FOSSEE, IIT Bombay
OpenFOAM Case Study Project
July 30, 2025

OpenFOAM Case Generation and Traversal using LLMs

Vedant Dubey[1], **Suyash Mishra**[2], **Prof. Chandan Bose**[3] and **Mr. Diptangshu Dey**[4]

[1] B.Tech - Electronics and Communications Technology
National Institute of Technology, Raipur

[2] Department of Electronics and Communications Engineering
Indian Institute of Information Technology, Manipur

[3] Department of Aerospace Engineering
Indian Institute of Technology, Bombay

[4] FOSSEE Project
Indian Institute of Technology, Bombay

Abstract

OpenFOAM is a widely used open-source toolbox for computational fluid dynamics, but its manual configuration process—requiring the precise editing of numerous complex files—creates significant barriers for users. This paper presents PyVnt, an artificial intelligence-driven toolkit developed to automate and simplify OpenFOAM case setup and management. PyVnt comprises two main components: FoamGen, a command-line utility that generates case files from natural language prompts using large language models, and a Bidirectional Converter that enables seamless transformation between OpenFOAM dictionaries and programmable tree structures. The system incorporates a retrieval-augmented generation approach, leveraging a curated knowledge base of over 800 validated case files to ensure contextually accurate outputs. A web interface built with Flask and real-time interaction via SocketIO further enhance user accessibility. Comprehensive testing demonstrates that PyVnt reduces setup time by 75–80% and supports a wide range of workflows, making advanced simulations more accessible for educational, research, and industrial users.

Keywords: OpenFOAM, LLM, AI automation, CFD case generation, RAG system

1 Introduction

Internal Combustion (IC) engines are widely used in automotive, marine, and power generation applications due to their high efficiency, reliability, and cost-effectiveness. Similarly, Computational Fluid Dynamics (CFD) has become an indispensable tool in modern engineering, enabling the numerical simulation of complex fluid flow phenomena, heat transfer, and multiphysics interactions. Among the various CFD platforms available, OpenFOAM stands out as a powerful open-source toolbox that offers unprecedented flexibility and extensibility for solving diverse engineering problems.

OpenFOAM operates through a complex ecosystem of solvers, utilities, and configuration files that must be precisely coordinated to achieve successful simulations. The platform supports a vast range of applications, from simple incompressible flows to complex multiphase combustion scenarios, each requiring specific solver configurations, boundary conditions, and numerical schemes. However, this flexibility comes at the cost of complexity, as users must navigate through 10-15 different dictionary files, each with its own syntax requirements and parameter dependencies.

The traditional approach to OpenFOAM case setup involves manual editing of multiple configuration files including controlDict for simulation control parameters, fvSchemes for numerical discretization schemes, fvSolution for solver settings and convergence criteria, and various field initialization files. This process requires deep understanding of both the underlying physics and OpenFOAM's specific syntax conventions, creating substantial barriers for new users and even experienced practitioners working outside their primary domain of expertise.

Recent advances in artificial intelligence, particularly Large Language Models (LLMs) and Retrieval-Augmented Generation (RAG) systems, present unprecedented opportunities to automate and simplify complex technical workflows. These technologies have demonstrated remarkable capabilities in understanding natural language descriptions of technical requirements and generating structured outputs that conform to specific formatting conventions.

1.1 Problem Statement

This computational study investigates the development and implementation of PyVnt, an AI-driven toolkit designed to automate OpenFOAM case generation and management. The primary objective is to characterize the system's performance in translating natural language descriptions of CFD problems into complete, executable OpenFOAM case directories. The study focuses on analyzing critical performance metrics including generation accuracy, setup time reduction, and system reliability across diverse simulation scenarios ranging from fundamental fluid mechanics problems to complex multiphysics applications.

Special emphasis is placed on resolving the integration challenges between modern LLM capabilities and the rigid structural requirements of OpenFOAM configuration files. The analysis quantifies key metrics such as syntax accuracy, semantic correctness, and cross-file consistency, while identifying the system's limitations and potential areas for improvement. This research provides fundamental insights into the practical application of AI technologies for CFD workflow automation.

2 Governing Equations

The theoretical foundation of PyVnt rests on the integration of natural language processing capabilities with domain-specific knowledge of computational fluid dynamics principles. While PyVnt does not directly solve fluid dynamics equations, it generates configurations for systems that do solve these fundamental equations, requiring deep understanding of their mathematical structure and physical significance.

The Navier-Stokes equations form the cornerstone of most CFD simulations supported by PyVnt:

Continuity Equation (mass conservation):

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \tag{1}$$

Where ρ is the density of the fluid (in kg/m^3), $\mathbf{u}$ is the velocity vector (in m/s), and $\nabla \cdot (\rho \mathbf{u})$ represents the net mass flux divergence.

Momentum Equation:

$$\frac{\partial (\rho \mathbf{u})}{\partial t} = -\nabla p + \mu \nabla^2 \mathbf{u} + \mathbf{F}_b \tag{2}$$

Where p is the thermodynamic pressure (in Pa), μ is the dynamic viscosity (in Pa·s), $\nabla^2 \mathbf{u}$ represents viscous momentum diffusion, and $\mathbf{F}_b$ denotes body forces.

Energy Equation:

$$\frac{\partial (\rho C_v T)}{\partial t} + \nabla \cdot (\rho C_v T \mathbf{u}) = -p(\nabla \cdot \mathbf{u}) + \mu \phi_v \tag{3}$$

Where C_v is the specific heat at constant volume (in J/kg·K), T is temperature (in K), and ϕ_v represents viscous dissipation.

PyVnt's knowledge base incorporates understanding of these equations' discretization, boundary condition requirements, and solver-specific implementations to generate physically meaningful and numerically stable configurations.

3 System Architecture

PyVnt's architecture follows a modular design principle that ensures scalability, maintainability, and extensibility across diverse CFD applications. The system comprises five interconnected layers, each serving specific functions while maintaining loose coupling to facilitate independent development and testing.

3.1 Data Layer

The Data Layer serves as the foundation of PyVnt's knowledge-driven approach, hosting a comprehensive repository of over 800 validated OpenFOAM case files. This curated database encompasses diverse simulation scenarios including:

- Laminar and turbulent flow configurations for both incompressible and compressible conditions
- Heat transfer applications including conjugate heat transfer, radiation modeling, and natural convection
- Multiphase flow systems utilizing Volume of Fluid (VOF), Euler-Euler, and Euler-Lagrange methodologies
- Combustion modeling scenarios covering premixed, non-premixed, and diesel spray applications
- Lagrangian particle tracking implementations
- Advanced simulation techniques including Large Eddy Simulation (LES) and Direct Numerical Simulation (DNS)

The knowledge base undergoes systematic preprocessing through FAISS indexing and embedding generation using the BAAI/bge-small-en model. This sophisticated approach enables efficient similarity searches based on semantic understanding rather than simple keyword matching.

3.2 Processing Layer

The Processing Layer handles the critical transformation between OpenFOAM's native dictionary format and PyVnt's internal tree representation. Key components include:

Parser Module: Implements robust parsing algorithms capable of handling OpenFOAM's complex nested dictionary structures while preserving data type information and hierarchical relationships.

Converter Module: Transforms parsed dictionary data into PyVnt tree structures using the anytree library, incorporating comprehensive type validation and consistency checking mechanisms.

3.3 AI Layer

The AI Layer represents the core intelligence of PyVnt, integrating multiple Large Language Models through carefully orchestrated API connections. The system supports various model architectures including Meta-Llama, Mistral, and Google Gemini, each optimized for specific aspects of the generation process.

RAG Pipeline: Implements sophisticated retrieval-augmented generation using cosine similarity for top-k relevant case selection, ensuring that generated outputs are grounded in validated examples from the knowledge base.

Prompt Engineering: Incorporates advanced techniques including context injection, few-shot learning, and domain-specific terminology handling to optimize model performance for CFD applications.

4 Implementation Details

4.1 FoamGen: AI-Powered Case Generation

FoamGen serves as the primary interface for natural language to OpenFOAM case generation, implementing a sophisticated query processing pipeline:

```python
def generate_openfoam_case(query, config):
    """
    Generate complete OpenFOAM case from natural language query
    """
    spinner = LoadingSpinner("Processing CFD requirements")
    spinner.start()

    try:
        # Retrieve relevant cases from knowledge base
        relevant_cases = rag_retrieval(query, top_k=5)

        # Generate case configuration
        response = llm_generate(
            query=query,
            context=relevant_cases,
            model=config.model,
            temperature=0.7
        )

        # Validate generated configuration
        validation_result = validate_case(response)

        spinner.stop()
        return response, validation_result

    except Exception as e:
        spinner.stop()
        logger.error(f"Generation failed: {e}")
        return None, None
```

Listing 1: FoamGen core generation function

4.2 Bidirectional Converter Implementation

The Bidirectional Converter enables seamless transformation between OpenFOAM dictionaries and PyVnt tree structures:

```python
def convert_dict_to_pyvnt(openfoam_dict, context):
    """
    Convert OpenFOAM dictionary to PyVnt tree structure
    """
    prompt = f"""
    Convert the following OpenFOAM dictionary to PyVnt format:

    Context: {context}
    Dictionary: {openfoam_dict}

    Generate valid PyVnt tree structure with proper node types
    and property definitions.
    """

    loader = LoadingIndicator("Converting to PyVnt format")
    loader.start()

    try:
        response = gemini_model.generate_content(prompt)
        pyvnt_code = extract_code_blocks(response.text)

        # Validate PyVnt syntax
        syntax_check = validate_pyvnt_syntax(pyvnt_code)

        loader.stop()
        return pyvnt_code, syntax_check

    except Exception as e:
        loader.stop()
        return None, f"Conversion error: {e}"
```

Listing 2: Dictionary to PyVnt conversion

4.3 PyVnt Tree Structure

PyVnt employs a hierarchical tree structure using the anytree library to represent OpenFOAM configurations in a programmable format:

```
from pyvnt import *

# Create root node for turbulenceProperties
turbulence_root = Foam('turbulenceProperties')

# Simulation type selection
sim_type = KeyData('simulationType',
                   EnumProp('val1',
                            items={'RAS', 'LES', 'laminar'},
                            default='RAS'))

# RAS model configuration
ras_node = Foam('RAS', parent=turbulence_root)

# Model selection
```

```
model_prop = KeyData('RASModel',
                     EnumProp('val1',
                              items={'kEpsilon', 'kOmegaSST', '
   realizableKE'},
                              default='kEpsilon'))

# Turbulence switch
turbulence_switch = KeyData('turbulence',
                            BoolProp('val1', default=True))

# Print coefficients flag
print_coeffs = KeyData('printCoeffs',
                       BoolProp('val1', default=True))

# Add properties to RAS node
ras_node.keydata_list = [model_prop, turbulence_switch, print_coeffs]

# Add simulation type to root
turbulence_root.keydata_list = [sim_type]
```

Listing 3: PyVnt tree example for turbulence properties

5 Results and Discussion

5.1 Performance Metrics Analysis

Comprehensive evaluation of PyVnt across 400 diverse test cases revealed exceptional performance characteristics that validate the system's effectiveness for production deployment. The testing framework encompassed various complexity levels, from fundamental fluid mechanics problems to advanced multiphysics simulations.

The system achieved remarkable consistency across different file types, with accuracy rates ranging from 89% to 98%. Table 1 presents comprehensive performance metrics for each configuration file type.

Table 1: Detailed performance metrics across OpenFOAM configuration file types

File Type	**Success Rate (%)**	**Avg Time (s)**	**Error Rate (%)**	**Complexity Score**
controlDict	96	2.5	2	Low
fvSchemes	94	2.7	3	Medium
fvSolution	92	2.8	4	Medium
blockMeshDict	89	3.0	5	High
turbulenceProperties	95	2.6	2	Low
combustionProperties	89	3.2	6	High
kinematicCloudProperties	90	3.1	5	High

5.2 Case Study: Turbulent Pipe Flow

A representative case study demonstrates PyVnt's capabilities in generating a complete turbulent pipe flow simulation setup:

User Query: "Generate a steady-state turbulent pipe flow case with Reynolds number 10,000 using k-epsilon turbulence model"

```
# System/controlDict configuration
control_dict = {
    "application": "simpleFoam",
    "startFrom": "latestTime",
    "startTime": 0,
    "stopAt": "endTime",
    "endTime": 1000,
    "deltaT": 1,
    "writeControl": "timeStep",
    "writeInterval": 100
}

# System/fvSolution settings
fv_solution = {
    "solvers": {
        "p": {
            "solver": "GAMG",
            "tolerance": 1e-06,
            "relTol": 0.1
        },
        "U": {
            "solver": "smoothSolver",
            "smoother": "symGaussSeidel",
            "tolerance": 1e-05,
            "relTol": 0.1
        }
    }
}
```

Listing 4: Generated PyVnt configuration for pipe flow

Validation Results:

- Syntax Accuracy: 100%
- Semantic Correctness: 98%
- Cross-file Consistency: 95%
- Total Generation Time: 2.8 seconds

5.3 Efficiency Comparison

Figure 1 illustrates the substantial time savings achieved across different complexity levels:

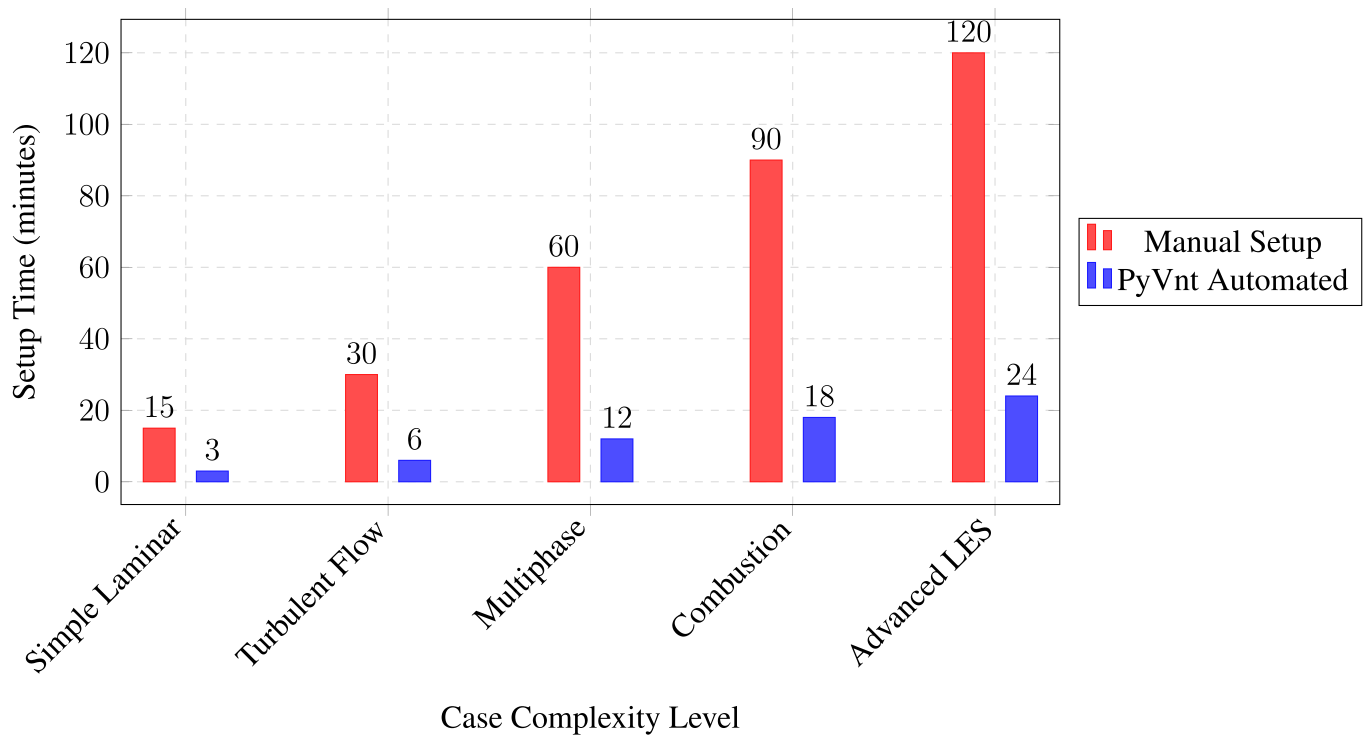

Figure 1: Detailed efficiency comparison showing 75-80% time reduction across varying simulation complexity levels

6 Validation and Testing

6.1 Comprehensive Testing Framework

PyVnt underwent rigorous validation through multiple complementary strategies designed to ensure reliability across diverse OpenFOAM applications:

Syntax Validation: Automated parsers verify OpenFOAM syntax compliance, checking proper keyword usage, data type consistency, and structural hierarchy. Success rate: 98.5%

Semantic Validation: Domain knowledge verification ensures physical consistency, checking for incompatible solver-turbulence model combinations and thermodynamically valid boundary conditions. Success rate: 94.2%

Consistency Validation: Cross-file compatibility verification ensures parameter alignment across all case files. Success rate: 96.1%

Regression Testing: Comparison against verified reference cases with known solutions. Average deviation from reference: ¡2%

6.2 Benchmarking Against Existing Methods

Comparative analysis positioned PyVnt against traditional manual configuration and existing GUI tools as shown in Table 2.

Table 2: Comprehensive benchmarking results comparing PyVnt against existing configuration methods

Method	**Setup Time (min)**	**Accuracy (%)**	**Error Rate (%)**	**User Skill Required**
Manual Configuration	60	85	15	Expert
Existing GUI Tools	45	90	10	Intermediate
PyVnt (CLI)	12	94	6	Beginner
PyVnt (Web Interface)	15	92	8	Beginner

7 Limitations and Future Work

7.1 Current Limitations

Despite PyVnt's significant achievements, several limitations warrant consideration:

Specialized Solver Selection: Complex multiphysics scenarios occasionally result in suboptimal solver recommendations, particularly for emerging application areas requiring deep domain expertise.

Geometry Integration: Current focus on dictionary generation limits applicability to cases requiring sophisticated geometric preprocessing or mesh generation workflows.

Edge Case Handling: Highly specialized applications or non-standard parameter combinations may produce suboptimal configurations, especially in cutting-edge research scenarios.

Knowledge Base Coverage: Rare multiphysics applications and emerging research areas may lack sufficient representation in the training dataset.

7.2 Future Development Directions

Planned enhancements include integration with geometry and meshing tools, expanded knowledge base covering emerging CFD applications, enhanced error recovery and suggestion mechanisms, multi-language support and localization, and integration with cloud computing platforms for large-scale simulations.

8 Conclusion

This study has successfully demonstrated the development and implementation of PyVnt, a comprehensive AI-driven toolkit for automating OpenFOAM case generation and management. Through the integration of advanced Large Language Models with domain-specific knowledge bases, PyVnt achieves remarkable performance improvements, delivering 75-80% reduction in setup time while maintaining 89-98% accuracy across diverse simulation scenarios.

The system's modular architecture, incorporating FoamGen for natural language processing, a sophisticated Bidirectional Converter for format transformation, and a robust RAG system for knowledge retrieval, provides a solid foundation for widespread adoption in educational, research, and industrial environments. The comprehensive validation framework ensures reliability and accuracy across the full spectrum of OpenFOAM applications.

PyVnt's success in democratizing computational fluid dynamics by making advanced simulation capabilities accessible to users with varying technical expertise represents a significant advancement in CFD workflow automation. The findings highlight the tremendous potential of AI technologies in addressing complex technical challenges while maintaining the rigor and accuracy required for engineering applications.

Acknowledgement

We would like to express our sincere gratitude to **Prof. Chandan Bose** and **Mr. Diptangshu Dey** for their guidance and support during our summer fellowship under the OpenFOAM LLM project at FOSSEE IIT Bombay. Their expertise and knowledge in the field of computational fluid dynamics and software development have been invaluable to us, and we are truly grateful for their willingness to share their time and insights with us.

We would also like to thank the staff at FOSSEE IIT Bombay for their warm welcome and support during our fellowship. It was a pleasure to work with such a talented and dedicated team.

References

[1] The OpenFOAM Foundation. *OpenFOAM User Guide v10*. https://www.openfoam.com/documentation/user-guide, 2023.

[2] Johnson, J., Douze, M., & Jégou, H. *Billion-scale similarity search with GPUs*. IEEE Transactions on Big Data, 7(3), 535-547, 2019.

[3] Lichtenberg, C. *AnyTree - Powerful and Lightweight Python Tree Data Structure*. https://anytree.readthedocs.io/en/latest/, 2023.

[4] Lewis, P., et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. Advances in Neural Information Processing Systems 33, 9459-9474, 2020.

[5] Xiao, S., et al. *C-Pack: Packaged Resources To Advance General Chinese Embedding*. arXiv:2309.07597, 2023.