surface to capture the stagnation point and the symmetric steady separation bubbles at the rear. Use a non-slip boundary condition on the cylinder wall

Step 2: Add Mesh Files

Copy the required OpenFOAM mesh directory into:

```
input/polyMesh
```

The pipeline directly reuses this mesh during case generation.

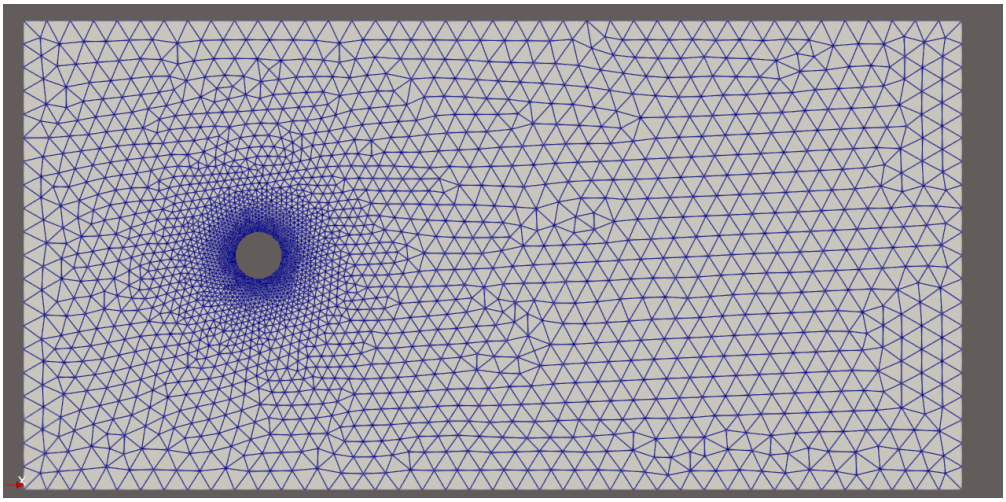

Figure 4: Mesh used

Step 3: Run the Case Generation Pipeline

Execute the case generation script:

```
./allrun
```

Step 4: Generated Case Structure

After execution, the generated OpenFOAM case is created inside:

```
file/Case
```

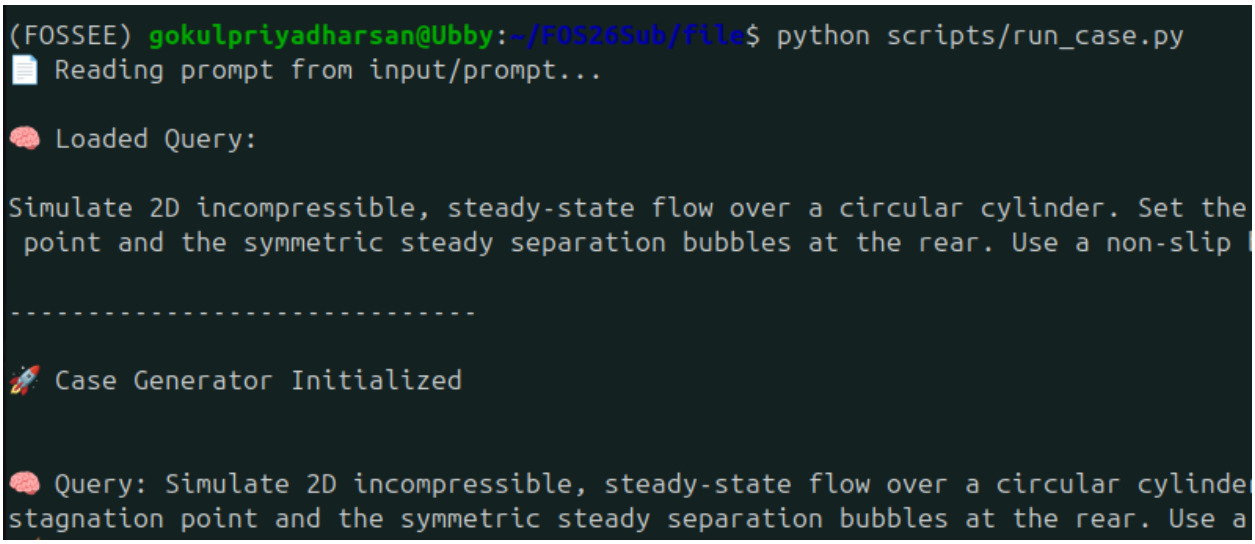

Figure 5: Initializig query

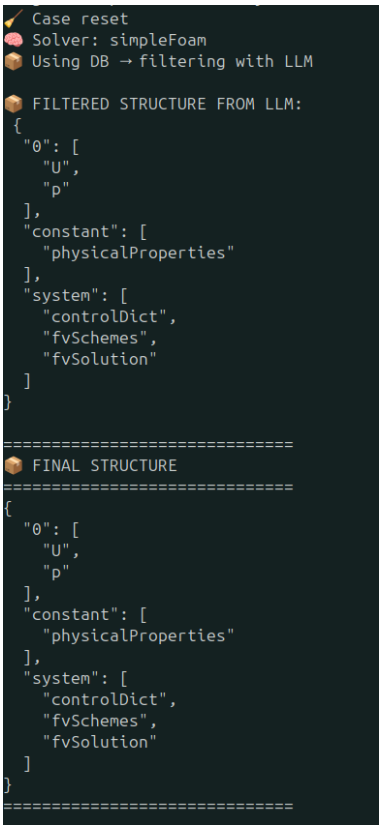

Figure 6: Deceiding structure

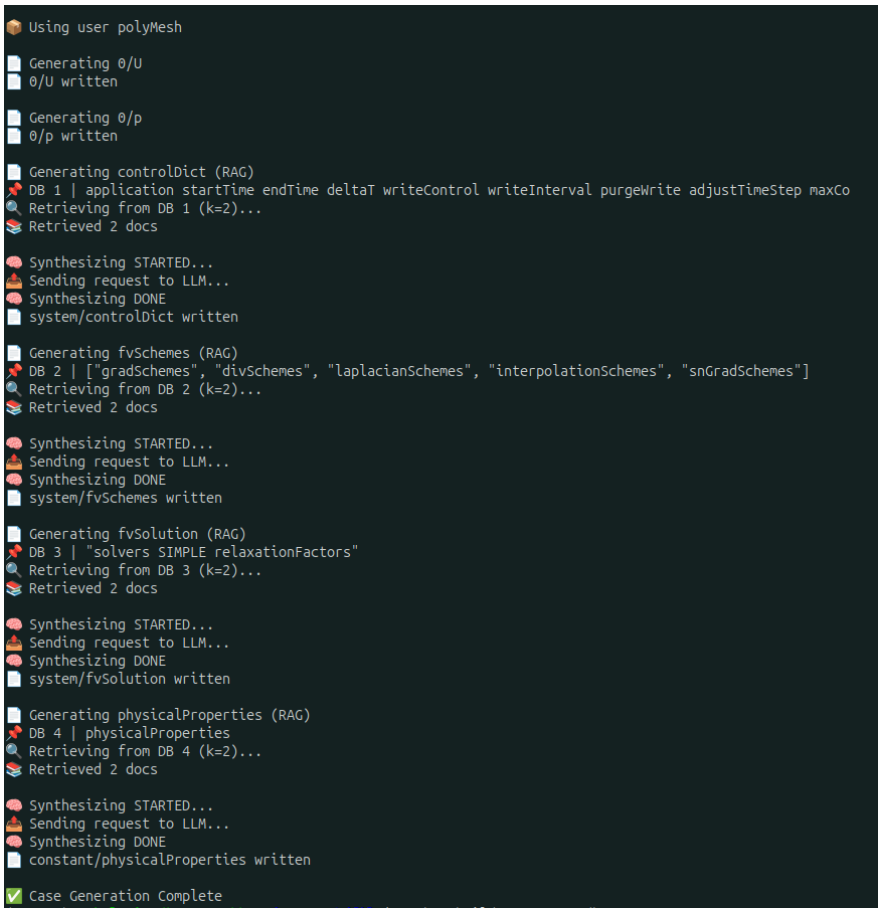

Figure 7: Case Generation

Step 6: Iterative Correction

This iterative correction workflow is executed by :

```
python scripts/run_error_loop.py
```

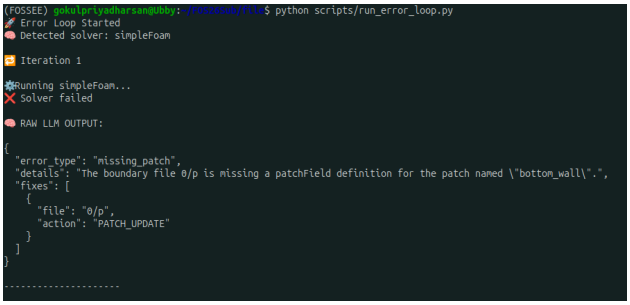

Figure 8: Error analyzer sample

The generated outputs typically provide a strong initialization template requiring only minimal manual corrections before final simulation execution.

7 Refereces

[1] AGN000. *foam-cfd-deploy*. https://github.com/AGN000/foam-cfd-deploy. GitHub repository. 2025.

[2] Ling Yue et al. "Foam-Agent 2.0: An End-to-End Composable Multi-Agent Framework for Automating CFD Simulation in OpenFOAM". In: *arXiv preprint arXiv:2509.18178* (2025). URL: https://arxiv.org/pdf/2509.18178.

[3] csml-rpi. *Foam-Agent*. https://github.com/csml-rpi/Foam-Agent. GitHub repository. 2025.

[4] Christopher J. Greenshields and Henry G. Weller. *Notes on Computational Fluid Dynamics: General Principles*. CFD Direct, 2025. URL: https://doc.cfd.direct/notes/cfd-general-principles/.